Ein Rahmenwerk für die qualitative Analyse der Paarprogrammierung

Stephan Salinger

Dissertation
zur Erlangung des Grades
Doktor der Naturwissenschaften (Dr. rer. nat.)
am Fachbereich für Mathematik und Informatik
der Freien Universität Berlin

Berlin 2013

Datum der Disputation: ***16.05.2013***

Dekan des Fachbereichs Mathematik und Informatik:
Prof. Dr. Rupert Klein

Gutachter:
Prof. Dr. Lutz Prechelt, *Freie Universität Berlin, Deutschland*
Prof. Dr. Yvonne Dittrich, *IT University of Copenhagen, Dänemark*

Für Ute

DANKSAGUNGEN

Mein besondere Dank gilt den folgenden Personen, ohne deren Beitrag diese Arbeit nicht möglich gewesen wäre:

- Lutz Prechelt für die Betreuung und die vielen hilfreichen Anmerkungen.
- Christopher Özbek und Franz Zieris für die unzähligen Diskussionen über die Methode und die Ergebnisse.
- Laura Plonka für die Durchführung von Paarkodierung.
- Sebastian Jekutsch für die Inspirationen zu Beginn der Arbeit.
- Ute Schulten, Gesine Milde, Susanne Halbekath und Eva Neumann für das unermüdliche Korrekturlesen und Ermuntern.
- Meinen Eltern und Geschwistern, für die Geduld und Liebe mit der sie die lange Phase der Konzentration ertragen haben.
- Den Studierenden und professionellen Entwicklern, die sich bereit gefunden haben, an Paarprogrammierungsaufzeichnungen teilzunehmen.
- Und allen dafür, dass sie mich stets ermutigt haben, weiter zu machen.

INHALTSVERZEICHNIS

TABELLENVERZEICHNIS

ABBILDUNGSVERZEICHNIS

VERZEICHNIS DER EXKURSE

Kapitel

1 Einführung

NEU ANFANGEN. Ja!, noch einmal anfangen, ganz anders.
Rainald Goetz, Irre

Bei der *Paarprogrammierung* (PP, *pair programming*) handelt es sich um eine Programmierpraktik, die von Williams et al. [219] wie folgt beschrieben wird:

> "In pair-programming, two programmers jointly produce one artifact (design, algorithm, code, etc.). The two programmers are like a coherent, intelligent organism working with one mind, responsible for every aspect of this artifact. One partner is the *driver* and has control of the pencil/mouse/keyboard and is writing the design or code. The other person [(the *observer*[1])] continuously and actively observes the work of the driver – watching for defects, thinking of alternatives, looking up resources, and considering strategic implications of the work at hand."

Obwohl die PP in einer Vielzahl wissenschaftlicher Untersuchungen analysiert wurde, muss festgehalten werden, dass ihr Nutzen und ihre Funktionsweise bisher nur unzureichend verstanden ist. Die Ergebnisse der bisherigen Studien ergeben vielmehr ein in vielen Belangen uneinheitliches und insgesamt unvollständiges Bild (Details hierzu folgen in Kapitel 2). Hier setzt die vorliegende Arbeit an. Allerdings nicht, indem neue Studien zu einzelnen Effekten der Praktik (Qualitätsverbesserung, Wissenstransfer etc.) präsentiert werden, sondern dadurch, dass ein Fundament für konsolidierte (qualitative) Analysen der PP gelegt wird. So werden mit dem so genannten „Rahmenwerk für die qualitative Analyse der PP" Methoden und Werkzeuge zur Verfügung gestellt, mit denen die PP aus unterschiedlichen Blickwinkeln und mit unterschiedlichen Intentionen untersucht und somit letztendlich die Funktionsweise als Ganzes verstanden und für spezifische Einsatzszenarien optimiert werden kann.

Die vorliegende Arbeit liefert somit einen Beitrag zum Softwareengineering, welches sich nach Helmut Balzert [11, S. 36] mit der „zielorientierten Bereitstellung und systematischen Verwendung von Prinzipien, Methoden und Werkzeugen für die arbeitsteilige, ingenieurmäßige Entwicklung und Anwendung von umfangreichen Software-Systemen" beschäftigt, wobei durch die Betonung der Zielorientierung die notwendige Berücksichtigung von Kosten, Zeit und Qualität hervorgehoben wird.

[1] Der Partner des Drivers wird in vielen Veröffentlichungen nicht Observer sondern *Navigator* genannt.

Unter *Qualität* wird in diesem Zusammenhang eine Menge operationalisierbarer Maße verstanden. Um diese systematisch aufstellen zu können, wird im Allgemeinen zwischen *Produkt-* und *Prozessqualität* unterschieden. Erstere bezieht sich auf Eigenschaften des an den Kunden übergebenen Ergebnisses, im Allgemeinen also auf Eigenschaften eines Softwarepakets, zweitere auf die Eigenschaften des Softwareentwicklungsprozesses, der dieses Endprodukt hervorgebracht hat. Hierbei versteht man unter einem *Softwareentwicklungsprozess* (oder auch kurz *Softwareprozess*) einen (konkreten) „Satz von Tätigkeiten und damit zusammenhängenden Ergebnissen [(Artefakten)], durch die ein Softwareprodukt entsteht“ [193, S. 23]. In Abgrenzung hierzu ist ein *Softwareprozessmodell* (oder auch *Vorgehensmodell*) „eine vereinfachte Beschreibung eines Softwareprozesses, die von einer bestimmten Perspektive [(z.B. der Architektur des Prozesses)] aus dargestellt wird“ [193, S. 24].

Betrachtet man die Produktqualität (Softwarequalität), kann man zwei Gruppen von Qualitätsindikatoren ausmachen. Zum einen soll das Endprodukt die an es gestellten Anforderungen – z.B. bez. Funktionalität, Zuverlässigkeit, Benutzbarkeit und Effizienz (siehe [10, S. 258 ff]) – erfüllen (*Brauchbarkeit*) und zum anderen soll es derart konstruiert sein, dass neue Anforderungen möglichst einfach in neue Funktionalitäten umgesetzt und anschließend integriert werden können (*Wartbarkeit*). Softwarequalität steht dabei in Hinsicht auf die verfügbare Produktivität (des eingesetzten Entwicklungsteams) im Widerstreit mit den oben genannten Faktoren Kosten und Zeit sowie dem Umfang („Teufelsquadrat“ [96]). Dies führt dazu, dass Qualitätsziele für die einzelnen Produktqualitätsmerkmale festgelegt werden müssen.

Im Allgemeinen wird davon ausgegangen, dass es eine direkte Korrelation zwischen der Softwarequalität (also der Brauchbarkeit und der Wartbarkeit in Hinsicht auf die festgelegten Qualitätsziele) und der Qualität des verwendeten Softwareentwicklungsprozesses gibt [48; 67]. Aus diesem Grund wurden in den letzten 40 Jahren unterschiedlichste Prozesse bzw. Prozessmodelle, die die Softwareentwicklung systematisieren und somit unter anderem weniger fehleranfällig machen sollen, vorgestellt und zur Anwendung gebracht. Das Ziel dieser Entwicklungen war dabei neben der Verbesserung der Produktqualität auch die Senkung der Kosten und des benötigten Zeitaufwandes.

Die entwickelten Lösungen reichen von plangetriebenen Verfahren wie dem Wasserfallmodell [23; 178] über risikogeleitete wie dem Spiralmodell [24; 25] bis hin zu agilen Methoden[2] wie eXtreme Programming (XP [12; 14]). Flankiert wurden diese Verfahren durch Taxonomien für Qualitätsstandards wie dem Capability Maturity Model Integration (CMMI[3]), prozesszentrierte Entwicklungsumgebungen (Process-centered Software Engineering Environments (PSEEs), siehe z.B. [4]) und die Einsicht, dass es nicht den universellen Softwareprozess bzw. das universelle Softwareprozessmodell gibt bzw. geben kann [22]. Prozessmodelle müssen

[2] Siehe „Manifesto for Agile Software Development“ unter http://agilemanifesto.org (Abruf: 08.11.2012)

[3] http://www.sei.cmu.edu/cmmi/ (Abruf: 26.10.2012))

Exkurs 1: *Wir praktizieren keine Paarprogrammierung – eine Anekdote*
Im Rahmen der Bemühungen Paarprogrammierungssitzungen im industriellen Umfeld aufzeichnen zu können, wurde zu Beginn der vorliegenden Arbeit Kontakt zu einem mittelständischen Unternehmen in Ostdeutschland aufgenommen, welches sich auf die Entwicklung von Software im Bereich Wissensmanagement spezialisiert hatte. Obwohl uns der Entwicklungsleiter telefonisch versicherte, dass bei ihnen bisher keine PP zum Einsatz kommt, wurden wir zu einem Gespräch eingeladen. Es sollten Möglichkeiten anderweitiger Kooperationen ausgelotet werden. Da der Entwicklungsleiter bei unserer Ankunft in der Entwicklungsabteilung noch in einem Gespräch war, wurden wir gebeten kurz zu warten. Wir nahmen also zwischen den Entwicklern Platz. Es vergingen kaum 5 Minuten und wir konnten beobachten wie einer der Entwickler einen Kollegen herbei rief. Es schien um ein gerade festgestelltes Fehlverhalten der in Entwicklung befindlichen Software zu gehen. Der Herbeigerufene unterbrach seine Arbeit und setzte sich neben seinen Kollegen. Nun konnten wir beobachten, wie beide Entwickler lebhaft diskutierten und abwechselnd die Tastatur bedienten. Nach ca. 25 Minuten wurden wir vom Entwicklungsleiter abgeholt. Die „Pair Debugger“ [218, S. 28] oder allgemeiner Paarprogrammierer arbeiteten zu diesem Zeitpunkt immer noch zusammen an einem Rechner.

vielmehr entsprechend den gegebenen Rahmenbedingungen (z.B. bezüglich einer konkreten Projektsituation) ausgewählt, ergänzt bzw. beschnitten und in der Regel auch kontinuierlich verbessert werden.

Mit ihrem Fokus auf die Praktik der PP zielt auch die vorliegende Arbeit auf die gerade beschriebene Optimierung von Softwarequalität, Kosten und Zeitaufwand durch Prozessverbesserung. Sie ist hierbei, auch wenn die PP eine der Kernpraktiken des XP-Prozesses [14, S. 42] bildet, keineswegs ausschließlich im Bereich der agilen Softwareentwicklung angesiedelt. Im Kern geht es bei der PP nämlich „nur“ darum, dass Artefakte, in der Regel Programmkode, nicht von einem Entwickler allein, sondern von zwei Entwicklern gemeinsam in einem in direkter, unmittelbarer Zusammenarbeit zu absolvierenden Schritt erstellt oder bearbeitet werden. Dieses Vorgehen ist weder an agile Methoden gebunden noch neu. Eine Reihe anekdotischer Berichte (siehe z.B. [218, S. 8]) weist vielmehr darauf hin, dass derartige Formen der Zusammenarbeit schon seit Jahrzehnten praktiziert werden, wenn auch nicht immer explizit unter diesem Namen (siehe auch Exkurs 1). Williams und Kessler [218, S. 8] zitieren diesbezüglich unter anderem eine Aussage von Fred Brooks, dem Autor von „The Mythical Man-Month“ [32]:

> „Fellow graduate student Bill Wright and I first tried pair programming when I was a grad student (1953-56). We produced 1500 lines of defect-free code; it ran correctly first try“

Wenn also auch die PP beim Aufkommen von agilen Entwicklungsmethoden wie XP Mitte/Ende der 1990er Jahre keine neue Praktik darstellte, war ihr organisierter oder gar obligatorischer Einsatz im Rahmen von Softwareprozessen bis

***Exkurs 2:** Makro- und Mikroprozessforschung*
Leon J. Osterweil [157] unterscheidet zwischen zwei Formen der Erforschung von Softwareprozessen. Die eine beschäftigt sich mit so genannten *Makroprozessen* (*macroprocess research*), die andere mit so genannten *Mikroprozessen* (*microprocess research*). Er beschreibt diese Formen und deren Zusammenhang wie folgt:

> „We propose that macroprocess research be characterized as investigations whose focus is on the study of the external behaviors of processes. Such behaviors include the speed of execution of processes, the characteristics of the software products they produce, the way in which resource infusion affects product nature and process speed, and the effect of changes in production timetables upon products, and the processes themselves. [...] We propose that microprocess research be characterized as investigations that focus on investigation of the precise specification of the details of software processes, for the purpose of inferring how those details effect the external behaviors of the processes. [...] A key goal of microprocess research is to provide detailed, accurate, low- level definitions of processes, and reasoning capabilities that are able to predict and explain the high level phenomena discovered by macroprocess investigations."

dahin eher unüblich. Dementsprechend rückten die propagierten Vorteile der PP, wie z.B. verbesserte Qualität der Ergebnisse, gesteigerte Performance, optimierter Wissenstransfer oder mehr Spaß an der Arbeit sowie auch die befürchteten Nachteile, wie z.B. erhöhte Kosten und zu überwindende Widerstände bei den Entwicklern, erst zu diesem Zeitpunkt in den Fokus des Interesses von Industrie und Forschung. Nachfolgend wurden sie in zahlreichen Veröffentlichungen diskutiert und – größtenteils mit quantitativen Methoden – analysiert. In Kapitel 2 wird auf diese Studien näher eingegangen. Wie eingangs angesprochen ergibt sich aus den Resultaten dieser Untersuchungen kein einheitliches Bild – insbesondere nicht in Hinsicht auf viele der propagierten Vor- und Nachteile. Der Großteil der bisherigen Studien liefert zwar Zahlen, z.B. bez. der Performance der Entwickler oder der resultierenden Produktqualität, kann aber weder deren Zustandekommen noch Diskrepanzen zu anderen vergleichbaren Studien hinreichend erklären. Der Hauptgrund hierfür besteht darin, dass die PP in der Regel als „Black Box" betrachtet wird und die interne Prozessstruktur, der so genannte Mikroprozess [157] (siehe auch Exkurs 2) der PP außen vor bleibt. Anders ausgedrückt: Anstatt zu versuchen, die Funktionsweise der PP zu verstehen, haben viele der bisherigen Studien „nur" Anstrengungen dahingehend unternommen, Effekte der PP zu messen.

An dieser Stelle setzen die im vorliegenden Dokument dargestellten Ergebnisse an, indem – wie oben schon angesprochen – methodische wie auch inhaltliche Grundlagen für eine konsolidierte Erforschung des Mikroprozesses der PP, also für ein Verstehen der PP als Ganzes zur Verfügung gestellt werden. Diese Ergebnisse wurden direkt mittels qualitativer Analysen von Paarprogrammierungssitzungen gewonnen. Da zu Beginn dieser Analysen nur in sehr geringem Maß auf vorhandene wissenschaftliche Erkenntnisse über den Paarprogrammierungsprozess an und

für sich und somit auch nicht auf weitreichende Erfahrungen in Hinsicht auf diesbezüglich geeignete Forschungsmethoden zurückgegriffen werden konnte, wurden für die Arbeit zwei zentrale Teilziele festgelegt:

1. Synthese einer speziell auf die Analyse des Paarprogrammierungsprozesses zugeschnittenen (qualitativen) Untersuchungsmethode.

2. Entwicklung eines adaptiven Systems von Konzepten und Anwendungsregeln, um die den Prozess der PP formenden grundlegenden Aktivitäten der einzelnen Paarmitglieder (später *Basisaktivitäten* genannt) beschreiben bzw. konzeptualisieren zu können. Es wurde später *Basisschicht* (BS) genannt.

Mit der Eigenschaft „adaptiv“ wurde betont, dass die Elemente der BS in nachgelagerten, partiell aufeinander aufbauenden Untersuchungen entsprechend aktuellen Interessenschwerpunkten angepasst werden können bzw. müssen – z.B. um sie in einem bestimmten größeren Zusammenhang, wie eine Theorie über den Wissenstransfer bei der PP, bringen zu können. Es war also nicht das Ziel, die BS dahingehend auszuarbeiten, dass sie direkt komplexe Einsichten in den Prozess der PP liefern kann. Die BS sollte es vielmehr „nur“ ermöglichen, bei weiterführenden, in so genannten *Schichten* (*layers*) organisierten Untersuchungen nicht bei Null beginnen zu müssen, sondern stattdessen auf die Möglichkeit zurückgreifen zu können, das System von Konzepten, später *Basiskonzeptmenge* (BKM) genannt, auszudifferenzieren, zu verallgemeinern, zu erweitern, zu beschneiden, zu ergänzen oder zum Zweck der Verbesserung der theoretischen Sensibilität einzusetzen (siehe auch Abbildung 1.1). Anders ausgedrückt: Mit der BS sollte eine „Sprache über die PP“ geschaffen werden, die in möglichst vielen nachfolgenden qualitativen Untersuchungen der PP verwendet werden kann. Eine Kernaufgabe bei der Herleitung der BS bestand somit darin zu charakterisieren, was unter einer Basisaktivität in Hinsicht auf Inhalt und Granularität verstanden werden soll. Außerdem wurde festgeschrieben, dass die BKM nicht unter Zuhilfenahme/Verwendung von allgemeinen Theorien oder bekannten Kodierschemata postuliert werden darf, sondern direkt aus Paarprogrammierungssitzungsdaten zu extrahieren ist. Mit dieser Vorgabe sollte verhindert werden, dass den zu analysierenden Daten wenig geeignete Theorien „aufgepfropft“ werden. Die BS sollte vielmehr derart in Paarprogrammierungsdaten verankert werden, dass mit ihrer Hilfe später tatsächlich auf die PP passende Theorien entwickelt werden können (siehe hierzu auch die Diskussion „Emergence vs. Forcing“ in [105]).

An dieser Stelle muss erwähnt werden, dass die angestrebten Ziele weder unabhängig voneinander noch von Anfang an in genau dieser Art und Weise expliziert wurden. So ist zu Beginn der Untersuchungen nicht ausdrücklich über eine speziell auf den Paarprogrammierungsprozess zugeschnittene Untersuchungsmethode nachgedacht worden. Vielmehr wurde davon ausgegangen, dass es möglich sein muss, sich einem mehr oder weniger ganzheitlichen Verständnis der PP auf direktem Weg nähern zu können – und zwar unter Verwendung der Grounded

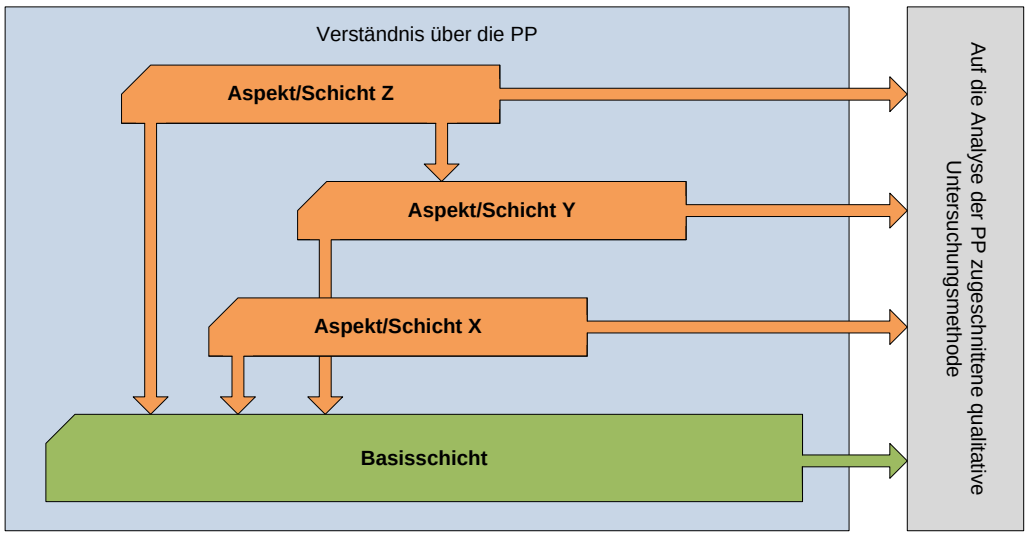


Abb. 1.1: Schematische Darstellung des Schichtenmodells zur Analyse der Paarprogrammierung: Im Rahmen der vorliegenden Arbeit wird ein so genanntes Rahmenwerk für die qualitative Analyse der PP vorgestellt. Es besteht einerseits aus einer auf die Analyse der PP zugeschnittenen Untersuchungsmethode (rechts; grau) und andererseits aus der so genannten Bassisschicht (unten; grün) – einem System von Konzepten und Anwendungsregeln, um die den Prozess der PP formenden grundlegenden Aktivitäten der einzelnen Paarmitglieder beschreiben zu können. Schon die Basisschicht selbst wurde unter Verwendung der auf die PP zugeschnittenen Untersuchungmethode entwickelt. Auch alle weiteren (qualitativen) Untersuchungen der PP sollten auf diese Methode zurückgreifen und in Schichten (mitte, orange) vorgenommen werden – sich also jeweils auf bestimmte Teilaspekte (z.B. den Wissenstransfers bei der PP) fokussieren. Die Basisschicht ist derart konstruiert, dass sie bei der Herleitung der weiteren Schichten als Ausgangspunkt verwendet werden kann. Idealerweise können Erkenntnisse, die bei der Analyse einer Schicht erlangt werden bei der Herleitung einer weiteren einfließen. Somit wird eine konsolidierte und effiziente Erforschung der PP möglich gemacht, die letztendlich zu einem umfassenden Verständnis über die Funktionsweise und Auswirkungen der PP führt. Die Pfeile in der Abbildung zeigen beispielhaft, in welcher Weise bei der Entwicklung einzelner Schichten auf andere Schichten und die Untersuchungsmethode zurückgegriffen wird.

Theory Methodology nach Strauss und Corbin [198]. Dieser Ansatz schlug aber mit der Einsicht fehl, in den komplexen Videoaufzeichnungen von Paarprogrammierungssitzungen (Ton + Webcam-Aufzeichnung der Probanden + Desktopaufzeichnung) verloren gegangen zu sein (Details hierzu folgen in Abschnitt 5.1). Als Konsequenz wurde einerseits die Grounded Theory durch speziell auf die identifizierten Probleme zugeschnittene Praktiken ergänzt und andererseits von dem Versuch abgerückt, den Prozess der PP „in einem Rutsch“ ergründen zu wollen. Stattdessen wurde – wie bereits oben erläutert – ein Vorgehen beschlossen, bei dem partiell aufeinander aufbauende Schichten betrachtet werden. Außerdem wurde festgelegt, dass in einem ersten Schritt die Entwicklung einer Basisschicht

erfolgen muss (siehe Punkt 2 oben) und Untersuchungen spezialisierter Schichten, z.B. qualitative Analysen zum Wissenstransfer, zur Rollenverteilung oder zu Mechanismen der Qualitätssicherung, nachgelagerten – zum Zeitpunkt der Veröffentlichung der vorliegenden Arbeit zum Teil bereits begonnenen – Studien (siehe auch Kapitel 12) überlassen werden sollen.

Der weitere Verlauf der Arbeit gestaltet sich wie folgt:

- Zuerst wird im zweiten Kapitel auf verwandte Arbeiten eingegangen. Hierbei liegt das Augenmerk – wie bereits angekündigt – vor allem auf der Tatsache, dass sich aus den bisher durchgeführten quantitativen Untersuchungen der PP in vielerlei Hinsicht kein konsistentes Bild ergeben hat. Darüber hinaus werden in diesem Kapitel Ergebnisse bisher durchgeführter qualitativer Studien erläutert. Auf verwandte Kodierschemata wird in diesem Zusammenhang noch nicht detaillierter eingegangen. Dies hat zwei Gründe: Zum einen wurde derartigen Schemata bei der Herleitung der Basisschicht keine Aufmerksamkeit geschenkt – vor allem deshalb, um nicht Gefahr zu laufen, die Daten aus einem evtl. ungeeigneten Blickwinkel zu betrachten. Zum anderen kann auf solche Schemata im Rahmen der vorliegenden Arbeit gezielter eingegangen werden, wenn sie unter Berücksichtigung der Basisschicht betrachtet werden.[4] So werden verwandte Kodierschemata erst gegen Ende der Arbeit in Abschnitt 11.3 erörtert.

- Nach der Vorstellung der verwandten Arbeiten wird in Kapitel 3 in die Grounded Theory Methodology – sowohl nach Glaser und Strauss [76] wie auch nach Strauss und Corbin [198] – eingeführt. Hierbei wird auch erläutert, aus welchen Gründen diese Forschungsmethodik (genau genommen die Methodik nach Strauss und Corbin) gewählt wurde und warum es unerlässlich ist, im Detail auf dieses Verfahren einzugehen.

- In Kapitel 4 werden dann die Daten erörtert, die im Rahmen der hier dargestellten Arbeit erhoben und zur Analyse herangezogen wurden. In diesem Zusammenhang wird auch auf die angewandten Datenerhebungsmethoden, insbesondere auf die im industriellen Umfeld vorgenommenen, eingegangen.

- Nachfolgend wird in Kapitel 5 der erste, fehlgeschlagene Analyseversuch skizziert, um direkt auf die Hilfsmittel einzugehen, die entwickelt wurden, um den aufgetretenen Problemen im weiteren Verlauf der Analysen begegnen zu können.

- Den Kern der Arbeit bilden die Kapitel 6 bis 9. Hier wird im Detail auf das zentrale Ergebnis der durchgeführten Analysen, die Basisschicht, eingegangen.

[4] Die Basisschicht wurde nicht als Kodierschema entworfen. Da sie aber unter anderem aus einer Menge von Konzepten besteht, ist es naheliegend sie mit Kodierschemata zu vergleichen.

- In Kapitel 10 wird dann noch einmal auf einige Aspekte der iterativen Herleitung der BS eingegangen.
- Nachfolgend wird die Güte der Ergebnisse diskutiert um die BS dann mit Kodierschemata aus verwandten Bereichen zu vergleichen (Kapitel 11).
- Im Anschluss wird in Kapitel 12 beispielhaft erörtert, wie die Basisschicht im Rahmen der Herleitung weiterer Schichten zur Anwendung gebracht werden kann.
- Die Arbeit schließt mit einer Zusammenfassung und einem Ausblick (Kapitel 13).

Im Rahmen der nachfolgenden Darstellungen werden Vorwärts- wie auch Rückwärtsverweise verwendet. Es ist nicht zwingend notwendig, diesen zu folgen. Das Dokument ist vielmehr derart angelegt, dass es sequenziell gelesen werden kann. Sollte das Dokument als Handbuch genutzt werden, kann über die Vorwärts- und Rückwärtsverweise schnell auf verwandte Aspekte zugegriffen werden.

Kapitel 2

Verwandte Arbeiten

Unanfechtbare Wahrheiten gibt es überhaupt nicht,
und wenn es welche gibt, so sind sie langweilig.
Theodor Fontane, Der Stechlin

Im vorliegenden Kapitel wird auf bisher durchgeführte empirische Untersuchungen der PP eingegangen. Der Großteil dieser Arbeiten beschäftigt sich mit den von Verfechtern der PP propagierten Vorteilen und/oder den von Kritikern befürchteten unerwünschten Effekten. Zu den propagierten Vorteilen gehören im Wesentlichen die folgenden (siehe z.B. [218]):

- **Erhöhte Geschwindigkeit:** Programmkode kann schneller entwickelt und zum Einsatz gebracht werden.
- **Verbesserte Qualität:** Der entwickelte Programmkode enthält weniger Defekte.
- **Verbessertes Design:** Der entwickelte Programmkode ist besser entworfen und leichter lesbar.
- **Konzentrierteres Arbeiten:** Paarprogrammierer arbeiten konzentrierter als Soloprogrammierer.
- **Erhöhte „Truck Number“ [218, S. 41]:** Jeder Teil des Kodes ist zumindest zwei Programmierern bekannt und kann somit zumindest von zwei Personen ohne längere Einarbeitungszeit korrigiert oder weiterentwickelt werden.
- **Verbesserter Wissenstransfer:** Die Partner lernen voneinander in unterschiedlichen Bereichen (Arbeitsstil, Design, Kodierpraktiken, Details von Programmiersprachen und Bibliotheken, Anforderungen, Anwendungsdomänen etc.).
- **Größeres Vertrauen:** Paarprogrammierer haben ein höheres Vertrauen in ihre Arbeitsergebnisse.
- **Höhere Zufriedenheit:** Paarprogrammierer haben mehr Spaß an ihrer Arbeit.

Bei den von Kritikern befürchteten Effekten handelt es sich vor allem um die folgenden beiden:

- **Höhere Kosten:** Wenn ein Paar nicht doppelt so schnell wie ein einzelner Entwickler arbeitet, ist PP teurer als die klassische Form der Programmierung mit nur einem Entwickler pro Aufgabe (*Soloprogrammierung* (SP)).

- **Starke Aversionen:** Einige Entwickler lehnen eine derartige Form der Zusammenarbeit prinzipiell oder in Bezug auf bestimmte Partner ab. PP mindert deren Produktivität und Motivation.

Das Ziel der nachfolgenden Darstellungen besteht in erster Linie darin, den Bedarf einer konsolidierten, qualitativen Erforschung der PP aufzuzeigen. Hierzu wird zuerst auf die bisherigen quantitativen Arbeiten (Abschnitt 2.1) und dann auf die bisherigen qualitativen Arbeiten (Abschnitt 2.2) eingegangen. Empirische Untersuchungen sollen als quantitativ gelten, wenn sie die PP weitgehend als „Black Box“ betrachten und nur Resultate „messen“ bzw. Quantifizierungen zum Ziel haben, und als qualitativ, wenn dem nicht so ist, also tiefer in die „Black Box“ hineingesehen wird. Diese Form der Unterscheidung ist nicht sehr scharf, soll aber vorerst ausreichen. Detaillierter wird auf die Differenzierung zwischen quantitativen und qualitativen Studien in Kapitel 3 eingegangen. Abschließend werden die Konsequenzen erläutert, die im Rahmen der vorliegenden Arbeit aus dem Stand der Forschung gezogen wurden (Abschnitt 2.3). Außerdem wird noch einmal auf die Begriffe Driver und Observer eingegangen (Abschnitt 2.4).

2.1 Quantitative Untersuchungen der Paarprogrammierung/„Black-Box“-Untersuchungen

Der überwiegende Teil wissenschaftlicher Untersuchungen der PP ist quantitativer Natur und betrachtet in (kontrollierten) Experimenten[1] oder Quasi-Experimenten die *Qualität der Ergebnisse* [5; 8; 39; 54; 93; 124; 125; 142; 143; 154; 162; 163; 219; 229], die *Entwicklungsgeschwindigkeit* [5; 8; 39; 40; 93; 142; 143; 150; 154; 176; 229] und/oder den *Entwicklungsaufwand* [5; 39; 40; 142; 143;

[1] Einige der Studien bezeichnen ihre Untersuchungen nur allgemein als Experiment (*experiment*) und nicht als kontrolliertes Experiment oder Quasi-Experiment (siehe z.B. [40; 124; 142; 150; 162; 163; 212]). Eine Veröffentlichung (Xu und Rajlich [229]) charakterisiert ihre Untersuchung als Fallstudie.

150; 154; 162; 163; 212; 229] – in der Regel im Vergleich zur SP.[2] Von Hannay et al. [88] wurden die Resultate dieser Studien 2009 in einer Meta-Analyse zusammengefasst.[3] Diese kommt zu folgenden Ergebnissen:

- **Qualität:** Die Resultate von 14 sich mit der Qualität der Arbeitsergebnisse auseinandersetzenden Studien (insgesamt 38 Untersuchungen mit 1160 Versuchspersonen; siehe Liste oben) zeigen zusammengenommen, dass die PP – im Vergleich zur SP – einen „kleinen“ (aber signifikanten) positiven Effekt hat,[4] liefern darüber hinaus aber ein heterogenes Bild (signifikante Heterogenität auf mittlerem Niveau (für Details siehe [88])). Beispielsweise messen Williams et al. [219] die Anzahl korrekt durchlaufener Testfälle und zeigen, dass Paare signifikant besser arbeiten (jedes Paar/jeder Soloprogrammierer hatte vier Programme zu entwickeln). Z.B. passierten die von Paaren als letztes entwickelten Programme im Durchschnitt 94,4% der Testfälle korrekt, die von Soloprogrammierern entwickelten hingegen nur 78,1%. Arisholm et al. [5] kommen hingegen auf eine Verbesserung der Qualität durch PP von 7% – allerdings unter Verwendung einer anderen Bewertungsmethode (unabhängige binäre Bewertungen (*correct*/*incorrect*) durch zwei „Senior Consultants“). Domino et al. [54] ermitteln sogar einen negativen Effekt der PP auf die Qualität – stehen mit diesem Ergebnis aber allein da.

- **Entwicklungsgeschwindigkeit:** Zusammengenommen zeigen 11 sich mit der Entwicklungsgeschwindigkeit beschäftigende Studien (insgesamt 21 Untersuchungen mit 669 Versuchspersonen; siehe Liste oben) einen „mittleren“ (ebenfalls signifikanten) positiven Effekt auf die Entwicklungsgeschwindig-

[2] Bei den Probanden in den hier betrachteten Studien handelt es sich in der Mehrzahl um Studierende [8; 40; 93; 124; 125; 142; 143; 150; 219; 229]. Nur vier Studien betrachteten professionelle Softwareentwickler [5; 39; 154; 176]. Bei drei Untersuchung wurden sowohl Studierende wie auch Professionals einbezogen [54; 162; 163]. Bei einigen der Darstellungen – insbesondere bei solchen von Untersuchungen mit Studierenden – wird explizit darauf hingewiesen, dass die Probanden im Vorfeld nur wenig oder gar keine Erfahrung mit PP gesammelt hatten und/oder in den untersuchten Konstellationen bisher noch niemals zusammen (an einem Rechner) agierten (siehe z.B. [5; 143; 229]). In vielen Arbeiten lassen sich hierzu aber direkt keine Informationen finden.

[3] Von Hannay et al. [88] wurden 18 Studien berücksichtigt. In diesen Arbeiten wird PP mit SP in Hinsicht auf Entwicklungsgeschwindigkeit, Entwicklungsaufwand und/oder Qualität verglichen. Herangezogen wurden Veröffentlichungen bis in das Jahr 2007. Qualitätskriterien wie z.B. Angemessenheit des Randomisierungsverfahrens oder Angemessenheit der Subjektauswahl wurden nicht berücksichtigt [88, S. 1111]. Details über weitere Auswahlkriterien sind auf S. 1111 der Arbeit zu finden.

[4] Bei den Analysen von Hannay et al. [88] wurden zwei Modelle zur Berechnung der Effektgröße herangezogen: Das „Fixed-Effects-Modell“ und das „Random-Effects-Modell“. Für die abhängige Variable Qualität ergab sich eine Effektgröße von 0,23 („Fixed“) bzw. 0,33 („Random“). Hannay et al. [88, S. 1114] bemerken mit Verweis auf Kampenes et al. [101], dass die Effektgröße im Vergleich zu anderen Experimenten im Software Engineering klein ist. Für Details siehe [88, S. 1114].

keit.[5] Allerdings boten auch diese Studienergebnisse ein eher heterogenes Bild (signifikante Heterogenität auf mittlerem Niveau (für Details siehe [88])). Beispielsweise benötigten die Soloprogrammierer bei Nosek [154] im Durchschnitt (nicht signifikante) 41% länger als die Paarprogrammierer, während bei Arisholm et al. [5] die Paare „nur“ 8% schneller waren.[6] Zwei der Studien zeigten sogar einen negativen Effekt (Verlangsamung) der PP auf die Entwicklungsgeschwindigkeit [8; 39][7].

– **Entwicklungsaufwand:** Auch hier wurden von Hannay et al. 11 Studien betrachtet (siehe Liste oben). In diesen wurden 18 Untersuchungen mit 586 Subjekten durchgeführt. Die Ergebnisse legen zusammengenommen nahe, dass die PP – im Vergleich zur SP – einen „mittleren“ (wiederum signifikanten) negativen Effekt auf den Entwicklungsaufwand hat,[8] liefern darüber hinaus aber auch ein heterogenes Bild (signifikante Heterogenität auf hohem Niveau (für Details siehe [88])). So kommen Rostaher und Hericko [176] zu dem Ergebnis, dass sich der Aufwand durch PP nahezu verdoppelt, während Williams et al. [219] – nachdem die Paare eine Einschwingphase (*pair-jelling*) hinter sich gebracht hatten[9] – eine Aufwandssteigerung von 15% ermitteln. In den beiden Studien von Phongpaibul und Boehm [162; 163] wird sogar ein positiver Effekt auf den Entwicklungsaufwand festgestellt.[10]

Es ist wichtig festzuhalten, dass nicht nur die Aufgaben und Teilnehmer in den Studien kaum miteinander vergleichbar sind, sondern dass die betrachteten abhängigen Variablen oft auch in gänzlich unterschiedlicher Weise gemessen wurden. So wurde die Qualität der Lösungen auf insgesamt elf[11] unterschiedliche Weisen erfasst [49], z.B. durch die Anzahl passierter Tests oder Qualitätsmetriken für objektorientierte Entwicklung. In Hinsicht auf die Entwicklungsgeschwindigkeit

[5] Für die Entwicklungsgeschwindigkeit ergab sich eine Effektgröße von 0,40 („Fixed“) bzw. 0,54 („Random“). Für Details siehe [88, S. 1114].

[6] Beide Studien wurden mit professionellen Softwareentwicklern durchgeführt. In der Studie von Arisholm et al. [5] wurden die Probanden derart gepaart, dass Entwickler mit ähnlicher Programmiererfahrung zusammenarbeiten mussten. Dieser Designentscheidung lagen Ergebnisse aus Studien zu Grunde, die zeigen, dass Entwickler mit ähnlichem Kompetenzstand erfolgreicher zusammenarbeiten als solche mit unterschiedlichem ([41; 68; 218]).

[7] In der Arbeit von Baheti et al. [8] wird die Entwicklungsgeschwindigkeit in Kodezeilen pro Stunde angegeben. Die Untersuchung von Canfora et al. [39] bezieht sich auf Softwaredesign im Paar (*pair designing*).

[8] Für den Entwicklungsaufwand ergab sich eine Effektgröße von -0,73 („Fixed“) bzw. -0,52 („Random“). Für Details siehe [88, S. 1114].

[9] Mit dem Begriff *pair-jelling* adressieren Williams et al. [219] die initiale Phase der Umstellung von SP auf PP. Sie erklären hiermit das Phänomen, dass die Studierenden in ihrem Experiment bei der ersten Aufgabe noch einen um 60% größeren Aufwand verursachten, bei den weiteren Aufgaben aber nur noch einen um 15% größeren.

[10] Wie schon Hannay et al. [88, S. 1115] ausführen, sind die Studien von Phongpaibul und Boehm [162; 163] nur bedingt mit den anderen vergleichbar, da in ihnen Teams, die PP betreiben, mit solchen, die Inspektionen nach Fagan [61] durchführen, verglichen werden.

[11] In vier in der Meta-Analyse berücksichtigten Studien wird die Qualität gar nicht gemessen.

wurde zwar im überwiegenden Teil der Studien die Bearbeitungsdauer untersucht – allerdings mittels zweier unterschiedlicher Kriterien: In einem Teil der Untersuchungen wurde die Zeit bis zur Fertigstellung gemessen (Entscheidung der Entwickler), in einem anderen Teil die Zeit, bis die Lösung einen gewissen Qualitätsstandard erfüllt hatte [88, S. 1114]. Auch der Aufwand wurde mit zwei unterschiedlichen Kriterien erfasst: Durch Verdopplung der Bearbeitungszeit der Paare oder, falls Teams miteinander verglichen wurden (z.B. Teams von Soloprogrammierern mit Teams von Paarprogrammierern), durch den Aufwand, der jeweils bei den Teams entstand [88, S. 1114].

Hannay et al. [88, S. 1120] halten nicht nur fest, dass die Varianz zwischen den Ergebnissen der einzelnen Studien hoch ist sondern auch, dass PP – zumindest in Hinsicht auf die von ihnen betrachteten Auswirkungen – nicht gleichförmig nutzbringend zu sein scheint. Das Ziel weiterführender Studien sollte deshalb darauf gerichtet werden, die auf die Effekte der PP einwirkenden Faktoren zu „entwirren".

Neben Qualität, Entwicklungsgeschwindigkeit und Entwicklungsaufwand wurden auch die folgenden Aspekte der PP quantitativ untersucht:

- **Arbeitszufriedenheit und Vertrauen in Ergebnisse:** Eine große Anzahl von Untersuchungen beschäftigte sich mit der Arbeitszufriedenheit und/oder dem Vertrauen der Programmierer in die von ihnen erzielten Ergebnisse. So wurden in vielen Studien nicht nur Zeiten gemessen und Ergebnisse beurteilt, sondern auch Entwickler in Hinsicht auf ihren Spaß an der (gerade geleisteten) Entwicklungsarbeit [8; 54; 86; 134; 154; 183; 207; 211; 212; 219] und/oder ihre Zuversicht in Hinsicht auf die Korrektheit ihrer Lösungen [86; 134; 154; 183; 207; 211; 219] befragt. Hierbei wurden in der Regel Aussagen von Paarprogrammierern mit solchen von Soloprogrammierern verglichen. Die Ergebnisse liefern ein einheitliches Bild: Paarprogrammierer haben im Vergleich zu Soloprogrammierern mehr Spaß an ihrer Arbeit und ein größeres Vertrauen in die von ihnen erstellten Lösungen. Nur das Ergebnis einer Studie von Vanhanen und Lassenius [212] lieferte eine weniger eindeutige Aussage. Hier hatten zwar 7 von 8 Paarprogrammierern und nur 5 von 8 Soloprogrammierern Spaß am verwendeten Arbeitsstil, trotzdem befanden nur 2 von 8 Paarprogrammierern PP als geeigneter für Projekte, die dem im Experiment zu bewerkstelligenden ähneln.

- **Einfluss von Fertigkeiten und Erfahrung:** In einigen Studien wurden Anstrengungen unternommen zu analysieren, inwieweit die Kompatibilität von PP praktizierenden Entwicklern mit ihren Fertigkeiten zusammenhängt. So untersuchten Katira et al. [103] die Kompatibilität von mehr als 564 Studierenden aus drei verschiedenen Kursen – einer Veranstaltung für Erstsemester (*freshman*), einer für Studierende vor dem ersten akademischen Abschluss (*undergraduate*) und einer für Studierende nach dem ersten akademischen Abschluss (*graduate*). In jedem Kurs wurden die Studierenden aufgefordert, mehrere Aufgaben in Paaren zu bearbeiten. Nur die

„Graduates“ hatten die Wahl zwischen PP und SP. Die Paarbildung wurde in allen Kursen bzw. bei allen Aufgaben von den Veranstaltern (in der Regel wohl randomisiert) vorgenommen.[12] Nach jeder Aufgabe bzw. Paarung bekamen alle Teilnehmer einen Online-Fragebogen. In diesem sollten sie Angaben zur Leistung des Partners sowie zur Kompatibilität des Paares machen. Die Analyse fokussierte sich auf die Fragen zur Kompatibilität. Hierbei wurden drei Hypothesen untersucht[13]: Paare sind kompatibler, wenn Studierende

- mit einem ähnlichen „Skill-Level“ zusammenarbeiten. Der „Skill-Level“ der Probanden wurde anhand ihrer Note zur Semestermitte festgelegt.
- mit einer ähnlich wahrgenommenen technischen Kompetenz zusammenarbeiten. Hierzu wurden die Studierenden nach jeder Sitzung gebeten, den Partner mit sich selbst zu vergleichen. Es konnte eine von drei Antwortmöglichkeiten ausgewählt werden: besser, ungefähr genauso gut, schwächer.
- mit einer ähnlichen Selbsteinschätzung[14] zusammenarbeiten. Diese Hypothese wurde nur bei den Erstsemestern untersucht. Hierzu wurden diese nach der ersten Aufgabe gebeten, sich selbst zu bewerten.

Insgesamt wurden in weniger als 10% der 1563 gesammelten Datenpunkte Paarungen als nicht kompatibel bewertet. Bei den „Graduates“ konnte ein positiver Zusammenhang zwischen „Skill-Level“ und Kompatibilität nachgewiesen werden – bei den Erstsemestern und „Undergraduates“ hingegen nicht. Für alle Gruppen konnte eine signifikante positive Beziehung zwischen Kompatibilität und wahrgenommener Kompetenz ermittelt werden. Bei den Erstsemestern zeigte sich darüber hinaus ein schwacher negativer Zusammenhang zwischen Selbsteinschätzung und Kompatibilität.[15] Interessanterweise wurde also der weitaus überwiegende Teil der (zufälligen)

[12] Die Erstsemester waren aufgefordert, bei jeder der vier Aufgaben/Projekte mit einem neuen Partner zusammenzuarbeiten. Die „Undergraduates“ mussten vier Aufgaben mit unterschiedlichen Partnern absolvieren. Bei der fünften und letzten im Paar zu absolvierenden Aufgabe sollte die Konstellation genauso wie in Aufgabe vier sein. Die „Graduates“ mussten drei Aufgaben absolvieren. Wenn sie sich dazu entschlossen hatten, im Paar zu arbeiten, konnten sie einen präferierten Partner angeben. Die Zuweisung wurde letztendlich aber vom Veranstalter vorgenommen.

[13] Anhand einer vierten Hypothese wurde die Kompatibilität in Hinsicht auf Persönlichkeitstypen untersucht.

[14] Die Selbsteinschätzung konnte in neun Stufen angegeben werden – von „I don't like programming, and I think I am not good at it. I can write simple programs, but have trouble writing new programs for solving new problems“ („Code-a-Phobes“) bis „I have no problems at all completing programming tasks to date, in fact they weren't challenging enough. I love to program and anticipate no difficulty with this course“ („Code-Warriors“).

[15] Alle Zusammenhänge wurden mittels des Rangkorrelationskoeffizienten von Spearman bestimmt.

Paarungen als kompatibel empfunden, wobei Partner bevorzugt wurden, die eine ähnliche technische Kompetenz zu besitzen schienen. Thomas et al. [207] liefern in ihrer Studie (Fragebogen und Experiment) mit mehr als 60 Studierenden Belege dafür, dass so genannte „Code-Warriors“[16] PP weniger mögen, wenn sie mit so genannten „Code-a-Phobes“[17] zusammenarbeiten müssen und dass Studierende die besten Arbeitsergebnisse erzielen, wenn sie mit solchen gepaart werden, die ein ähnliches Selbstvertrauen haben. Das erste Ergebnis passt zu Resultaten einer qualitativen Studie von Cao und Xu (s. S. 37 f).

– **Einfluss von Persönlichkeitseigenschaften:** Häufiger als der Einfluss von Fertigkeiten wurde der von Persönlichkeitseigenschaften untersucht – beispielsweise in [43; 45; 52; 87; 93; 102; 103; 114; 191; 192; 220]. Zur Klassifizierung der Persönlichkeit wurde hierbei überwiegend der Myers-Briggs-Typindikator (MBTI, [146; 147]) verwendet. Es kamen aber auch andere Modelle wie die „International Personality Item Pool Representation of the NEO-PI-R“ (IPIP-NEO [81; 82]) zum Einsatz. Katira et al. [103] (für Details siehe S. 31) stellten z.B. die Hypothese auf, dass Paare mit unterschiedlichem Persönlichkeitstyp kompatibler sind. In ihrer Studie konnten sie zwar zeigen, dass sich in der Gruppe der Erstsemester die Studierenden dann als kompatibler bewerteten, wenn sie einen unterschiedlichen MBTI besaßen (Unterschied in mindestens einer Dimension), in der Gruppe der „Undergraduates“ war es ihnen aber nicht möglich, einen solchen Zusammenhang zu identifizieren.[18] Auch Salleh et al. [183] untersuchten in einem Experiment (*single factor between-group design*) an 54 Studierenden den Einfluss der Persönlichkeit auf die PP – allerdings unter Verwendung der IPIP-NEO. Als Experimentgruppe dienten Paare mit unterschiedlichen Persönlichkeiten, als Kontrollgruppe solche mit ähnlichen Persönlichkeiten, wobei die Dimension „Gewissenhaftigkeit“ als Unterscheidungskriterium fungierte. Das Ziel der Untersuchung bestand darin, den Einsatz der PP als pädagogisches Werkzeug zu optimieren. Deshalb wurde als abhängige Variable die Studienleistung (*academic performance*) betrachtet. Die zentrale Hypothese lautete: Unterschiede in den Persönlichkeitseigenschaften beeinflussen den Erfolg von PP einsetzenden Studierenden. Die entsprechende Nullhypothese konnte allerdings im Rahmen des Experimentes nicht zurückgewiesen werden. Der Artikel gibt darüber hinaus einen Überblick über 10 weitere sich mit Persönlichkeitstypen und PP auseinandersetzenden Studien. Es wird festgehalten, dass diese Untersuchungen zwar unterschiedlichste Ergebnisse liefern, dass die meisten aber zu dem Schluss kommen, dass Persönlichkeitseigenschaften keinen erheblichen Einfluss auf die Effektivität der PP

[16] Siehe Fußnote 14.
[17] Siehe Fußnote 14.
[18] Es wurde der Rangkorrelationskoeffizient von Spearman verwendet. Für die Gruppe der „Graduates“ lagen keine MBTI-Daten vor.

besitzen [183, S. 251].[19] Hannay et al. [87] fassen dies für ihre Studie aus dem Jahr 2010 wie folgt zusammen:

> „Personality traits in general have modest predictive value on pair programming performance compared with expertise, task complexity, and country. We conclude that more effort should be spent on investigating other performance-related predictors such as expertise, and task complexity, as well as other promising predictors, such as programming skill and learning."

– **Arbeitsverteilung:** Plonka et al. [166] stellen in ihrer Studie die Frage, ob es sich bei der PP um eine Verschwendung von Ressourcen handelt. Um sich einer Antwort anzunähern, untersuchen sie die Verteilung der Redebeiträge und Driveraktivitäten zwischen den Partnern. Als Datenmaterial dienen ihnen 21 Videoaufzeichnungen von Paarprogrammierungssitzungen mit professionellen Entwicklern. Sie kommen zu folgendem Schluss:

> „As a result, we found that about two thirds of the PP sessions are not equitable in terms of driving and verbal contributions. Furthermore, we found that 10 sessions out of the 21 sessions are not equitable with respect to both types of contributions and that in 6 out of this 10 PP sessions, one developer is dominating both types of contributions. [...] [Our] results support the findings of Höfer [97], who reported in the context of a study with undergraduate students, that most pairs do not share the keyboard and mouse equally. Based on another study conducted with students, Cao and Xu [41][20] pointed out that there is usually a leader in PP sessions depending on factors such as competence level, experience and personalities. We found that in 6 PP sessions, one developer dominated the verbal as well as the driving contributions. However, this does not necessarily imply that this developer led the whole session as our quantitative results do not provide enough information about the interactions among the developers."

In Hinsicht auf die Verschwendung von Ressourcen halten sie fest:

> „Based on our results, we conclude that the quantitative concept of equity of participation (in terms of driving and verbal contributions) is not a sufficient metric to evaluate whether PP is a waste of resources."

[19] An dieser Stelle wird nicht zwischen unterschiedlichen Effektivitätsmaßen unterschieden.
[20] Auf die Studie von Cao und Xu wird auf S. 37 eingegangen.

Interviews, die Plonka et al. mit den Paaren nach den Sitzungen durchführten, lassen sie darüber hinaus zu folgender Aussage in Hinsicht auf Faktoren, die die Verteilung beeinflussen, kommen[21]:

> „We identified four factors: workstation, work style, personal preferences and PP experience, and skill differences. Modifying those factors could change the participation balance in a pair. “

– **Einsatz der PP in der Lehre:** Eine Reihe von – auch qualitative Methoden verwendenden – Studien beschäftigt sich mit dem Nutzen, den die PP in der Lehre, in der Regel in der Hochschulausbildung von Informatikern, erbringen kann (siehe z.B. [2; 18; 133; 148; 149; 205; 211; 221]) – in den meisten Fällen, indem solo arbeitende Studierende mit solchen, die ihre Aufgaben in Paaren erledigen, in Hinsicht auf ihre Studienleistungen verglichen werden. Nagappan et al. [148] fassen ihre diesbezüglichen Ergebnisse wie folgt zusammen:

> „Student pair programmers were more self-sufficient, generally perform better on projects and exams, and were more likely to complete the class with a grade of C or better than their solo counterparts. Results indicate that pair programming creates a laboratory environment conducive to more advanced, active learning than traditional labs; students and lab instructors report labs to be more productive and less frustrating.“

Auch McDowell et al. [133] kommen zu einem Ergebnis, welches für den Einsatz der PP in der Lehre spricht:

> „Students who programmed in pairs produced better programs, completed the course at higher rates, and performed about as well on the final exam as students who programmed independently. Our findings suggest that collaboration is an effective pedagogical tool for teaching introductory programming.“

Manche Studien liefern aber auch weniger positive Resultate. So berichtet Tessem [205, S. 133 f] aus einem Kurs mit 6 XP praktizierenden Entwicklern folgendes:

> „All programmers reported that they found pair programming a positive experience, enhancing learning and also leading to higher quality. However, this is somewhat contradicted by three of the programmers who also use negative phrases like “extremely inefficient”, “very exhausting”, “waste of time”, and “tiresome”. [...]

[21] Details zur Art der Interviews und der Methode der Analyse der Interviews finden sich in dem Artikel nicht.

> Other problems mentioned are communication problems within pairs and pairs that lost focus on the tasks they had committed to. Several of the programmers mention that the only way to solve the communication problems is to show more courage in criticizing your partner's work and more acceptance of criticisms."

Zusammenfassend muss aber festgehalten werden, dass der überwiegende Teil der Studien zu einer eher positiven Bewertung des Einsatzes der PP in der Lehre kommt (siehe z.B. [18; 133; 148; 149; 211; 221]) – sowohl in Hinsicht auf die resultierenden Studienleistungen wie auch in Hinsicht auf andere Effekte wie Minimierung von Frustration oder Reduzierung der Arbeitsbelastung [211]. Allerdings wurde kaum beleuchtet, unter welchen Bedingungen ein Einsatz der PP besonders zu empfehlen oder von einem solchen vielleicht sogar besser abzuraten ist. Eine der Ausnahmen bildet die Studie von Chaparro et al. [2], durchgeführt in einem zehnwöchigen Kurs mit 80 Studierenden, von denen 58 an der Untersuchung (teilnehmende Beobachtung, Fragebögen, leitfadengesteuerte Interviews, Feldnotizen) teilnahmen. Sie kommt zu dem Schluss:

> „This study suggests that students' skill level and the programming task play a major role in the perceived effectiveness of pair programming as an educational technique. Students seem to enjoy and benefit from pair programming if the skill level gap is not too big. Also, debugging seems to be a difficult task for pair collaboration."

Diese Bewertung wird in Teilen auch durch eine Untersuchung von VanDeGrift [211] gestützt. Hier hatten von 293 Studierenden knapp 23% einen Unterschied im „Skill-Level" als Problem bei der PP wahrgenommen (pro Person waren drei Projekte à 2 Wochen zu absolvieren). Häufiger wurden nur Probleme mit Persönlichkeitsunterschieden (knapp 26%) und der Terminplanung (knapp 48%) genannt. Bevan et al. [18] kommen in ihrer Studie (Vergleich von Anfängerkursen, in denen PP praktiziert wird, mit einem, in dem dies nicht getan wird) zu dem Schluss:

> „We emphasize, however, that the implementation of pairing within the freshman classroom requires careful attention to detail; behaviors that are not common in industry or even upper-division classes can undermine the stability of a given pair. In this setting, the most critical aspect of creating an effective pair programming implementation is to minimize the potential scheduling conflicts between partners. The additional support of a culture that emphasizes cooperation, mutual respect, and shared responsibility paves the way for partners to work out attitude-based problems."

2.2 Qualitative Untersuchungen der Paarprogrammierung

Wie bereits angedeutet wurden bisher weitaus weniger qualitative als quantitative Studien der PP durchgeführt. Die wichtigsten werden im Folgenden grob nach Themengebieten geordnet vorgestellt.

– **Fertigkeiten und Erfahrung:** Cao und Xu berichten 2005 [41] von ihrer explorativen Studie zu Aktivitätsmustern bei der PP (*activity patterns of pair programming*). In dieser Untersuchung fanden die Fertigkeiten der Probanden besondere Berücksichtigung. Die Autoren beobachteten Studierende, die an einem neunwöchigen Softwareprojekt teilnahmen, in dem agile Methoden (kurze Iterationen, PP etc.) zur Anwendung gebracht werden sollten. Den Studierenden war es erlaubt, sich eigenständig zu Paaren zusammenfinden. Diejenigen, die dies nicht taten, wurden durch die Versuchsleitung zu Paaren gruppiert. Zu Beginn der Untersuchung gab es 10 Paare[22], von denen am Ende des Projektes nur 6 übrig geblieben waren. Ein Entwickler arbeitete von Anfang an allein. Cao und Xu teilten die Studierenden auf Basis ihrer bis zur Mitte der Veranstaltung erbrachten Leistungen in drei Fertigkeitsklassen: Hoher Grad an Fertigkeiten (im Weiteren kurz *Hoch*), mittlerer Grad an Fertigkeiten (im Weiteren kurz *Mittel*) und niedriger Grad an Fertigkeiten (im Weiteren kurz *Niedrig*). Bei den am Ende verbleibenden 6 Paaren handelte es sich um zwei Hoch-Hoch-Kombinationen, zwei Mittel-Mittel-Kombinationen und zwei Hoch-Niedrig-Kombinationen. Drei Hoch-Niedrig-Kombinationen und eine Mittel-Niedrig-Kombination hatten sich aufgelöst. Niedrig-Niedrig-Kombinationen oder Hoch-Mittel-Kombinationen waren nicht angetreten. Die Paare wurden bei ihrer Arbeit auf Video aufgezeichnet. Dies geschah in der zweiten Iteration. Darüber hinaus wurden Fragebögen ausgegeben und Arbeitstagebücher zur Aufwandserfassung gefordert. Die resultierenden Daten wurden mittels Protokollanalyse [59] ausgewertet. Gestartet wurde mit einer Kombination der Kodierschemata von Lim et al. [121] und Okada et al. [156], um dann während der Analyse ein eigenes Schema zu entwickeln (5 Kategorien (*Leader's activities*, *Ask for opinions*, *Explanatory activities*, *Critiques*, *Summary of results*) mit 12 Subaktivitäten; siehe auch Abschnitt 11.3). Die Ergebnisse werden von Cao und Xu [41] wie folgt zusammengefasst:

> „We found that pair programming engaged in activities such as asking for opinions, requesting for explanations, critiquing partners' approaches and summarizing current status. These activities lead to more deeper-level thinking [...]. The high-high combination involves more deeper-thinking activities than the medium-medium

[22] Die Angaben über die Anzahl der Paare zu Beginn der Untersuchung schwanken im Artikel zwischen 10 und 11.

> level combinations, while the high-low combination has the least interaction between participants. Generally speaking, the high-high pairs enjoyed pair programming experience more than the other pairs. [...] The performance of the medium-medium pairs was not satisfying. Very limited knowledge was generated. The performance of the high-low pairs was not satisfying. The high competent participants did not enjoy the practice while the low competent participants benefited from the practice. [...] Another finding is that there is a leader in each pair. We found that who is the leader depends on the factors such as competence level, experience and personalities. The study suggests that different combinations in pair programming achieve different outcomes.“

Im Artikel wird der Analyseprozesses nicht detailliert beschrieben. Es bleibt z.B. unklar, inwieweit zu Beginn eine Vorstellung über das, was unter Aktivitätsmustern verstanden werden soll, existierte bzw. wie stark diese Vorstellung durch die verwendeten Kodierschemata geprägt wurde.

– **Driver und Observer:** Chong und Hurlbutt [46] führten 2005 eine viermonatige ethnographische Studie mit professionellen, in zwei Teams (6-9 Entwickler und 9-10 Entwickler) arbeitenden Entwicklern durch. Sie besuchten jedes der Teams einmal in der Woche und beobachteten Paarprogrammierungssitzungen, indem sie sich hinter Paare setzten und Notizen zu deren Interaktionen und Aktivitäten machten. Falls möglich nahmen sie die Dialoge der Paare auf und transkribierten sie später.[23] Im Rahmen der nicht näher erläuterten Analysevorgänge („identify consistent and repeated patterns of behavior“) entwickelten sie ein im Artikel nicht beschriebenes Kodierschema. Ihre zentrale Erkenntnis bezieht sich auf die Rollen Driver und Observer:

> „Aside from the task of typing, we found no consistent division of labor between the “driver” and the “navigator”. Instead, the two programmers moved from task to task together, considering and discussing issues at the same strategic “range” or level of abstraction.“

Darüber hinaus halten sie fest:

> „When gaps in expertise were sufficiently large, the programmer with more expertise dominated the pair programming interaction. We use the term expertise here to refer to a combination of programmer skill and knowledge.“

[23] Im Artikel finden sich keine Angaben über die Anzahl der beobachteten/aufgezeichneten Sitzungen.

Zu den Ergebnissen von Chong und Hurlbutt passen diejenigen einer qualitativ-quantitativen Untersuchung von Freudenberg (geborene Bryant) et al. [34; 66]. Hier wurden in vier, in unterschiedlichen Unternehmen stattfindenden einwöchigen Studien professionelle Entwicklerpaare bei ihrer täglichen Arbeit beobachtet. Insgesamt wurden 18 unterschiedliche Paarungen einbezogen und 24 Stunden Material transkribiert und analysiert.[24] Der Fokus der Untersuchung lag auf den Paarprogrammierungsrollen und deren Hauptmerkmalen. Es kam die *Verbal Protocol Analysis* nach Chi [44] zum Einsatz. Da Bryant et al. besonders am Detailgrad/Abstraktionslevel interessiert waren, auf dem die Entwickler diskutierten, orientierte sich das verwendete Kodierschema an dem vom Pennington [160]. Es wurde ergänzt durch ein Konzept des Schemas von Good und Brna [83] sowie durch zwei eigene. Das eigene Konzept „VAGUE“ diente dazu, eine vollständige Kodierung zu erhalten, also auch Äußerungen einordnen zu können, deren Abstraktionslevel nicht zu ermitteln war.[25] Insgesamt kamen somit sechs Konzepte zur Kodierung von Äußerungen zum Einsatz. Jeder Äußerung wurde ein Konzept exklusiv zugeordnet. Neben dieser Kodierung wurde für jede Äußerung festgehalten, welche Rolle (Driver, Observer) der Sprecher inne hatte. Ihr Ergebnis fassen Bryant et al. [34] wie folgt zusammen:

> „It would seem from our findings that contrary to what has been previously reported (e.g. [100; 217]) the role of the navigator is not defined by their correcting syntax and grammar significantly more than the driver. In fact, utterances at this level were scarce in the pair programming sessions observed and on the infrequent occasions in which they do occur are almost evenly distributed between driver and navigator roles [...]. Similarly in contradiction to what has previously been suggested (e.g. [52; 92]) the pair programmers in the sessions observed did not show a general difference in the level of abstraction of their discussions according to role. In fact, rather than working at a higher level of abstraction, the pattern of abstraction levels of navigator's utterances are very similar to those of the driver. Both partners tended to speak more at the [level of 'abstract chunks of code'], which further contradicts existing suggestions as it shows the driver working consistently at one of the higher levels of abstraction along with the navigator.“

– **Kollaboration:** Bevor Bryant et al. ihre Untersuchungen zu Rollen und Abstraktionsniveaus veröffentlichten (siehe oben), publizierten sie eine Stu-

[24] Auch wenn es nicht explizit erwähnt wird, muss davon ausgegangen werden, dass Audioaufzeichnungen der Sitzungen vorgenommen wurden. Nur in einem Unternehmen konnten auch Videoaufzeichnungen durchgeführt werden. Die Aufzeichnung von Bildschirminhalten war in keiner der Firmen möglich.

[25] Im Durchschnitt wurden in jeder Sitzung 57% der Äußerungen als „VAGUE“ klassifiziert.

die über Kollaboration bei der PP [33]. Schon dieser Studie lagen die oben erläuterten Daten zugrunde (hier wurden allerdings 36 Sitzungen berücksichtigt). Bei der Analyse wurde ebenfalls die *Verbal Protocol Analysis* nach Chi [44] verwendet. Im Rahmen der Analyse wurde ein Kodierschema zur Klassifizierung von Teilaufgaben (*generic subtask types*) bei der PP entwickelt (für Details siehe Abschnitt 11.3). Ihr Ziel bzw. ihr Vorgehen fassen Bryant et al. wie folgt zusammen:

> „From recordings of their conversations we analyze which generic sub-tasks were discussed and use the contribution of new information as a means of discerning the extent to which each pair collaborated."

Sie kommen zu folgendem Ergebnis:

> „This report highlights pair programming as highly collaborative, with both partners contributing information to almost every subtask, irrelevant of role. This contrasts with suggestions that the benefits of pair programming may come from encouraging verbalization, facilitating overhearing or peer pressure from being watched. [...] While generally very high (over 80%), the level of collaboration varied according to task. Refactoring and writing new code showed the highest level of collaboration and therefore one might suggest that the challenging nature of these tasks made pairing on them most valuable. [...] The studies performed showed very evenly distributed contributions across role, with the driver contributing only slightly more than the navigator. This negates claims that the driver and navigator roles may be oriented toward different types of task, but further investigation is required if we are to fully understand whether a task benefits from the driver and navigator focusing on different aspects (e.g. working at different levels of abstraction)."

– **Lernen:** In ihrer auf Protokollanalyse [59] basierenden Studie vergleichen Xu et al. [228] fortgeschrittene Entwickler mit Experten in Hinsicht auf diejenigen Lernprozesse, die bei der inkrementellen Programmentwicklung (Test-First-Programmierung [14, S. 50] und Refaktorierung (*Refactoring*) [65]) stattfinden. Die von ihnen durchgeführte Untersuchung repliziert eine Paarprogrammierungsbeobachtung von Martin und Koss [127]. Während es sich bei Martin und Koss allerdings um Entwickler mit mehr als 15 Jahren Erfahrung handelt, beobachten Xu et al. zwei studentische Paare („Graduates"). Die von den Paaren zu bearbeitende Aufgabe übernahmen Xu et al. von Martin und Koss. Vom ersten Paar wurde ein Video aufgezeichnet, vom zweiten der Bildschirm sowie Ton. Es dauerte jeweils ca. 5 Stunden,

bis die Paare die Aufgabe erledigt hatten. Die Daten wurden nachfolgend transkribiert. Die zentralen Fragestellungen von Xu et al. lauteten:

- "Do different pairs follow the same cognitive process?
- If different pairs follow different cognitive processes, what are the differences?
- Are there differences between intermediate level programmers and experts?"

Als Kodierschema setzen Xu et al. die so genannte „Self Directed Learning Theory" ein, eine Kombination aus der „Constructivist Learning Theory" [164] und „Bloom's taxonomy of the cognitive domain" [20] (siehe auch Abschnitt 11.3), wobei die Dialoge „grob" in 64 bzw. 62 Episoden zerlegt wurden. In ihrem Artikel formulieren sie ihre wichtigsten Ergebnisse wie folgt:

> „During incremental software development with the test-first approach, absorption[26] is the dominant activity behind the process. Reorganization[27] often occurs, but denial and expulsion occasionally appear. Four out of six of Bloom's levels were identified in the replicated case study.
>
> In terms of differences between experts and intermediates, experts tend to discuss broadly on the problems and related domain concepts, while intermediates try to deal with individual concepts before discussing new ones. Experts are also willing to recognize and reconsider inappropriate design decisions, but intermediates seem to keep using them. [...] Experts seem to, more or less, take knowledge as granted and spend more time to synthesize the knowledge and generate hypothesis.

Abschließend soll noch darauf hingewiesen werden, dass es Xu et al. weniger um die Erforschung der PP als um die Analyse der Lernprozesse von Entwicklern ging. Die PP war hier ein geeignetes Mittel, da kognitive Prozesse bei der PP auf natürliche Weise, nämlich in der Kommunikation der Partner, sichtbar werden können.

2.3 Schlussfolgerungen aus der bisherigen Paarprogrammierungsforschung

Betrachtet man die oben dargestellten quantitativen und qualitativen Untersuchungen der PP kann festgehalten werden:

[26] *Absorption*: „Occurs when learners add new facts to their knowledge" [228, S. 340].

[27] *Reorganization*: „When the learners reorganize their knowledge to aid future absorption of new facts, we call their cognitive activity *reorganization*" [228, S. 341].

- Es ist eine Vielzahl quantitativer, den Prozess der PP weitgehend als „Black Box“ betrachtender Studien durchgeführt worden. Der Status quo auf diesem Sektor lässt sich wie folgt zusammenfassen:
  - Auch Untersuchungen bezüglich gleicher Aspekte fanden in der Regel unter allenfalls ähnlichen Randbedingungen sowie unter Verwendung unterschiedlicher Maße statt, ohne dass dies näher diskutiert wurde.
  - Zusammengenommen deuten die Ergebnisse der zu einem Aspekt gehörenden Studien zwar in der Regel in eine bestimmte Richtung, sind aber trotzdem heterogen. Dies legt die Vermutung nahe, dass die PP nicht in jeder Situation im selben Maß nutzbringend ist.
  - Erklärungsmodelle für Effekte oder Diskrepanzen zwischen Ergebnissen werden kaum geliefert. Insbesondere helfen die Studien wenig dabei zu verstehen, welche Faktoren bei der PP in welcher Weise eine (zentrale) Rolle spielen.
- Die Anzahl der bisher durchgeführten qualitativen Studien ist gering. Hier gilt:
  - Es werden weitgehend voneinander losgelöste einzelne Aspekte betrachtet.
  - In keiner der Studien wurde explizit versucht herauszufinden, durch welche Aktivitäten eine Paarprogrammierungssitzung maßgeblich „geformt“ wird. Vielmehr wurde bei einer Reihe der Analysen auf bereits existierende, aber nicht speziell auf die PP zugeschnitten Modelle bzw. Kodierschemata zurückgegriffen. Somit besteht die Möglichkeit, dass – zumindest in Teilen – wichtige Phänomene nicht oder nur unzureichend berücksichtigt wurden. Nur in den Studien von Chong und Hurlbutt [46], Cao und Xu [41] und Bryant et al. [33] ist anders vorgegangen worden. So entwickelten Chong und Hurlbutt zwar ein eigenes Schema, dieses wird aber im Artikel nicht beschrieben. Cao und Xu adressieren im Titel ihrer Studie Aktivitätsmuster bei der PP. Ihr Fokus liegt dann aber doch weniger auf einer Ausarbeitung eines Schemas von grundlegenden Aktivitäten sondern mehr auf einem Vergleich von Paaren mit unterschiedlichen Fertigkeitsniveaus. Bryant et al. [33] haben ihr Schema zwar aus den Daten gewonnen, dieses ist aber auf zu absolvierenden Aufgaben und nicht auf Aktivitäten fokussiert (siehe auch Abschnitt 11.3).
  - In keiner der bisherigen Studien wurde explizit versucht, eine allgemeine Theorie der PP zu erarbeiten oder die Grundlagen für eine solche zu legen.

Die weitgehend heterogenen Ergebnisse der quantitativen Studien auf der einen Seite und die inhaltlich voneinander losgelösten und kaum miteinander

integrierbaren Ergebnisse der wenigen qualitativen Studien auf der anderen Seite lassen den Wunsch nach einer konsolidierten Paarprogrammierungsforschung aufkommen – mit dem Ziel, ein holistisches Bild der Praktik zu generieren und die den Erfolg beeinflussenden Faktoren identifizieren zu können. Wie bereits in der Einleitung erläutert hakt die vorliegende Arbeit genau an dieser Stelle ein (siehe Kapitel 1, S. 22 ff).

2.4 Definition der Begriffe Driver und Observer

Entgegen der in Kapitel 1 wiedergegebenen Beschreibung von Williams et al. [219] wird das Einnehmen der Rollen Driver oder Observer in der vorliegenden Arbeit nicht mit einem Fokussieren auf bestimmte kognitive Funktionen wie z.B. einer Ausrichtung auf strategisches Denken gleichgesetzt. Vom Driver wird vielmehr nur als demjenigen gesprochen, der aktuell die Kontrolle über Maus und Tastatur ausübt, und vom Observer als demjenigen, dessen Partner gerade in der Rolle des Drivers aktiv ist. Außerdem wird eine Zusammenarbeit von zwei Entwicklern an einem Rechner unabhängig davon, ob und wie oft ein Rollenwechsel durchgeführt wird, als PP bezeichnet – auch wenn Williams et al. [219] betonen, dass die Rollen innerhalb einer Paarprogrammierungssitzung bewusst „periodisch“ getauscht werden sollten. Mit diesen Festlegungen wird folgenden Punkten Rechnung getragen:

- Die oben beschriebenen Charakterisierungen der Rollen durch bestimmte kognitive Funktionen sind präskriptiver oder auf Anekdoten beruhender Natur. Weder ihre Nützlichkeit noch ihre Praktikabilität wurde bisher systematisch analysiert.
- Zumindest im industriellen Umfeld scheint es kaum (standardisierte) Vorschriften zur Fokussierung auf derartige kognitive Funktionen zu geben. Vielmehr ist es so, dass Entwickler bei der PP weitgehend ohne den Prozess betreffende Vorgaben oder Hinweise agieren.[28]
- Eine Reihe von Veröffentlichungen zeigt, dass einem derartigen Bild der Driver- und Observer-Rolle zumindest in Teilen widersprochen werden muss (siehe Verweise in Wray [224] oder S. 38).

Mit den hier gemachten Festlegungen werden also nicht mehr Annahmen als unbedingt nötig und angebracht gemacht.

[28] Bei dieser Aussage handelt es sich nicht um das Ergebnis formeller Untersuchungen, sondern um eine auf Beobachtungen basierende These.

Kapitel 3

Die Methode der Grounded Theory

Protect Me From What I Want
Jenny Holzer

Wie im ersten Kapitel beschrieben, geht es in der vorliegenden Arbeit darum, den Grundstein dafür zu legen, dass nachfolgend sukzessive ein in Theorien gebettetes Verständnis über das, was bei der PP „passiert“ erlangt werden kann. Dieses Ziel ist nur unter Zuhilfenahme von qualitativen Methoden zu erreichen, also solchen, von denen Bortz und Döring [28, S. 299] berichten, dass sie als in gewisser Weise idiografisch (im Gegensatz zu nomothetisch)[1], im Feld arbeitend (im Gegensatz zu im Labor arbeitend), induktiv (im Gegensatz zu deduktiv), holistisch (im Gegensatz zu partikular), explorativ (im Gegensatz zu explanativ) und auf das Verstehen zielend (im Gegensatz zu auf das Erklären zielend) beschrieben werden.[2] Insbesondere in der qualitativen Sozialforschung und mit der von Mayring diagnostizierten „qualitativen Wende“ in den 1980er Jahren [130], wurden (in Deutschland) eine ganze Reihe von Verfahren „entwickelt bzw. wieder entdeckt“ [132, S.103], denen zugeschrieben wird, dass sie derartige Eigenschaften besitzen. Unter ihnen gibt es „strukturierte und weniger stark strukturierte Verfahren [sowie] Vorgehensweisen, die mehr auf Beobachtungsmaterial, und solche, die auf verbale Daten bezogen sind“ [132, S.103]. Mayring [132, S. 104 ff] erwähnt unter anderem die Grounded Theory (entwickelt Anfang der 1960er Jahre von Barney G. Glaser und Anselm L. Strauss [77]), die Phänomenologische Analyse (Phänomenologie; ursprünglich als philosophische Methode von Edmund Husserl (1859-1938) entwickelt) und die Qualitative Inhaltsanalyse (eine in den „ersten

[1] „Die Unterscheidung zwischen nomothetischen [...] und idiographischen Wissenschaftsdisziplinen geht auf Windelband [[222]] zurück und sollte ursprünglich die Naturwissenschaften und die Geisteswissenschaften differenzieren. Während Naturwissenschaftler generalisierende Naturgesetze aufstellen (nomothetisch vorgehen), sei es Ziel der Geisteswissenschaftler, individualisierend einzelne historische Ereignisse oder Kulturprodukte zu beschreiben (idiographisches Verfahren). Diese Begriffsbestimmung gilt heute als wenig hilfreich, da die Sozial- und Humanwissenschaften typischerweise Aussagen treffen, die weder universell auf alle Individuen und sozialen Gebilde zutreffen noch singulär nur ein einzelnes Ereignis oder Erlebnis beschreiben“ [28, S. 299].

[2] In [28, S. 299] wird dafür „plädiert, solche Gegensätze nicht als Dichotomien, sondern allenfalls als bipolare Dimensionen aufzufassen und sie nur äußerst vorsichtig zu verwenden. Die Kategorien sind nämlich sehr stark durch Wertungen überdeckt und geben damit die Forschungspraxis verzerrt wieder.“ Kirk und Miller [108, S. 10] weisen darüber hinaus auf folgendes hin: „By our pragmatic view, qualitative research does imply a commitment to field activities. It does not imply a commitment to innumeracy. Qualitative research is an empirical, socially located phenomenon, defined by its own history, not simply a residual grab-bag compromising things that are not quantitative.“

Jahrzehnten“ des letzten Jahrhunderts als zunächst in weiten Teilen quantitative kommunikationswissenschaftliche Technik entwickelte Methode). Darüber hinaus ist sicherlich noch die Protokollanalyse (*protocol analysis* [59]), ein seit über 35 Jahren in der Kognitionspsychologie bekanntes Verfahren [98, S. 127] zu erwähnen.

Mayring [132, S.107] beschreibt die Grounded Theory[3] (GTM) als besonders geeignet für explorative Studien, in denen der „Gegenstandsbereich noch neu und unerforscht ist“. Nach Bortz und Döring [28, S. 332] propagiert sie ein „vorurteilsfreies, induktives und offenes Herangehen“, welches durch explizierte „Faustregeln“ diszipliniert wird und zielt, im Gegensatz zur qualitativen Inhaltsanalye nach Mayring [131], „die im Ergebnis eine Reihe von nur locker verbundenen Kategorien durch die Zusammenfassung von zugeordneten Textstellen beschreibt, [...] stärker auf eine feine Vernetzung von Kategorien und Subkategorien ab.“ Diese Eigenschaften der GTM führten vor dem Hintergrund der Tatsache, dass zu Beginn der Untersuchungen kaum qualitative Ergebnisse in Bezug auf den Prozess der PP existierten, auf die sinnvollerweise aufgebaut werden konnte, dazu, dass die GTM als geeignetes Instrumentarium für das Erreichen des hier beschriebenen Forschungsziels ausgewählt wurde. Alternative Methoden wie z.B. die Protokollanalyse schienen weniger passend zu sein, da sie zumindest mit einem partiell vordefinierten Kodierschema oder einem theoretischen Modell starten und darüber hinaus dadurch, dass sie in Hinsicht auf die Untersuchung kognitiver Prozesse entworfen wurden, spezialisierter als erforderlich sind.

In den folgenden Unterabschnitten wird eine Einführung in die GTM als ein handlungs- und interaktionsorientiertes Verfahren zur systematischen, qualitativen Analyse von Daten gegeben. Hierbei werden historische, prozedurale und auch erkenntnistheoretische Aspekte der Methode erörtert. Eine in dieser Weise umfangreiche Darstellung ist aus den folgenden Gründen notwendig:

1. Im Zusammenhang mit Untersuchungen von softwaretechnischen Fragestellungen kommt die GTM bisher nur selten zum Einsatz. Dementsprechend sind in diesem Bereich sowohl die grundsätzlichen Prinzipien wie auch die methodischen Details noch weitgehend unbekannt. Darüber hinaus genügen Durchführung und Darstellung oft nicht den von der Methodik avisierten Ansprüchen (z.B. bezüglich der Nachvollziehbarkeit des Analyseprozesses). Bei einigen GTM-Studien (im Software Engineering wie auch in anderen Disziplinen) handelt es sich sogar um eine unzulässige Etikettierung, durch die ein eher unstrukturiertes qualitatives Vorgehen (nachträglich) legitimiert werden soll [155; 203, S.147 bzw. S.633].

[3] Wie von Mey und Mruck [138] bemerkt, werden Übersetzungen des Begriffs, wie z.B. *Gegenstandsverankerte Theoriebildung*, kaum mehr verwendet. Vielmehr ist es wichtig zwischen einer „*Grounded Theory* als Ergebnis einer empirischen Studie (eben die generierte *Theorie*) einerseits und der *Methodologie der Grounded Theory* (GTM) anderseits“ zu unterscheiden. Achtung: Genau genommen muss man von der *Methode der Grounded Theory* und nicht von der Methodologie der Grounded Theory sprechen.

2. Ein humanwissenschaftlicher Untersuchungsgegenstand – und als ein solcher kann der Prozess der PP verstanden werden — „liegt nie völlig offen, er muss immer auch durch Interpretation erschlossen werden [132, S.22]“. Diese Interpretationen finden nicht unbeeinflusst statt. Das Vorverständnis wirkt sich auf die Interpretation aus. So ist es zum Zweck der Beurteilung der Ergebnisse unverzichtbar, dass die Beschreibung der Untersuchung und der Resultate einerseits das Vorverständnis des Forschers offen legt und andererseits nachzeichnet, wie dieses in den Forschungsprozess eingeflossen ist und diesen und damit auch das Ergebnis beeinflusst hat. Die diesbezüglichen Darstellungen können allerdings nur vor dem Hintergrund präziser Kenntnisse der verwendeten Methode bewertet werden.

3. Obwohl bei der Darstellung mancher existierender Studien der Eindruck entsteht, dass nur eine Form der GTM existiert, ist dies nicht so. Vielmehr existieren zumindest zwei unterschiedliche GTM-Strömungen. Für das Vorgehen bei der Forschung (und die spätere Beurteilung des Ergebnisses) ist es nicht unerheblich, welcher Strömung gefolgt wird.

4. Die Anwendung der GTM im Rahmen der in der vorliegenden Arbeit präsentierten Analysen von komplexem Videomaterial erfordert Erweiterungen der regulären Methoden (siehe Kapitel 5). Diese können nur vor dem Hintergrund eines vertieften Verständnisses der GTM beurteilt werden.

5. Das zentrale Ergebnis der vorliegenden Arbeit, die so genannte Basisschicht bzw. die so genannte Basiskonzeptmenge, ist dafür konzipiert, einen Ausgangspunkt für weitere, thematisch spezialisiertere GTM-Untersuchungen des PP-Prozesses zu bilden. In diesem Zusammenhang muss geklärt werden, in wie weit dieses Vorgehen (in Schichten) noch konform zur GTM ist. Hierfür müssen Vorgehen und Ziele der GTM detailliert verstanden sein.

Um das angestrebte Verständnis zu erreichen, werden zuerst die Grundprinzipien der GTM, wie sie initial von Glaser und Strauss [76] beschrieben wurden, vorgestellt (Abschnitt 3.1). Darauf aufbauend wird in Abschnitt 3.2 eine häufig verwendete Variante erläutert, die von Strauss und Corbin [198] entwickelt wurde. Sie konkretisiert viele bei Glaser und Strauss nur vage dargestellten Prozessaspekte und bildet die Grundlage für die in der vorliegenden Arbeit durchgeführten Analysen des PP-Prozesses. Abschließend (Abschnitt 3.3) wird auf die Unterschiede zwischen verschiedenen GTM-Zweigen eingegangen, genau genommen auf die zwischen der GTM nach Glaser und der GTM nach Strauss/Corbin. Im Rahmen der Erläuterung der GTM werden erkenntnistheoretische Positionen nur soweit erörtert, wie dies für die Diskussion über die Beeinflussung von Theoriebildung durch (theoretisches) Vorwissen relevant ist.

3.1 Grundprinzipien der GTM nach Glaser und Strauss

Die ursprüngliche Version der GTM wurde Anfang der 1960er Jahre von Barney Glaser und Anselm Strauss im Zusammenhang mit psychiatrischen und medizinsoziologischen Studien entwickelt [75; 200] und losgelöst von konkreten soziologischen Untersuchungen erstmals 1967 in „The Discovery of Grounded Theory: Strategies for Qualitative Research“ [76][4] veröffentlicht.[5] Obwohl viele der später erschienenen Varianten erhebliche Erweiterungen der Methode, speziell bez. der Strukturierung des Prozesses aufweisen, sind die in diesem Buch beschriebenen grundlegenden Prinzipien weitgehend gültig geblieben.

3.1.1 Ziel der GTM

Glaser und Strauss entwickelten die GTM mit dem Ziel, eine Methodik zur Verfügung zu haben, mit der die „Entdeckung von Theorie aus – in der Sozialforschung systematisch gewonnenen und analysierten – Daten vorangetrieben werden kann“ [77, S. 11], wobei sie dieses Bestreben durch die Beobachtung motivierten, dass in der Soziologie – zumindest bis in die 1960er Jahre hinein – „das Verifizieren von Theorien überbewertet und dementsprechend der vorherige Schritt, zu erkunden, welche Konzepte und Hypothesen für den Bereich, den man untersuchen möchte, überhaupt relevant sind, unterbewertet“ wird [77, S. 12].

Demzufolge wurden die durch die GTM zur Verfügung gestellten Verfahren nicht dafür konzipiert, eine vorhandene Theorie zu verifizieren. Bei ihrer Anwendung soll es vielmehr darum gehen, eine neue, so genannte *Grounded Theory* (GT) direkt aus Daten heraus entwickeln zu können. Hierbei müssen „Hypothesen und Konzepte nicht nur [initial] aus den Daten stammen, sondern im Laufe der Forschung systematisch mit Bezug auf die Daten ausgearbeitet werden“ [77, S. 15].

Die GTM stellt also das Erkunden von Daten zum Zwecke der Theoriebildung über menschliche Verhaltensmuster und soziale Prozesse in den Mittelpunkt, wobei trotz der Konzentration auf den Aspekt der Theorieentdeckung auch die Verifikation eine nicht unerhebliche Rolle spielt. Glaser und Strauss betonen dies, indem sie feststellen, dass „eine möglichst weitgehende Verifizierung als auch möglichst genaue empirische Belege notwendig zur Entdeckung und Generierung von Theorie“ gehören [77, S. 38]. Sie stellen aber heraus, dass die Verifikation die Generierung nicht „abwürgen“ darf. Diese Beurteilung der Rolle der Verifikation im Rahmen der GTM ist diffizil, da nicht unmittelbar klar ist, ab welchem Punkt ein

[4] Die nachfolgenden Verweise und Zitate beziehen sich auf die deutsche Übersetzung von Axel T. Paul und Stefan Kaufmann mit dem Titel „Grounded Theory: Strategien qualitativer Forschung“ [77].

[5] Genau genommen veröffentlichte Barney Glaser schon 1965 einen Artikel [72], der ein zentrales Element der GTM, die Methode des ständigen Vergleichens, vorschlägt.

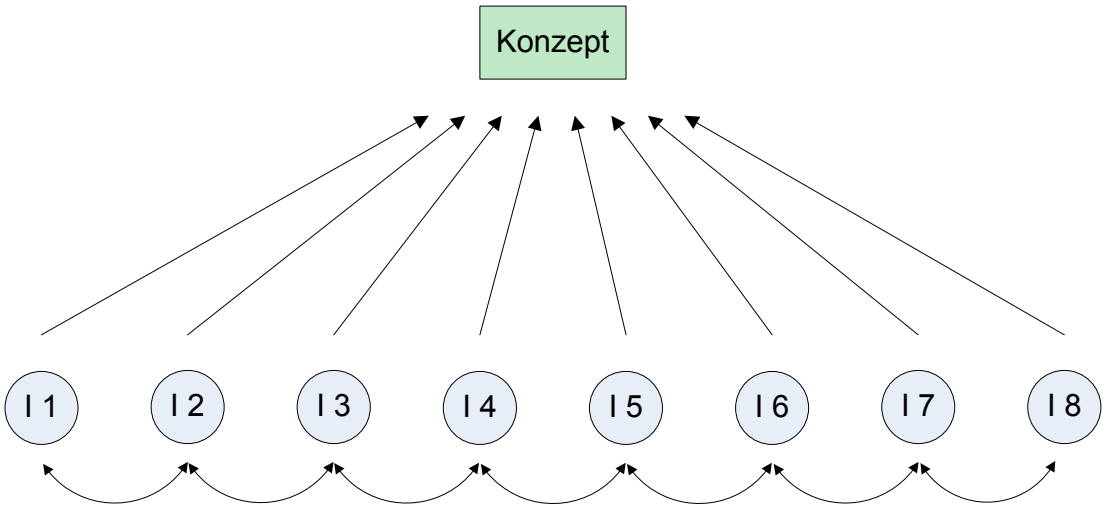


Abb. 3.1: Konzept-Indikator-Modell nach Strauss [197, S. 54f.] (Abb. entspricht Abb. 2 in [197, S. 54]): Konkrete Datenausschnitte werden mit einem Konzept versehen (kodiert), das heißt zu „*empirischen Indikatoren* [(hier *I1* bis *I8*)] für ein Konzept“ gemacht, welches der Forscher „zunächst vorläufig, später mit mehr Sicherheit aus den Daten ableitet.“ Der Forscher vergleicht die Indikatoren „fortwährend“ miteinander, um Ähnlichkeiten und Unterschiede herauszuarbeiten und so den „kleinste[n] gemeinsame[n] Nenner“ zu finden bzw. auf Konzeptebene Differenzierungen durchzuführen.

Forscher ausreichend Verifikation betrieben hat. Auf S. 57 wird hierauf genauer eingegangen.

3.1.2 Zentrale Begriffe

Die zentralen Tätigkeiten im Rahmen der GTM sind *Konzeptualisierung* und *komparative (vergleichende) Analyse*. Sie gehen Hand in Hand. Das Grundprinzip besteht darin, dass sich der Forscher beim Aufbau der Theorie auf *konzeptionelle Kategorien* stützt, die im Laufe der Analyse aus Tatsachen, sprich Datenausschnitten, extrahiert werden. Dabei kann „ein Konzept [...] aus nur einer einzigen »Tatsache» gewonnen werden“ [77, S. 33]. Diese Tatsache fungiert „dann als [...] einer vieler möglicher Indikatoren für das Konzept [...] Genau diese Indikatoren [sind es, die im Rahmen der] komparative[n] Analyse gesucht“ und miteinander verglichen werden [77, S. 33] – siehe hierzu auch Abbildung 3.1).[6] Sie dienen dazu, die einzelnen Konzepte und ihre Eigenschaften auszudifferenzieren, in Beziehung zu setzen und letztendlich Theoriebildung zu ermöglichen.

Die Begriffe *Konzept* (anderweitig auch *Kode* genannt (*concept* bzw. *code*)) und *Kategorie* (bzw. konzeptionelle Kategorie (*category*)) werden hierbei von

[6] Mit dem Begriff der komparativen Analyse werden in der Soziologie und Anthropologie vielfältige Anwendungsbereiche zusammengefasst. Glaser und Strauss verwenden ihn aber eingeschränkt (siehe z.B. [77, S. 33]): Die komparative Analyse dient der Generierung von Theorie und nicht explizit der Verifikation oder der Verallgemeinerung (Suche nach strukturellen Grenzen eines bekannten Tatbestandes).

Glaser und Strauss weitgehend synonym verwendet. Dies liegt darin begründet, dass die Übergänge zwischen beiden Termini fließend sind. Zur besseren Strukturierung des Analyseprozesses sollte aber zumindest theoretisch zwischen ihnen unterschieden werden können – wobei hier (zumindest vorläufig) auch zwischen Kodes und Konzepten differenziert werden soll, indem der Begriff Konzept als Oberbegriff für Kodes und Kategorien eingesetzt wird.[7]

Ein Kode ist „in der Regel [...] datennah", d.h. direkt an Ausschnitte der Daten (im Allgemeinen Textstellen) geknüpft, während eine Kategorie „Bestandteil der zu entwickelnden Theorie" ist [17, S.187].[8] Eine Kategorie bildet dabei oft einen Oberbegriff, der mehrere Kodes zusammen fasst, bzw. als abstrakteres Konzept höherer Ordnung eine Klassifikation von Konzepten [199, S. 43].

Darüber hinaus unterscheiden sich die beiden Begriffe aber auch in Bezug auf ihre innere Struktur. „Ein Kode ist im Wesentlichen eine Begriffsassoziation [...], eine Kategorie hingegen hat ein komplexeres Innenleben" [17, S.187]. Dieses Innenleben kann durch *Eigenschaften* formuliert werden. Sie werden wie alle anderen Konzepte von den Daten indiziert, stehen aber nicht für sich alleine, sondern bilden einen „konzeptuelle[n] Aspekt oder ein Element einer Kategorie" [77, S. 45].[9] Trotz dieser Unterscheidungsmöglichkeit wird im Rahmen der im vorliegenden Dokument beschriebenen qualitativen Untersuchungen fast ausschließlich der Terminus Konzept verwendet – auch wenn es sich in der Regel nicht um datennahe Begriffsassoziationen handelt. Auf den Grund hierfür wird in Abschnitt 6.1 auf Seite 125 eingegangen.

Mutmaßliche Beziehungen zwischen Kategorien bzw. Kategorien inklusive ihrer Eigenschaften werden von Glaser und Strauss als *Hypothesen* bezeichnet [77, S. 48f.]. Diese Definition steht im Einklang mit dem allgemeinen Hypothesenbegriff. Glaser und Strauss [77, S. 49] weisen allerdings explizit darauf hin, dass es bei der Hypothesengenerierung zunächst um eine Verankerung in den empirischen Daten geht und nicht darum, genug Material zu sammeln, um einen „Beweis führen zu können". Hypothesen dienen dazu, sich dem Datenmaterial zu nähern, und

[7] Kelle und Kluge weisen [107, S. 60] darauf hin, dass „der Begriff der ‚Kategorie' [...] manchmal zur Verwirrung [führt], weil er in der Literatur ziemlich uneinheitlich verwendet wird und von Begriffen wie ‚Konzept', ‚Merkmal', ‚Variable' oder ‚Kode' gelegentlich streng unterschieden, aber auch (manchmal nur wenige Seiten später!) mit diesen Begriffen gleichgesetzt wird."

[8] In einer späteren Veröffentlichung verwendet Strauss den Begriff Kode für alle Produkte des Analyseprozesses, seien es Kategorien oder auch Beziehungen zwischen Kategorien [196, S.21]. Der Grund hierfür dürfte darin bestehen, dass er unter dem Begriff Kodieren (*coding*) alle Analyseschritte, also z.B. auch das Aufstellen von Hypothesen, subsumiert (siehe hierzu auch Abschnitt 3.2 in dem vom offenen, axialen bzw. selektiven Kodieren gesprochen wird). Obwohl der Begriff Kodierung in der vorliegenden Arbeit konform zu Strauss verwendet wird, wird der Begriff Kode nicht in der allgemeinen Form von Strauss zur Anwendung gebracht.

[9] Der Eigenschaftsbegriff von Glaser und Strauss unterscheidet sich von dem aus der Informatik, speziell von dem der objektorientierten Programmierung, dadurch, dass die Eigenschaften einer Kategorie nicht notwendigerweise eine festgelegte Struktur besitzen. Vielmehr sind für Glaser und Strauss Eigenschaften Beschreibungen zu einem (in den Daten entdeckten) Aspekt einer Kategorie.

sollten über weite Strecken der Analyse explizit als etwas Vorläufiges angesehen werden.

Die aus den Daten herausgearbeiteten Kategorien, ihre Eigenschaften sowie die im Laufe des Prozesses aufgestellten, möglichst verallgemeinerten Hypothesen bilden schließlich zusammen das, was Glaser und Strauss unter einer Theorie, genauer einer *Grounded Theory*, verstehen (siehe z.B. [77, S. 52.]). Sie ist „nichts anderes als ein Ausdruck der in den Daten verborgenen Ordnung“ [77, S. 50.]. Glaser und Strauss unterscheiden in diesem Zusammenhang zwischen der Theorie und ihrer Darstellung. Dabei ziehen sie bei der Darstellung die fortlaufende theoretische Diskussion (Diskurs) einem kodifizierten Aussagegefüge vor. Sie begründen diese Entscheidung mit dem Prozesscharakter der Theoriegenerierung, welcher auch in der Darstellung des Ergebnisses erkennbar sein soll. Es muss klar werden, dass die dargestellte „Grounded Theory [...] kein perfektes Produkt [ist], sondern in permanenter Entwicklung begriffen. Zwar kann eine Theorie für die Publikation als ein fertiges Produkt behandelt werden, doch man hat in Rechnung zu stellen, dass sie noch weiter entwickelt wird“ [77, S. 41].

Im nächsten Abschnitt wird im Zusammenhang mit der Darstellung des Kodierprozesses erläutert, dass die geforderte Integration der Elemente hin zu einem Diskurs selbst zu einem Teil der komparativen Analyse werden kann.

3.1.3 Kodierprozess nach Glaser und Strauss

Der eigentliche Arbeitsprozess des Kodierens, angefangen bei der Generierung und Zuweisung von Konzepten zu einzelnen „Vorfällen“ in den Daten (das so genannte *Annotieren*[10]) bis hin zur Verdichtung der entwickelten Kategorien und dem Herausarbeiten von Beziehungen, also dem Generieren einer Theorie, wird von Glaser und Strauss in [76] nur grob ausgearbeitet. Erst nachfolgende Veröffentlichungen, z.B. von Glaser [73], Strauss [196] oder Strauss und Corbin [198] liefern hier klarere, aber nicht in allen Punkten identische und verträgliche Beschreibungen (siehe hierzu auch Abschnitt 3.3) „The Discovery of Grounded Theory“ legt allerdings vier ineinandergreifende Grundprinzipien des Kodierprozesses fest:

1. Iteratives Vorgehen
2. Theoretisches Sampling
3. Ständiges Vergleichen

[10] Im Allgemeinen wird die Zuordnung von Kategorien zu Textteilen Kodieren genannt [28; 50] (in den hier referenzierten Quellen wird tatsächlich der Terminus „Kategorie“ und nicht der Terminus „Konzept“ verwendet). Da dieser Begriff im Rahmen der GTM weiter gefasst verwendet wird, wird der reine Vorgang der Zuordnung von Kategorien (bzw. Konzepten) hier auch als Annotieren bezeichnet. Bei Glaser und Strauss handelt es sich diesbezüglich noch überwiegend um das Notieren von Kategorien an den Rand von Feldaufzeichnungen oder das Verwenden von Karteikarten. Heutzutage werden Daten eher elektronisch, unter Zuhilfenahme von QDA-Software annotiert. Dies gilt insbesondere für die in dieser Arbeit verwendeten Videos.

4. Ignorieren von vorhandener Theorie

Die Prinzipien sollen den Forscher leiten und demonstrieren, dass die GTM „in erster Linie ein Forschungsstil, eine Vorgehensweise, eine Haltung [...] und erst in zweiter Linie eine *einfache* Auswertungsmethode“ ist [138, S. 17]. Für sie gilt:

1. **Iteratives Vorgehen:** Glaser und Strauss wenden sich mit der GTM gegen die „traditionelle“ Auffassung „eines sequentiellen Vorgehens, in dem Planung, Datenerhebung, Datenanalyse und Theoriebildung getrennte Arbeitsphasen darstellen“ [138, S. 12f.]. Mey und Mruck [138, S. 13] betonen, dass Glaser und Strauss die Forschung als iterativen Prozess sehen, in dem ständig zwischen Datenerhebung, Datenanalyse und Theoriebildung hin- und hergewechselt wird, bzw. Datenanalyse und Theoriebildung teilweise sogar parallel ablaufen. Insbesondere legen sie dem Forscher nahe, mit der Datenanalyse sowie der Theoriebildung schon nach der Erhebung des ersten Datensatzes zu beginnen.

2. **Theoretisches Sampling:** Beim theoretischen Sampling handelt es sich um eine Methode zur Datenerhebung, die eng mit dem iterativen Vorgehensmodell der GTM verknüpft ist: Das Sammeln von Daten erfolgt in Teilschritten. Der Forscher entscheidet erst während der Analyse der in den vorhergehenden Schritten erhobenen Daten, welche Daten er im Weiteren benötigt, um die Generierung von Theorie voran bringen zu können.

   Die zugrunde liegende Idee ist, dass erst die im Entstehen begriffene Theorie, z.B. eine vage formulierte Hypothese, die nächsten Schritte anzeigt. „Der Soziologe kennt [diese Schritte] nicht, bevor der Forschungsprozess selbst ihn nicht vor neue Fragen stellt“ [77, S. 55].

   „Praktisch stellt sich das [Verfahren dann] als eine Kette aufeinander aufbauender Auswahlentscheidungen entlang des Forschungsprozesses dar, wobei die Auswahlkriterien im Verlauf des Projektes zunehmend spezifischer und eindeutiger werden [201, S. 30].“

   Glaser und Strauss [77] diskutieren zwei Vorgehensmodelle für das theoretische Sampling: Zum einen die Auswahl von geeigneten Vergleichsgruppen, zum anderen die so genannten *Datenschnitte*.

   Das Hauptkriterium für die Auswahl von Vergleichsgruppen ist für Glaser und Strauss „deren *theoretische Relevanz* für die Ausarbeitung emergenter Kategorien“ [77, S. 57]. Dies bedeutet, dass die Art und Anzahl der Gruppen derart gewählt werden sollte, dass „ihr Vergleich [dem Forscher] dabei hilft, möglichst viele Eigenschaften von Kategorien zu generieren und diese aufeinander zu beziehen“ [77, S. 57]. Es sollten also nicht möglichst viele Gruppen betrachtet werden, sondern vorrangig solche, die nach systematischer Betrachtung der bisherigen Ergebnisse zumindest theoretisch ein großes Potential für die Entdeckung neuer Aspekte besitzen. Hierbei

sollte man vor allem über Gruppenkriterien (z.B. Typen von Organisationen) nachdenken, die in Bezug auf den gerade interessierenden Aspekt eine möglichst kontrastierende Wirkung besitzen.

Bei den Datenschnitten geht es hingegen darum, unterschiedliche Arten von Daten, seien es Felddaten oder dokumentarische Daten, zu berücksichtigen bzw. unterschiedliche Techniken zur Datenerhebung anzuwenden. Hierdurch sollen verschiedene Ansichten auf die *Phänomene*[11] erzeugt werden, welche zur Sättigung (es werden keine neuen Erkenntnisse mehr erwartet (siehe auch S. 55)) der Kategorien beitragen können. Glaser und Strauss [77, S. 74] verweisen allerdings darauf, dass die Strategien zur Datenbeschaffung oft „durch strukturelle Zwänge bestimmt“ sind. Nicht immer können alle wünschenswerten Daten beschafft werden.[12]

Es ist wichtig, sich den Unterschied zwischen theoretischem Sampling und statistischem Sampling, der so genannten Zufallsauswahl, in Hinsicht auf Zuschnitt und Glaubwürdigkeit der Forschung bewusst zu machen [77, S. 70]. Während das theoretische Sampling auf die Entdeckung und Ausarbeitung von Kategorien und deren Eigenschaften und Beziehungen, letztendlich also auf eine Sättigung der Kategorien zielt, geht es beim statistischen Sampling darum, „Individuen empirisch exakt auf für die Beschreibung und Verifizierung notwendige Kategorien zu verteilen“ [77, S. 70].

Nach Glaser und Strauss [77, S. 70] müssen theoretisches und statistisches Sampling nicht kombiniert werden, solange es bei der Generierung von Theorie um Hypothesen über die Richtung von Beziehungen zwischen Kategorien bzw. Eigenschaften, aber nicht über deren Stärke bzw. Größen-

[11] Der Begriff Phänomen wird in [76; 77] (wie auch in [198] (siehe Abschnitt 3.2)) nicht explizit eingeführt. Er wird sogar teilweise synonym zum Begriff Kategorie verwendet. Dies deutet – in gewisser und überraschender Weise – auf ein Verständnis des Begriffs im Kantschen Sinne hin. „Kant unterscheidet zwischen den Phaenomena und den Noumena und weist beide Begriffe als transzendental aus. Die uns in Raum und Zeit (die beiden ›Vorstellungsformen‹ des Menschen) empirisch gegebenen Erscheinungen werden von Kant insofern als Phaenomena bezeichnet, als sie »als Gegenstände nach der Einheit der Kategorien gedacht werden«, was heißen soll, dass uns die Gegenstände nach der Erfahrung nicht anders als unter den subjektiven Bedingungen der Sinnlichkeit und des Verstandes gegeben sind. Die Erkenntnisvermögen des Menschen also konstituieren die erscheinende Welt gemäß den Kategorien, und die Erkenntnis der »Welt an sich« ist prinzipiell nicht möglich“ [173, S. 531]. Im Rahmen der vorliegenden Arbeit soll unter einem Phänomen eine auf die zu untersuchenden Daten bezogene Beobachtung verstanden werden, die mittels eines Konzeptes (oder auch einer Kategorie) benannt und beschrieben wird. Sie ist an einen Ausschnitt der Daten gekoppelt.

[12] Glaser und Strauss [77, S. 74] zitieren in diesem Zusammenhang Dalton [51], der eine Sekretärin bestechen musste, um geheime Personalakten einzusehen, in denen die ethnische Zusammensetzung einer Verwaltungshierarchie aufgeschlüsselt war. Dieses Beispiel ist sicherlich extrem, verdeutlicht aber Probleme bei der Datenbeschaffung, die auch im Zusammenhang mit den in dieser Arbeit behandelten Fragestellungen auftreten. So wäre es für die Untersuchung des Prozesses der PP hilfreich, Persönlichkeitsmerkmale der Paarmitglieder zu erheben (z.B. über den *Myers-Briggs Type Indicator* [146; 147]). Die Erhebung solcher Daten ist aber im professionellen IT-Umfeld schwierig.

ordnung geht. Erst in dem Fall, wenn ein Sample eine bereits aufgestellte Hypothese falsifizieren kann, werden „Unausgeglichenheiten“ des Samples diskutiert, bzw. als die Beziehungen beeinflussender (oder sogar ändernder) Faktor in die Theorie aufgenommen. Glaser und Strauss argumentieren also – wie sie selbst betonen – über die unterschiedlichen Forschungszwecke, wenn sie mit dem von ihnen propagierten Vorgehen eine entgegengesetzte Position zu Udy [210] einnehmen, welcher der Vermeidung von Samplingproblemen durch den Verzicht auf statistische Methoden widerspricht (siehe hierzu Fußnote 20 in [77]).

3. **Ständiges Vergleichen:** Der Kodierprozess der GTM, d.h. das Erzeugen und Annotieren von Konzepten bzw. Kategorien, das Ausarbeiten von Kategorien mittels Eigenschaften und das Inbeziehungsetzen von Kategorien oder von Eigenschaften von Kategorien, ist ein heuristischer Prozess, der im Dialog mit den empirischen Daten verläuft. Hierbei wird die Theoriegewinnung, wie oben bereits erläutert, mittels komparativer Analyse vorangetrieben. Es kommt die Methode des ständigen Vergleichens zur Anwendung, um Unterschiede sowie Ähnlichkeiten in den Daten zu entdecken. Glaser und Strauss beschreiben das ständige Vergleichen als einen Prozess über vier Phasen [77, S. 111ff.].

   In der ersten Phase [77, S. 111ff.] wird beim Annotieren eines Vorkommnisses durch eine Kategorie dieses Vorkommnis mit vorhergehenden der gleichen Kategorie verglichen. Dies führt nach Glaser und Strauss [77, S. 112] dazu, dass sich zügig theoretische Eigenschaften von Kategorien herausbilden können und der Forscher Charakteristika von Kategorien verstehen lernt.[13] Erkenntnisse, seien sie auch vorläufig, sollten hierbei unmittelbar in Form so genannter *Memos* (*memos*) festgehalten werden. Nur so können die (theoretischen) Einsichten des Forschers über den gesamten Prozess der GTM nutzbar gemacht werden.

   Die zweite Phase [77, S. 114ff.] widmet sich dem Integrieren von Kategorien und Eigenschaften. Kernidee hierbei ist es, dass im Laufe des Kodierprozesses auch zunehmend Vorkommnisse bez. verschiedener Kategorien bzw. Eigenschaften verglichen werden, um Verbindungen zwischen Kategorien bzw. ihren Eigenschaften zu erkennen. Die Vergleiche dienen also dem Zweck, einen „zusammenhängenden theoretischen Sinn zu eruieren“ [77, S. 115]. Dabei sollte, wenn es der Untersuchungsgegenstand erfordert, auch die innere Entwicklung von Kategorien bzw. ihrer zugrunde liegenden Vorkommnisse betrachtet werden, also verglichen werden, wie sich Eigenschaften einer Kategorie über die Zeit verhalten.

[13] Ab Kapitel 6 wird erläutert, wie Eigenschaften im Rahmen der hier vorgestellten Untersuchungen dazu verwendet werden können, Abgrenzungen zwischen Konzepten besser zu verstehen und nachfolgend durch den Split oder auch die Zusammenführung von Konzepten eine den Daten angemessenere, handhabbarere Konzeptmenge zu erhalten.

In der dritten Phase [77, S. 116ff.] wird durch das ständige Vergleichen sowohl die entstehende Theorie sowie auch der Analyseprozess selbst begrenzt. Dies geschieht jeweils auf unterschiedliche Weise.

In Hinsicht auf die Begrenzung der Theorie, bzw. genauer der Anzahl der Theorieelemente, wird die Menge der Kategorien und ihrer Eigenschaften auf Gleichartigkeiten untersucht und eruiert, ob die Theorie mit „weniger, aber abstrakteren Konzepten“ [77, S. 116] beschreibbar ist (*Reduktion*). Hierdurch werden nach Glaser und Strauss [77, S. 117] sukzessive zwei wesentliche Anforderungen von Merton [137] an Theorie erfüllt: „Erstens die Sparsamkeit in der Verwendung von Variablen und Formulierungen und zweitens Reichweite in der Anwendbarkeit der Theorie auf ein breites Spektrum von Situationen.“

Die Begrenzung des Prozesses bzw. der durchzuführenden Prozessschritte findet im Zusammenhang mit der Ausarbeitung der Kategorien statt. Hier stellt sich im Laufe der Untersuchungen die so genannte *theoretische Sättigung* ein. Der Forscher „lernt [...] schnell zu erkennen“, ob ein neues Vorkommnis bez. einer Kategorie neue Aspekte generiert [77, S. 117]. Kodierung und Vergleich von Vorkommnissen finden dann nur noch statt, falls dies der Fall ist.

Nach Glaser und Strauss beschleunigt diese Form des Aussparens von Kodierungen den Analyseprozess, ohne dass Einschränkungen bez. der Erkenntnisgenerierung zu erwarten sind. Sie begründen dies damit, dass die ausgelassenen Kodierungen keine neuen Informationen beitragen und außerdem im Rahmen der GTM im Allgemeinen nicht das Ziel verfolgt wird, Vorkommnisse pro Kategorie zu zählen. Allerdings verweisen sie auch auf Merton [137], der ausführt, dass durch das Zählen neue theoretische Ideen erzeugt werden können. Darüber hinaus sind vollständige Kodierungen auch im Zusammenhang mit der Untersuchung von Abläufen wichtig. In Abschnitt 3.2 wird hierauf genauer eingegangen.

Zusammenfassend betrachtet sind Reduktion und Sättigung zwei unterschiedliche Formen von Begrenzung, die aus der Anwendung des ständigen Vergleichens folgen, denen aber in der Regel durch das theoretische Sampling eine wohl begründete Ausdehnung folgt (siehe auch [77, S. 118]).

In der vierten Phase [77, S. 119], in der es um das Darstellen der Ergebnisse, also um das „Abfassen der Theorie“ geht und in der der Forscher kodierte Daten, Memos, Kategorien mit Eigenschaften und Beziehungen zur Verfügung hat, werden nach Glaser und Strauss jeweils alle auf eine Kategorie bezogenen Memos erst zusammengestellt und dann verglichen und zusam-

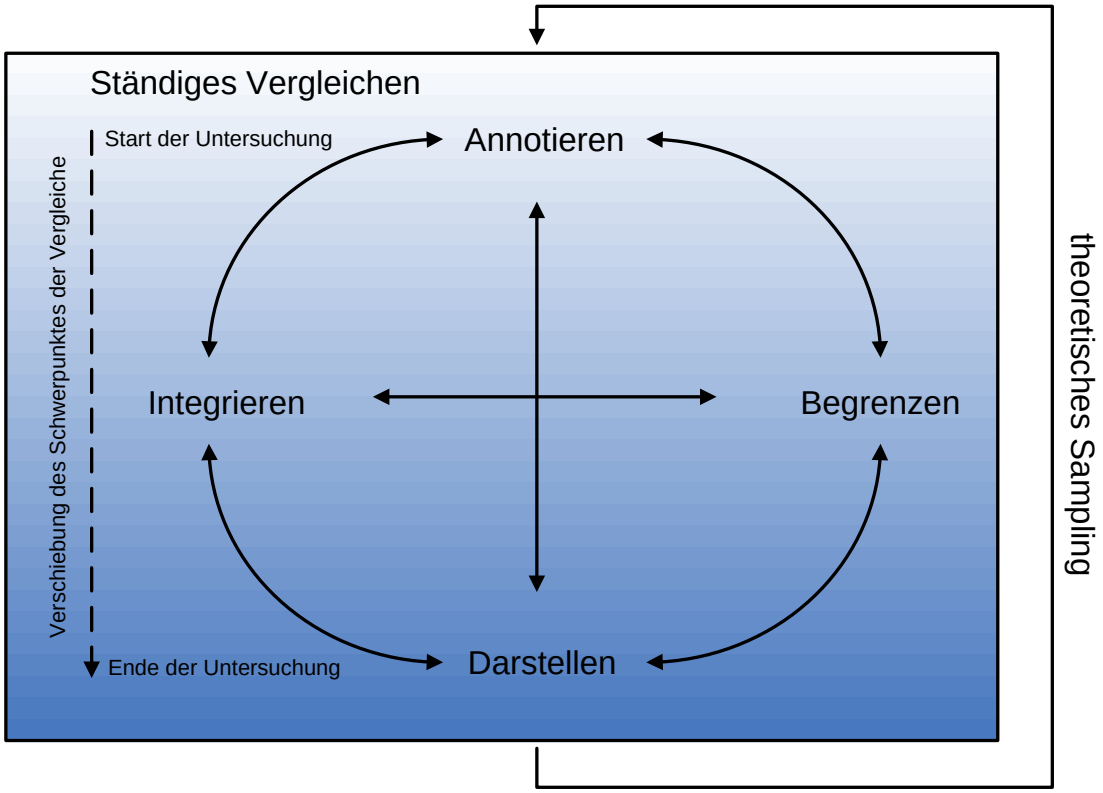


Abb. 3.2: Vereinfachte Darstellung der Interaktion der vier Phasen, in denen nach Glaser und Strauss [77] die Methode des ständigen Vergleichens verwendet wird: Zwischen den unterschiedlichen Formen von Vergleichen – beim Annotieren (und Ausarbeiten) von Kategorien, beim Integrieren von Kategorien und Eigenschaften, beim Begrenzen und Verallgemeinern der Theorie sowie beim Ausarbeiten der Theorie (z.B. anhand der geschriebenen Memos) – wird auch innerhalb eines durch das theoretische Sampling begrenzten Iterationsschrittes hin- und hergewechselt. Im Laufe der Untersuchung verschiebt sich aber der Schwerpunkt vom Annotieren auf das Darstellen.

mengefasst.[14] Lücken in der Theorie können so erkannt (und nachfolgend beseitigt) werden, bevor diese niedergeschrieben wird.

Die vier Phasen des Vergleichens können und sollen nicht sequentiell verlaufen sondern interagieren miteinander. Abbildung 3.2 verdeutlicht dies.

4. **Ignorieren von vorhandener Theorie:** Da die GTM den Schwerpunkt auf das Entdecken und Entwickeln von Kategorien legt, schlagen Glaser und Strauss vor, Literatur über Theorie bez. des Untersuchungsfeldes zunächst so weit wie möglich zu ignorieren und die Verwendung von Kategorien be-

[14] Das Zusammenstellen und Zusammenfassen von Memos hat bei Glaser und Strauss, die zum Zeitpunkt der Veröffentlichung mit handschriftlichen Notizen an bzw. in den Daten arbeiteten, noch eine andere Bedeutung als heutzutage, wo Daten, Kategorien und Memos elektronisch verarbeitet werden können, diese Schritte also besser in den Gesamtprozess integriert werden können.

stehender Theorien (externe Kategorien) zu vermeiden. Hierdurch soll verhindert werden, dass die entstehende Theorie durch unpassende, weil eher anderen Fragen angemessene Konzepte „kontaminiert“ wird.

Glaser und Strauss [77, S. 47] verweisen darauf, dass „Ähnlichkeiten und Konvergenzen mit der Literatur“ auch nach der Entwicklung der Kategorien analysiert werden können.

3.1.4 Beginn einer Untersuchung mittels der GTM

Wie bereits ausführlich erörtert startet eine auf der GTM basierende Untersuchung nicht mit einer oder mehreren Hypothesen. Hypothesen sollen vielmehr erst im Laufe des Prozesses entdeckt werden. Somit stellt sich die Frage, von welchem Punkt aus ein Forscher dann mit seinen Studien beginnen sollte, z.B. um Entscheidungen bez. der Erhebung bzw. Beschaffung der ersten Daten zu fällen oder erste Konzepte zu identifizieren.

In „The Discovery of Grounded Theory“ machen Glaser und Strauss hierzu kaum konkrete Angaben. Sie gehen implizit davon aus, dass der Forscher, wie bei allen anderen empirischen Untersuchungen, durch eine Fragestellung geleitet wird. Trutschkat et al. [209, S. 236][15] weisen allerdings darauf hin, dass diese im Zusammenhang mit der GTM „zu Beginn des Forschungsprojektes sehr offen formuliert [ist] und [...] erst im Verlauf der Forschung eine Präzisierung und Konkretisierung“ erfährt. Sie betonen, dass die „anfängliche Offenheit der Fragestellung [...] mit der abduktiven Forschungslogik der GTM zu erklären [ist]“ (siehe auch Exkurs 3)

In Kapitel 5 wird erörtert, wie diese (anfängliche) Offenheit, speziell im Zusammenhang mit den in der vorliegenden Arbeit geschilderten Untersuchungen, Probleme verursachen kann. Ferner werden Lösungsansätze in Form von zusätzlichen Praktiken vorgestellt.

3.1.5 Abschluss einer Untersuchung mittels der GTM

Bei der Verwendung der GTM ist nicht nur – wie gerade beschrieben – der Einstieg in den Prozess mit Schwierigkeiten verbunden. Schnell stellt sich auch die Frage, wann und wie man mit einer Analyse auf Basis der GTM zu einem Abschluss kommen kann. Diese gewinnt an Brisanz durch die Einsicht, dass eine Grounded Theory, wie auf S. 51 beschrieben, eigentlich nie fertig ist.

Glaser und Strauss verweisen diesbezüglich im Wesentlichen auf die Urteilskraft des Forschers. Er kann zu einem Abschluss kommen und seine Ergebnisse veröffentlichen, wenn er sich sicher ist, dass folgende, sehr allgemein formulierten Kriterien erfüllt werden [77, S. 229]:

[15] Diese Aussage wird von Trutschka et al. [209] in Bezug auf die GTM nach Strauss und Corbin [199, S. 23] gemacht. An dieser Stelle unterscheidet sich der Ansatz von Strauss und Corbin aber nicht wesentlich von dem ursprünglichen von Glaser und Strauss.

***Exkurs 3:** Abduktion und Grounded Theory*

Nach [173, S. 236] handelt es sich bei der Abduktion um einen „von Peirce 1866 eingeführten dritte[n] Modus des syllogistischen Schließens[a] neben Deduktion und Induktion. [...] Der abduktive Modus [schließt] vom Resultat und der Regel auf den Fall." So wird beim abduktiven Schließen „ausgehend von einer einzelnen Beobachtung wie beispielsweise ›Sokrates kann sprechen‹ und einer allgemeinen Gesetzmäßigkeit wie ›Alle Menschen können sprechen‹ auf einen hypothetischen Grund (oder eine Ursache) wie z.B. ›Sokrates ist ein Mensch‹ geschlossen" [173, S. 176]. „Die Abduktion stellt somit eigentlich nur eine Hypothese auf. [...] Im Gegensatz zum deduktiven Schließen, wo die Konklusion zwingend aus der Prämisse folgt, lassen Induktion und Abduktion streng genommen nur Wahrscheinlichkeitsurteile zu. Allerdings sind die Urteile im Gegensatz zur analytischen Deduktion synthetisch, d.h. Erkenntnis erweiternd" [173, S. 236].

Strübing [201, S. 45] weist unter Referenzierung auf Reichertz [174, S. 263] darauf hin, dass Peirce in seinem Spätwerk die oben erläuterte abduktive Schlussform „als »qualitative Induktion« ausdrücklich von der *Abduktion*" unterscheidet. „Der gravierende Unterschied zwischen beiden ist die Möglichkeit, tatsächlich *neues* Wissen zu gewinnen, wie es nur die Abduktion bietet." Strübing [201, S. 45] erläutert, dass Peirce im Rahmen der Abduktion den Schritt von der „vorsprachlichen Fassung [eines] Wahrnehmungsinhaltes [(*percept*)] zu einem »Wahrnehmungsurteil«" betrachtet. „Dieser Schritt erfolgt vermittels der Abduktion, bei dem das aktuelle percept mit den Erinnerungen vergangener percepte (also mit stilisierten percepten, Peirce bezeichnet das als »percipuum«) verglichen werden [sic]." Hierbei kann der Fall auftreten, dass man das aktuelle percept keinem bekannten percipuum zuordnen kann und man ein neues percipuum „erfinden" muss. Dieser „Prozess, der eigentliche »abduktive Blitz«, geschieht unwillkürlich: »Der Schluss von percept und percipuum auf ein Wahrnehmungsurteil liegt außerhalb jeder Kritik und jeder Kontrolle, er ist weder gut noch schlecht - er ist eben« [174, S. 269]." Strübing [201, S. 46] fügt hinzu, dass sich „in der grounded theory [...] Vorschläge zu einer Systematisierung und technischen Unterstützung dieser Prozesse ausmachen" lassen. Sie „zeigt auf, wie Abduktionen produktiv in den Forschungsprozess integriert werden können."

[a] Ein Syllogismus ist „ein Schluss mit zwei Prämissen und einer Konklusion" [173, S. 236].

- Der entwickelte „konzeptionelle Rahmen" bildet eine „systematische Theorie", also etwas fokussiert und strukturiert Beschreibendes, Erklärendes oder Vorhersagendes,
- was eine „hinreichend präzise" Antwort auf die ursprüngliche, bzw. sich weiterentwickelte Frage liefert
- und von anderen Forschern, die verwandte Gebiete untersuchen, verwendet werden kann.

Eine besondere Bedeutung kommt in diesem Zusammenhang der oder den Kategorien zu, die im Laufe der Untersuchungen in den Mittelpunkt der Theoriebil-

dung gerückt sind (*Kernkategorien*). Wenn diese auch nach umfangreichen Überlegungen und Anstrengungen bez. des theoretischen Samplings gesättigt sind, ist dies ein wichtiges Indiz dafür, dass die im Fokus stehenden Fragestellungen ausreichend untersucht wurden.

In diesem Zusammenhang sei noch einmal darauf hingewiesen, dass es sich bei allen Verfahren der GTM eher um Leitlinien als um Vorschriften handelt. Der Forscher muss im Einzelfall entscheiden, in welcher Weise ihn die Vorschläge zum Ziel bringen. Für den Fall des Untersuchungsabschlusses bedeutet dies, dass sich der Forscher vor allem davon überzeugen muss, dass er selbst seine Schlussfolgerungen und Darstellungen ohne jede Einschränkung (bzw. ohne jede nicht diskutierte Einschränkung) für glaubwürdig und relevant erachtet. Hierzu kann er die oben genannten Maßstäbe verwenden. Hierbei sollte er sich vor Augen halten, dass es nicht darum geht, sicherzustellen, dass ein anderer Forscher auf Grundlage der selben Daten die selbe Analyse mit den selben Ergebnissen durchgeführt hätte. Ein Grund hierfür besteht darin, dass die resultierenden Analyseschritte und somit Ergebnisse unter anderem von der so genannten theoretischen Sensibilität des Forschers abhängen. Diese wird im nächsten Abschnitt erläutert.

3.1.6 Theoretische Sensibilität und der Einfluss von Vorwissen

Nach Kelle [106, S.34] nähren die Darstellungen von Glaser und Strauss [76], die ausführen, dass „ein Forscher unvoreingenommen von theoretischen Vorüberlegungen an die Untersuchung empirischer Phänomene herangehen soll“, damit er die Realität so wahrnimmt wie sie tatsächlich ist, eine Vorstellung über die GTM als ein radikal induktivistisches Forschungsmodell („naiver Induktivismus“ [42, S.48]), „wie es ursprünglich im 17. und beginnenden 18. Jahrhundert von Vertretern des frühen Empirismus (wie Fancis Bacon, David Hume oder John Locke) entwickelt wurde“ und welches „in der modernen Erkenntnistheorie [...] als völlig überholt“ gilt.

Diese Wahrnehmung muss aber relativiert werden. Strübing [201, S.56] weist diesbezüglich darauf hin, dass Glaser und Strauss von einem Forscher erwarten, dass er „hinlänglich *theoretisch sensibel*“ ist. Dies bedeutet, dass er in der Lage sein sollte, in den Daten wichtige theoretische Zusammenhänge zu erkennen und mittels konzeptioneller Beschreibungen zu formulieren. Glaser und Strauss betonen, dass diese theoretische Sensibilität von den persönlichen Neigungen bzw. vom Temperament des Forschers und vom Ausmaß des Einsatzes vorher bekannter, festgelegter Kategorien (bzw. Theorien) geprägt wird [77, S. 54]. Sie schließen also keineswegs kategorisch die Verwendung von theoretischem Vorwissen aus.

Darüber hinaus ist Glaser und Strauss bewusst, dass ein gewisser Teil des Vorwissens eines Forschers unvermeidlich in den Analyseprozess eingebracht wird. Dies wird spätestens durch die Betonung der Tatsache offensichtlich, dass "vermutlich kein Soziologe, bevor er seine Forschung angeht, alle Theorie aus seinem Geist streichen“ kann [77, S. 257]. Glaser und Strauss folgen somit letztendlich

Hanson [89, S. 19], der feststellt, dass Beobachtungen nicht von allem theoretischen Einfluss losgelöst werden können, da das Sehen selbst eine theoriebeladene (*theory-laden*) Tätigkeit ist. Die Beobachtung von x wird durch das Vorwissen über x verformt.

Obwohl Glaser und Strauss also die Nützlichkeit von (theoretischem) Vorwissen nicht abstreiten bzw. festhalten, dass sich der Forscher seines Vorwissens nicht vollständig entledigen kann, muss festgehalten werden, dass sie letztendlich das Prinzip der theoretischen Sensibilität nicht wirklich mit der Idee integrieren, dass Konzepte ohne die Verwendung von vorgefassten Theorien oder Hypothesen aus den Daten *emergieren* [105].[16] Darüber hinaus erläutern sie nicht, in welcher Weise ein theoretisch sensibler Forscher das vorhandene Theoriewissen gezielt einsetzen kann, z.B. um der Gefahr aus dem Weg zu gehen, zu Beginn der Forschung in den Daten „unterzugehen".

Diese Themen werden erst in späteren Veröffentlichungen, z.B. von Strauss und Corbin [196; 198], adressiert und gehören zu den Faktoren, die in gewisser Weise zu einer Auseinanderentwicklung zweier unterschiedlicher GTM-Zweige geführt haben (siehe Abschnitt 3.3).

3.1.7 Zusammenfassung

Bei der GTM nach Glaser und Strauss [76] handelt es sich um eine Sammlung von Richtlinien, welche dazu dienen soll, Einsichten und Einfälle, die bei der strukturierten Durchsicht von Daten entstehen, in relevante konzeptionelle Kategorien, Eigenschaften und Hypothesen zu verwandeln, bzw. den diesbezüglichen Prozess zu befördern[17]. Ausgangspunkt ist dabei eine „grobe", auf einen bestimmten aber unbekannten Bereich bezogene Fragestellung.

Die zentralen Mechanismen der GTM sind das ständige Vergleichen und das theoretische Sampling. Sie werden in einem iterativen Prozess angewendet, der zur Sättigung der entwickelten Kategorien und zur Entwicklung von Beziehungen zwischen diesen, der so genannten Hypothesenbildung, führen soll.

Eine der wesentlichen Eigenschaften des Prozesses besteht nach Glaser und Strauss darin, dass die Ungenauigkeiten einzelner Daten durch die komparative Analyse sowie unterschiedliche Datenquellen korrigiert werden und die Integration der „Theorie die Ungenauigkeiten hypothetischer Schlussfolgerungen [...] tendenziell richtig stellt" [77, S.228].

Aus der Beschreibung von Glaser und Strauss ergeben sich allerdings – vor allem für einen Anfänger – drei bereits angesprochene Probleme:

[16] Zum Begriff *Emergenz* merkt Kelle [105] an: „GLASER and STRAUSS proposed a "general method of comparative analysis" which would allow for the "emergence" of categories from the data as an alternative to the hypothetico-deductive approach in social research."

[17] Genau genommen gehen Glaser und Strauss [77, S. 257] an dieser Stelle noch etwas weiter, in dem sie zulassen, dass persönliche Erfahrungen (oder sogar die Erfahrungen anderer), selbst wenn sie nicht in Form von Feldnotizen protokolliert wurden und weit zurück liegen, als Ideen zur Theoriebildung verwendet werden können oder sogar sollten.

1. Der Iterationsprozess ist nicht explizit in unterschiedliche Phasen aufgeteilt. Prinzipiell können alle Analyseschritte zu jedem Zeitpunkt stattfinden. Obwohl plausibel ist, dass bestimmte Aktionen verstärkt in bestimmten Phasen der Analyse verwendet werden, das Erschaffen neuer Konzepte also eher zu Beginn, das Integrieren von Theorie eher zum Ende der Untersuchungen stattfindet, stellt die GTM keinen klassischen Prozess bzw. kein „Kochrezept“ zur Verfügung.

2. Es wird keine konkrete Hilfestellung in Hinsicht auf die Identifikation von Konzepten, Kategorien oder Hypothesen in den Daten gegeben. Insbesondere „wird das Konzept der theoretischen Sensibilität [...] nicht in methodologische Regeln umgesetzt [106, S.36]“.

3. Die theoretische Sensibilität scheint im Widerspruch zum „Emergenz“-Anspruch zu stehen, da sie anscheinend eine gewisse inhaltliche Kompetenz erfordert.

Strauss und Corbin adressieren diese Probleme in ihrer Veröffentlichung von 1990 [198] und schlagen eine „gradlinigere“ Variante der GTM vor. Sie wird im nächsten Abschnitt beschrieben.

3.2 Die GTM nach Strauss und Corbin

„The Discovery of Grounded Theory“ war das erste und letzte Buch über die GTM, welches Glaser und Strauss zusammen veröffentlichten. Danach publizierten die Autoren ihre Sichten auf die GTM nur noch getrennt von einander, z.B. Glaser zuerst mit „Theoretical Sensitivity“ [73] und Strauss einige Jahre später mit „Qualitative Analysis for Social Scientists“ [196].

Beide Veröffentlichungen richteten sich vorrangig an fortgeschrittene Forscher. Erst 1990 veröffentlichte Strauss zusammen mit Juliet Corbin eine Darstellung der Ideen aus [196], die auch als Einführung für Anfänger konzipiert war: „Basics of Qualitative Research: Grounded Theory Procedures and Techniques“ [198][18]. Hier wird der Kodierprozess in Phasen aufgeteilt, es wird ein Kodierparadigma (das so genannte paradigmatische Modell) vorgestellt, es werden Regeln zum Umgang mit Vorwissen bzw. Literatur ausgearbeitet und Techniken zur Erhöhung der theoretischen Sensibilität empfohlen.[19]

Die folgenden Unterabschnitte beschreiben die Strukturierungsmaßnahmen sowie die wesentlichen Unterschiede bzw. Erweiterung der Variante von Strauss

[18] Die nachfolgenden Verweise und Zitate beziehen sich auf die deutsche Übersetzung von Solveigh Niewiarrar und Heiner Legewie mit dem Titel „Grounded Theory: Grundlagen Qualitativer Sozialforschung“ [199].

[19] Auch Glaser adressiert diese Probleme, allerdings mit abweichenden Methoden und Zielen (siehe z.B. [73]). Da die vorliegende Arbeit dem Ansatz von Strauss und Corbin folgt, werden die Ansätze von Glaser nicht oder nur in soweit dargestellt, wie dies zum besseren Verständnis des gewählten Weges beiträgt.

und Corbin im Vergleich zur ursprünglichen Version von Glaser und Strauss. Eine Darstellung und Beurteilung der von Glaser nach der Veröffentlichung des Buches von Strauss und Corbin formulierten – allerdings von Strauss nie direkt öffentlich erwiderten – Kritik folgt in Abschnitt 3.3.

3.2.1 Aufteilung des Kodierprozesses

Strauss und Corbin teilen den Prozess der Datenanalyse (Kodierprozess) in drei, im Allgemeinen nicht sequentiell zu durchlaufende Phasen auf:

1. Offenes Kodieren
2. Axiales Kodieren
3. Selektives Kodieren

Hierbei werden die verschiedenen Formen des ständigen Vergleichens, wie sie von Glaser und Strauss beschrieben wurden (siehe Abschnitt 3.1.3), in eine Art Prozessstruktur integriert:

1. **Offenes Kodieren:** Beim offenen Kodieren wird das vorliegende Datenmaterial, Strauss und Corbin nennen hier Beobachtungen und Interviews, aber auch Bücher, Videobänder, Statistiken und andere Dokumente, in möglichst kleine Einheiten, z.B. Sätze, Abschnitte oder Episoden, „aufgebrochen“ und konzeptionalisiert. Ziel ist es, eine Fülle von Kodes zu generieren, die in-vivo oder in „Anlehnung an soziologische Konstrukte“ benannt werden [138, S.29].

   Der Forscher führt hierbei drei Schritte parallel durch: Er identifiziert eine interessante Einheit, ein so genanntes Phänomen, und benennt und beschreibt es auf einer konzeptionellen Ebene. Letzteres erfolgt über spezielle Memos (so genannte *Kodenotizen* (siehe auch S. 76)), die dem annotierten Konzept zugeordnet werden. Dabei stellt er sich Fragen wie „Was ist das?“ oder „Was repräsentiert es?“ [199, S. 45]. Außerdem vergleicht er es mit anderen identifizierten Einheiten, „so daß ähnliche Phänomene den selben Namen [(das selbe Konzept)] bekommen können“ [199, S. 45].

   Strauss [196, S.28f.] betont, dass es zu diesem Zeitpunkt nicht darauf ankommt sofort die „wahre“ Bedeutung jedes Datenausschnittes zu finden. Vielmehr hat die Interpretation zu Beginn noch den Stellenwert eines Versuches. Der Forscher befasst sich in erster Linie damit, die Daten für die nächsten Schritte der Analyse nutzbar zu machen. Was auch immer hierbei nicht stimmig ist, wird in späteren Untersuchungsphasen berichtigt bzw. heuristisch erschlossen.

   Strauss und Corbin [199, S. 47] ergänzen, dass im Laufe dieses Annotationsprozesses gleichartige Konzepte möglichst zu Kategorien gruppiert werden

sollten.[20] Hierdurch kann die Anzahl von Einheiten, mit denen gearbeitet werden muss, reduziert werden. Kategorien bekommen abstraktere Namen, die helfen sollten, diese zu entwickeln, d.h. ihre Charakteristika in Form von Eigenschaften und potentiell zugehörigen Werten herauszuarbeiten.

Diese Ausdifferenzierungen von Kategorien durch Eigenschaften und ihre Werte – bez. des letzteren nennen Strauss und Corbin den Prozess *Dimensionalisierung*[21] [199, S. 43] – geschieht durch die Suche nach und Klassifikation von Ähnlichkeiten und Unterschieden in den zugehörigen Daten bzw. Datenausschnitten. Dabei entwickelt sich der „kategoriale Charakter" der Kodes im Allgemeinen umso stärker, desto weiter die Datenanalyse fortschreitet [17, S.187].[22]. Ein zentrales Ziel dabei ist, weitgehend jede Annotation einer Kategorie in Bezug auf die sich herausbildenden konkreten Untersuchungsziele durch eine Menge spezifischer Werte, das so genannte *dimensionale Profil*, möglichst genau beschreiben zu können. Tabelle 3.1 stellt ein aus dem Buch von Strauss und Corbin [199, S. 53] entnommenes Beispiel für eine Zuweisung von Eigenschaften und ihren Ausprägungen (Werten) zu einer Kategorie dar.

Die Dimensionalisierung muss und sollte aber nicht auf dieser Ebene stehen bleiben. So betont Strauss [196, S.14f.], dass es sinnvoll sein kann, Eigenschaften selbst wieder mit Eigenschaften zu versehen und zu dimensionalisieren (*Subdimensionalisierung*). Er ergänzt, dass sich diese Subdimensionalisierungen nicht allein aus den vorhandenen Daten ergeben müssen, sondern auch unter Einbeziehung von Kontextwissen (des Forschers) gewonnen werden können. Hierzu sollte sich der Forscher so genannte *generative* Fragen bez. der bisher gemachten Unterscheidungen stellen (siehe

[20] An dieser Stelle sei darauf hingewiesen, dass Strauss und Corbin den Terminus Kode nicht verwenden und trotzdem nicht klar zwischen Konzepten und Kategorien unterscheiden (siehe auch Fußnote 7 aus Seite 50). So legen sie fest, dass unter Konzepten „konzeptuelle Bezeichnungen oder Etiketten, die einzelnen Ereignissen, Vorkommnissen oder anderen Beispielen für Phänomene zugeordnet werden" verstanden werden sollen und unter einer Kategorie „eine Klassifikation von Konzepten [...] unter einem Konzept höherer Ordnung." Diese Klassifikation „wird erstellt, wenn Konzepte miteinander verglichen werden und sich offenbar auf ein ähnliches Phänomen beziehen" [199, S. 43]. Somit ist jede Kategorie bei Strauss und Corbin schon definitionsgemäß ein Konzept. Außerdem weisen Strauss und Corbin nur wenige Seiten später darauf hin, dass auch Konzepte gruppierende Wirkung haben sollten: „Wir vergleichen bei unserem weiteren Vorgehen Vorfall mit Vorfall, so daß ähnliche Phänomene denselben Namen bekommen können. Ansonsten würden wir zu viele Namen erhalten und sehr verwirrt werden" [199, S. 45]. Die Ebenen der Vergleiche bei Konzepten und Kategorien unterscheiden sich nur graduell. Schließlich spielen letztendlich auch bei Kategorien die Phänomene die entscheidene Rolle.

[21] Eigenschaften können sich dabei (zunächst) anonym herausbilden bzw. nur durch die Aufzählung ihre Werte definiert sein.

[22] Aus Sicht der Objektorientierung in der Informatik kann man die Kategorien als Klassen, Eigenschaften als deren Attribute und Annotationen als Instanzen einer Klasse sehen. Der in der GTM verwendete Begriff der Dimension bildet sich dann im Wesentlichen auf den Wertebereich von Objektattributen ab.

Eigenschaft der Kategorie *Beobachten*	**Dimensionale Ausprägung (pro Ereignis)**
Intensität	hoch - - - niedrig
Dauer	lang - - - kurz
Ausmaß	viel - - - wenig

Tab. 3.1: Fiktives Beispiel von Strauss und Corbin [199, S. 53] für Eigenschaften und Dimensionen anhand der Kategorie *Beobachten*. Diese Kategorie ist als ein Arbeitstyp einer bez. ihres Verhaltens untersuchten Person identifiziert worden: Die Person beobachtete in Abständen unterschiedliche Bereiche ihrer Umgebung. Das Auftreten eines Beobachtungsphänomens ließ sich durch drei Eigenschaften näher charakterisieren: Durch die Eigenschaft *Intensität* lässt sich für jeden Beobachtungsvorgang festhalten, wie stark der im Blickwinkel befindliche Bereich beobachtet wird. Die Eigenschaft *Dauer* macht eine Zeitangabe bez. des Beobachtungsvorgangs, die Eigenschaft *Ausmaß* dient zum Vergleich mit anderen unter Beobachtung stehenden Bereichen. Hierbei wird festgehalten, ob der Bereich mehr als andere Bereiche beobachtet wird. Die letzte Eigenschaft unterscheidet sich somit strukturell von den anderen beiden, da sie ein Verhältnis beschreibt. Ihre Ausprägungen können nur unter der Berücksichtigung von größeren Abschnitten des Prozessverlaufs bewertet werden.

S. 73). Auch Mey und Mruck [138, S.29] heben dieses Vorgehen heraus, indem sie sagen, dass im Zuge der Dimensionalisierung „theoretisch relevante Merkmalsausprägungen der jeweiligen Kategorie festgelegt und in einer begrifflichen Analyse expliziert“ werden.

Nach Strauss [196, S.15] hilft dieses Vorgehen vor allem dabei, das theoretische Sampling gezielter betreiben zu können. Die Datenerhebung kann in Hinsicht auf die potentiellen Merkmale (z.B. Subdimensionen) und die mit ihnen zusammenhängenden vorläufigen Hypothesen (z.B. in Form von Fragen) gesteuert werden.

Abschließend muss festgehalten werden, dass jede Eigenschaft zusammen mit einer ihrer Ausprägungen selbst auch ein Konzept (oder sogar eine Kategorie) ist, denn sie bilden eine (evtl. sogar ausdifferenzierbare) Bezeichnung, die einem Ausschnitt der Daten zugeordnet werden kann. Aus vorhandenen Konzepten können also mittels Eigenschaften und deren Ausprägungen neue Konzepte erzeugt werden, aus dem Konzept *Beobachten* z.B. das Konzept *lange Beobachten*. Diese Transformationsmöglichkeit beeinflusst direkt die in den nächsten Abschnitten beschriebene Generierung, Ausarbeitung und Verwendung der Menge der Konzepte zur Beschreibung des PP-Prozesses, also die Entwicklung und den Einsatz der so genannten *Basiskonzeptmenge* (siehe hierzu Kapitel 6f).

2. **Axiales Kodieren:** Beim axialen Kodieren soll „über das Klassifizieren hinaus die Interpretation und Erklärung“ vorangetrieben werden, indem die durch das offene Kodieren aufgebrochenen Strukturen in neuer Art zusammengefügt werden [199, S. 75f.]. Hierbei geht es nicht darum „mehrere Hauptkategorien in Beziehung zu setzen, um eine umfassende theoretische Formulierung zu erstellen“ [199, S. 76]. Dies geschieht erst im Rahmen des selektiven Kodierens (siehe S. 69). Strauss und Corbin betonen, dass man im Rahmen des axialen Kodierens „immer noch mit der Entwicklung von einzelnen Kategorien beschäftigt [ist], aber jetzt mit der Entwicklung jenseits der Eigenschaften und Dimensionen.“[23] Strübing [201, S. 27f.] konkretisiert dies, in dem er festhält, dass sich „das axiale Kodieren explizit einzelnen empirischen Vorkommnissen sowie deren Abstraktion zuwendet. Es geht nicht um die Beantwortung der umfassenden Forschungsfrage, sondern um die Erklärung des Zustandekommens und der Konsequenzen eines bestimmten Ereignisses. [...] Es wird nur die ›dünne Schicht‹ der Zusammenhänge rund um eines von einer ganzen Reihe von Phänomenen herausgearbeitet, die zunächst als solche verstanden und erklärt sein müssen, bevor wir eine umfassendere Theorie des untersuchten Feldes erarbeiten können.“

   Strauss und Corbin schlagen in diesem Zusammenhang die Verwendung eines Kodierparadigmas, des so genannten *paradigmatischen Modells* vor, um Kategorien in ein „allgemeines kausales Handlungsmodell“ einzuordnen [138, S. 29].[24] Der Zweck dieses Modells besteht im Wesentlichen darin, dem Forscher ein epistemologisches Schema an die Hand zu geben, mit dem er bewusst das menschliche Handeln und Interagieren, das Verwenden von Strategien beim Umgang mit Situationen bzw. Situationsinterpretationen und die daraus resultierenden Konsequenzen untersuchen kann [199, S. 76].

   Im Einzelnen geht es

   (a) „um Phänomene, auf die das Handeln gerichtet ist,

   (b) um kausale Bedingungen für diese Phänomene [(*ursächliche Bedingungen*)],

   (c) um Eigenschaften des Handlungskontextes [(*Kontext*)],

   (d) um *intervenierende Bedingungen*,

[23] Vor dem Hintergrund der hier im weiteren Verlauf erörterten, auf Strauss und Corbin zurückgehenden analytischen Arbeitsschritte des axialen Kodierens (siehe z.B. S. 68), ist diese Aussage irritierend. Schließlich wird im Rahmen dieser Arbeitsschritte auch vorgeschrieben, weitere Eigenschaften zu suchen. Allerdings handelt es sich hierbei in erster Linie um Eigenschaften in einem System von Ursachen und Wirkungen und nicht um solche, die im Rahmen eines offenen Kodierens entdeckt werden. Berg und Milmeister [17, S. 199] formulieren dies wie folgt: „Wir können der Analyse weiter auf die Sprünge helfen, indem wir nicht nur ‚assoziativ‘ vorgehen, sondern als heuristischen ‚Schmierstoff‘ der Analyse gewissermaßen axiale Modelle [...] benutzen.“

[24] Nach Strauss [196, S.28], ist ein Kodierverfahren, in dem die einzelnen Elemente des Paradigmas nicht berücksichtigt werden, kein echtes Kodieren.

(e) um *Handlungs- und Interaktionsstrategien* [der Akteure im Umgang mit dem Phänomen [201, S. 27]] oder

(f) um deren *Konsequenzen*“ (Kelle [104, S. 328] nach Mey und Mruck [138, S. 30]).

Zwei dieser Elemente bedürfen einer näheren Erläuterung, zum einen der Kontext, zum anderen die intervenierenden Bedingungen:

Kontext: Der Handlungskontext, oder auch *spezifische Kontext*, besteht nach Strauss und Corbin [199, S. 80 f] aus Ausprägungen der Eigenschaften des im Mittelpunkt der Betrachtung stehenden Phänomens (evtl. unter Hinzunahme von Eigenschaften der für das Phänomen ursächlichen Bedingungen). Hierbei handelt es sich um den spezifischen Ausschnitt der dimensionalen Profile der beteiligten Kategorien, welcher diejenigen direkt mit dem Phänomen zusammenhängende Informationen (Bedingungen) beschreibt, die die nachfolgenden Handlungs- und Interaktionsstrategien beeinflussen (können) bzw. in dem diese stattfinden. Vereinfacht ausgedrückt: Ein Kontext wird aus denjenigen konkreten Eigenschaftsausprägungen des Phänomens (oder auch der Ursachen des Phänomens) gebildet, die die Bedingungen, in denen Handlungen und Interaktionen stattfinden, bestimmen.

Intervenierende Bedingungen: Unter intervenierenden Bedingungen verstehen Strauss und Corbin den „breiteren strukturellen Kontext, der zu einem Phänomen gehört. Intervenierende Bedingungen wirken entweder fördernd oder einengend auf Handlungs- und interaktionale Strategien ein, die innerhalb eines spezifischen Kontexts eingesetzt werden“ [199, S. 82]. Im Gegensatz zum spezifischen Kontext handelt es sich bei den intervenierenden Bedingungen also um Faktoren, die nicht unmittelbar Eigenschaften des im Zentrum der Untersuchung stehenden Phänomens sind. Diese Faktoren können z.B. Zeit, Raum, Kultur, sozialökonomischen Status, technischen Status, Karriere, Geschichte sowie individuelle Biographien beschreiben.

Im Rahmen des paradigmatischen Modells erklärt sich auch die Bezeichnung „axial“ für diesen Kodierschritt: Kategorien bzw. Phänomene werden entlang der Achse eines allgemeinen Handlungsmodells analysiert. In Abbildung 3.3 wird das paradigmatische Modell im Ganzen gezeigt und erläutert.

Das axiale Kodieren kann nach Strauss und Corbin [199, S. 86ff.] durch eine weitgehend simultane Durchführung von vier analytischen Arbeitsschritten durchgeführt werden - wobei die „analytischen Verfahren“ des „Stellens von Fragen“ und des „Ziehens von Vergleichen“ angewendet werden sollten:

(a) **Hypothetisches in Beziehung setzen**: Subkategorien (siehe Abbildung 3.3) werden hypothetisch zu einer Kategorie in Beziehung gesetzt.

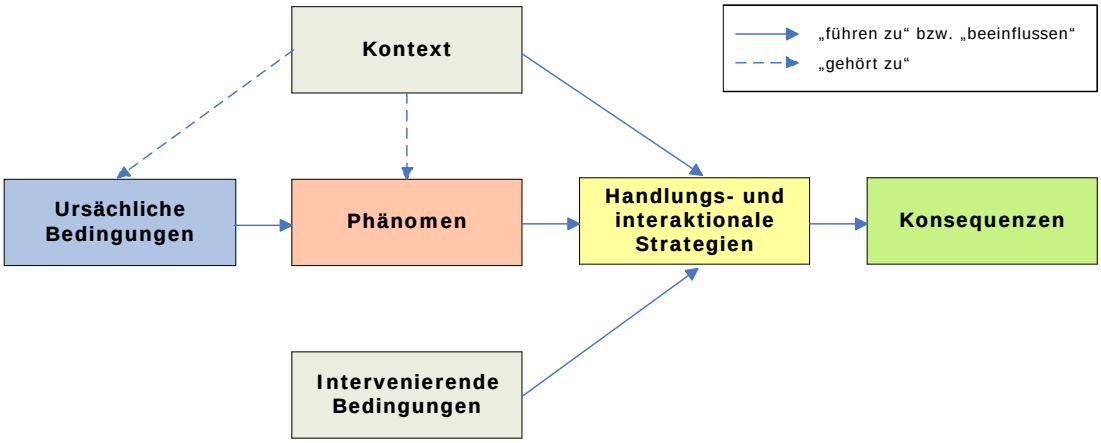


Abb. 3.3: Darstellung des paradigmatischen Modells nach Strauss und Corbin [199, S. 78]. Im Gegensatz zu einer Abbildung bei Strauss und Corbin, in der das Modell stark vereinfacht als lineare Abfolge skizziert wird (Ursächliche Bedingung → Phänomen → Kontext → Intervenierende Bedingungen → Handlungs- und interaktionale Strategien → Konsequenzen), sind die Zusammenhänge in der vorliegenden Graphik präziser dargestellt. Strauss und Corbin [199, S. 76] halten diesbezüglich fest: „Beim axialen Kodieren liegt unser Fokus darauf, eine Kategorie (*Phänomen*) in Bezug auf die *Bedingungen* zu spezifizieren, die das Phänomen verursachen; den *Kontext* (ihren spezifischen Satz von Eigenschaften), in den das Phänomen eingebettet ist [(, das heißt, die Menge von Eigenschaften, die im Sinne der Beschreibung auf S. 66 zum Phänomen „gehören“)]; die *Handlungs- und interaktionalen Strategien*, durch die es bewältigt, mit ihm umgegangen oder durch die es ausgeführt wird; und die *Konsequenzen* dieser Strategien. Weil diese spezifizierenden Kennzeichen einer Kategorie ihr Präzision verleihen, nennen wir sie *Subkategorien*“ [199, S. 76]. Die im Fokus stehende Kategorie repräsentiert dabei „die zentrale Idee, das Ereignis, Geschehnis, auf das eine Reihe von Handlungen/Interaktionen gerichtet sind, um es zu bewältigen oder damit umzugehen.“ [199, S. 79]. Man beachte: 1. In anderen Quellen wird der Begriff Subkategorie in ähnlicher aber nicht vollständig identischer Weise verwendet. Strübing [201, S. 20] spricht von Subkategorien, wenn als Folge des ständigen Vergleichens Unterscheidungen innerhalb einer Kategorie herausgearbeitet werden. Mey und Mruck [138, S. 30] verwenden den Begriff Subkategorie direkt als Synonym für den Begriff Eigenschaft bzw. als Übersetzung des Begriffes *property*. Alle Beschreibungen spiegeln in gewisser Weise die auf S. 64 erläuterte Verwandtschaft zwischen Eigenschaften und Kategorien wider. Im Rahmen der vorliegenden Arbeit wird der Begriff Subkategorie nur im Zusammenhang mit dem paradigmatischen Modell verwendet. 2. Strauss und Corbin [199] verwenden im Zusammenhang mit dem paradigmatischen Modell die Begriffe Phänomen und Kategorie parallel. Über die in Abschnitt 3.1.3 dargestellte Verwandtschaft der Begriffe hinaus liegt dies auch daran, dass die Verwendung des paradigmatischen Modells mit einem ständigen Wechsel zwischen induktivem und deduktivem Denken einhergeht. Siehe hierzu S.68.

Hierbei wird die Natur der Beziehungen anhand des paradigmatischen Modells bezeichnet.

(b) **Verifizieren der Hypothesen**: Die so entstandenen Hypothesen werden anhand der tatsächlichen Daten verifiziert. Es wird nach Hinweisen oder Ereignissen gesucht, die die Hypothesen bestätigen oder widerlegen bzw. ein tieferes, variationsreicheres Verständnis der gemachten Aussagen erzeugen.

(c) **Eigenschaften suchen**: Die „Suche nach [weiteren] Eigenschaften der Kategorien und Subkategorien, und nach der dimensionalen Einordnung der Daten (Ereignisse, Geschehnisse, etc.), auf die sie verweisen“ wird fortgesetzt. Es wird „Spezifität“ angestrebt, indem versucht wird, „jedes Ereignis [der gerade betrachteten Kategorie] in den Daten hinsichtlich seiner genauen dimensionalen Ausprägung zu bestimmen und zu spezifizieren.“ Der im Entstehen begriffenen Theorie wird „*konzeptionelle Dichte*“ verliehen, indem so viele Variationen wie möglich aufgedeckt werden.

(d) **Variationen von Phänomenen untersuchen**: In den dimensionalen Ausprägungen der Ereignisse wird nach Mustern gesucht, die der Charakterisierung von Variationen von Phänomenen, dienen können. Von besonderem Interesse sind hierbei Gruppen spezifischer, auf ein Phänomen bezogener Eigenschaften von Bedingungen, Strategien und Konsequenzen, die miteinander „interagieren“ (z.B. Maßnahmen in Hinsicht auf ein Phänomen, die unter bestimmten Bedingungen nicht funktionieren und deshalb dazu führen, dass bestimmte Strategien „mit der Zeit“ auf eine bestimmte Art geändert werden)

Der Prozess des axialen Kodierens hat somit zwei zentrale miteinander zusammenhängende Eigenschaften:

(a) Es wird kontinuierlich zwischen deduktivem und induktivem Denken hin- und hergewechselt. Einerseits werden allgemeine Aussagen über Beziehungen (zwischen Kategorien) aufgestellt und beim Verifizieren in speziellere Erkenntnisse (z.B. über Variationen) heruntergebrochen, andererseits werden konkrete Phänomene betrachtet und verglichen, um Eigenschaften zu extrahieren, die der Ausgestaltung einer konzeptionell dichteren Theorie dienen.

(b) Das Formulieren von Hypothesen wird durch die theoretische Sensibilität des Forschers beeinflusst. Denn selbst wenn das Aufstellen einer Hypothese nachweisbar durch eine konkrete Beobachtung in den Daten ausgelöst wurde, ist es wahrscheinlich, dass die Beobachtung selbst nur deshalb gemacht werden konnte, weil der Beobachter eine „gewisse“ Vorstellung über bestimmte Zusammenhänge mitbringt (siehe auch 3.1.6 bzw. [89]).

Beide Eigenschaften können zu Problemen führen. Zum einen erfordert die Parallelität bzw. der ständige Wechsel zwischen deduktivem und induktivem Denken (man kann auch von einem fließenden Übergang sprechen), dass sich der Forscher zu jedem Zeitpunkt des axialen Kodierens bewusst macht, ob er gerade über Eigenschaften einzelner konkreter Phänomene oder über allgemeine konzeptionelle Zusammenhänge nachdenkt bzw. spekuliert. Der Forscher muss also regelmäßig eine Metasicht auf seinen Forschungsprozess einnehmen. Zum anderen entsteht ein Konflikt zwischen der notwendigen theoretischen Sensibilität und dem Anspruch nach „Emergenz“ (siehe auch Abschnitt 3.1.6).

Um dem ersten Problem zu begegnen, wird in Abschnitt 5.2.3 ein UML-basiertes Modell zur Strukturierung der Analyseergebnisse und des Analyseprozesses vorgestellt. Das zweite Problem wird in Abschnitt 3.3 näher erörtert.

Abschließend sei noch auf Anmerkungen von Breuer [30, S. 86f.] zur Verwendung des paradigmatischen Modells hingewiesen. Er bemerkt, dass „das Paradigmatische Modell auf abstrakter Ebene einen Rahmen [liefert], dessen Logik für eine Vielzahl konkreter Theorieentwicklungen forschungspraktisch durchaus passt.“ Er fügt aber hinzu, dass sich „GTM-Forscher/innen [...] bei ihren Projekten [...] mitunter schwer [tun], diese Vorgaben bzw. die Begrifflichkeit des Kodierparadigmas durchgängig anzuwenden.“ In diesem Zusammenhang weist er darauf hin, dass die Unterscheidung zwischen Bedingungen und Konsequenzen oder zwischen Kontext und intervenierenden Bedingungen mitunter beliebig erscheint und „es [...] auch Fälle gibt, in denen die gesamte Modellierungslogik auf ein fokussiertes Thema nicht passen will.“ Er folgert, dass „die Vorgabe des Paradigmatischen Modells dafür geeignet [ist] zu zeigen, wie eine modellhafte Zusammenfügung der entwickelten Konzepte und Kategorien aussehen kann. Sie fordert zum Denken in Zusammenhängen und Bedingungsgefügen heraus. Allerdings muss das Konfigurieren von Kategorien in einer Gesamtarchitektur nicht in jedem Fall genau so und nach dieser Logik gemacht werden.“ Berg und Milmeister [17, S. 199]weisen darauf hin, dass „die Literatur [...] unterschiedliche Verfahrensweisen [bietet], die je nach Zielsetzung und Fragestellung mehr oder weniger geeignet sind, die sich aber auch miteinander kombinieren lassen.“

3. **Selektives Kodieren:** Die Verfahren des offenen und axialen Kodierens reichen aus, wenn es um die reine Themenanalyse[25] oder Konzeptentwicklung geht. Für die Entwicklung einer Theorie sind aber noch weitere Schritte notwendig [199, S. 93]. Sie werden von Strauss und Corbin unter dem Begriff selektives Kodieren beschrieben.

[25] Die Themenanalyse macht „Gebrauch von der semantischen Eigenschaft von Texten oder Textstellen, [indem diese] nach inhaltlichen Gesichtspunkten klassifiziert werden [136, S.145].

Ziel ist es, die in den entwickelten Kategorien, Beziehungen, Memos etc. vorhandene Information aufzugreifen und systematisch zu einem Bild der Wirklichkeit zu entwickeln, „das konzeptuell, nachvollziehbar und vor allem gegenstandsverankert ist“ [199, S. 95]. Hierbei sollte sich der Forscher folgender Punkte bewusst sein (entnommen einer persönlichen Mitteilung von Paul Atkinson an Strauss und Corbin [199, S. 94ff.]):

- Das Ergebnis ist nicht die Lösung eines Puzzles oder mathematischen Problems, sondern muss neu erschaffen werden.
- Es kann nicht immer alles in eine Version gepresst werden.
- Es kann mehrere verschiedene Wege des Zusammenfügens geben.

Ausgehend von diesen Prämissen sollte der Forscher zuerst die so genannte *Kernkategorie*, das zentrale Phänomen, um das herum andere Kategorien integriert werden sollen, festlegen. Strauss und Corbin schlagen vor, zu diesem Zweck eine Geschichte, „eine beschreibende Erzählung oder Darstellung über das zentrale Phänomen der Untersuchung“ [199, S. 94], zu formulieren. Hierbei spielt die Beantwortung von Fragen wie „Was ist im Untersuchungsbereich am auffallendsten?“, „Welche Phänomene werden wieder und wieder in den Daten widergespiegelt?“ oder „Was halte ich für das Hauptproblem?“ eine entscheidende Rolle. Aufgrund der zurück liegenden intensiven Beschäftigung mit den Daten sollte es dem Forscher zum Zeitpunkt des selektiven Kodierens möglich sein, diese Fragen zu beantworten. Ihm wird klar geworden sein, „worum es in [seiner] Forschung eigentlich geht“ [199, S. 95].

Als Ergebnis sollte dann eine möglichst kurze Erzählung entstehen, die keine Fallgeschichte, sondern eine Zusammenfassung vieler Einzelgeschichten ist und nur das Wesentliche wiedergibt. Ausgehend von dieser beschreibenden Geschichte muss dann eine analytische Darstellung entwickelt werden. Hierzu wird die Geschichte zuerst konzeptionalisiert, in der Sprache von Strauss und Corbin in den so genannten *roten Faden der Geschichte* überführt. „Das bedeutet – wie beim offenen und axialen Kodieren – daß [zuerst] dem zentralen Phänomen ein Name gegeben werden muß (und daß es als Kategorie allmählich mit anderen in Beziehung gesetzt wird)“ [199, S. 98].

Hier kann der Fundus der bereits entwickelten Kategorien daraufhin analysiert werden, ob bereits eine passende Kategorie existiert, die abstrakt genug ist, um die Rolle der Kernkategorie einzunehmen. Ist dies nicht der Fall, muss eine solche durch eine geeignete Benennung des zentralen Phänomens geschaffen und in Bezug auf ihre Eigenschaften ausgearbeitet werden.

Nun können die anderen Kategorien unter Zuhilfenahme des paradigmatischen Modells mit der Kernkategorie in Verbindung gesetzt werden. Wobei der schwierige Teil darin besteht, die Kategorien in Bezug auf das paradigmatische Modell so an- bzw. umzuordnen, dass sie zur Geschichte passen.

Hierbei sollte der Forscher folgendes beachten: Sollte das Verbinden der Kategorien zu diesem Zeitpunkt nicht lösbare Probleme aufwerfen, so kann dies daran liegen, dass in der Logik der Geschichte etwas falsch ist oder fehlt. In diesem Fall sollte die Geschichte umgeschrieben bzw. neu erzählt werden [199, S. 106].

Strauss und Corbin [199, S. 102] heben in diesem Zusammenhang noch einmal hervor, dass die Verbindungen der Kategorien miteinander weit komplexer sind als einfache Beziehungen der Art „Ursache führt zur Konsequenz“. Dies liegt unter anderem an den intervenierenden Bedingungen, die beispielsweise erklären, warum eine Person eine Reihe von Strategien wählt, während eine andere das nicht tut.

Wie beim axialen Kodieren wird auch bei der Integration von Kategorien bez. der Kernkategorie „eine Kombination aus induktivem und deduktivem Denken angewendet, wobei [...] durchgängig zwischen dem Stellen von Fragen, dem Aufstellen von Hypothesen und dem Vergleichen hin- und her[ge]pendelt [wird]“ [199, S. 107]. Dabei wird verstärkt nach dem Auftreten von Mustern – wiederholt auftauchenden Beziehungen zwischen Eigenschaften und Dimensionen – bez. der Kernkategorie geforscht. Hierbei sind vor allem Muster interessant, die den Kontext bilden, also die Handlungen und interaktionalen Strategien beeinflussen. Anhand dieser Kontexte ist es dann beispielsweise möglich, die Theorie in all ihrer Vielfalt diskursiv darzustellen.

Es muss noch einmal festgehalten werden: Alle Schritte werden in Hinsicht auf die Geschichte durchgeführt. Sie wirkt „als ein Filter, ein Verstehensraster, eine Vorstrukturierung [...] [17, S.203]“ Die Daten werden hierbei noch einmal dahingehend durchforstet, was sie in Bezug auf die Geschichte hergeben und ob sie die angestrebte Integration der Kategorien (möglichst in jedem Fall) unterstützen.

Letztendlich ergibt sich die Theorie also nicht kanonisch aus den Daten, sondern beruht auch auf der theoretischen Sensibilität des die Geschichte „erfindenden“ Forschers [17, S.202]. Im Allgemeinen muss die Geschichte dabei im Laufe des selektiven Kodierens aufgrund der Daten zum Teil neu erzählt werden. Wie beim axialen Kodieren macht der Forscher hierbei Aussagen über Beziehungen, die mittels der Daten validiert bzw. in Kontexte eingebettet oder modifiziert werden. Hierbei werden neben der Kernkategorie weitere Kategorien ausgewählt und ausdifferenziert, die in diesem Zusammenhang „einer weiteren Verfeinerung und Entwicklung bedürfen“ [199, S. 94].

Analog zum ständigen Wechsel zwischen den verschiedenen Formen des ständigen Vergleichens (siehe Bild 3.2) werden auch offenes, axiales und selektives Kodieren nicht rein sequentiell durchlaufen. Der Forscher bewegt sich vielmehr zwischen den einzelnen Formen des Kodierens hin und her. Dies geschieht oft

unbewusst, insbesondere zwischen dem offenen und dem axialen Kodieren [199, S. 40]. Gründe bzw. Ursachen für den Wechsel zwischen den Kodierformen sind beispielsweise:

- Die Datensammlung verläuft iterativ, da die Analyse das Sampling der Daten leitet (siehe hierzu auch die Erläuterungen zum theoretischen Sampling in Abschnitt 3.1.3)
- Beim selektiven Kodieren werden relevante Konzepte entdeckt, die spärlich entwickelt oder nicht integriert sind.
- Im Rahmen von Maßnahmen zur Verbesserung der eigenen theoretischen Sensibilität wird der Blick auf bisher nicht berücksichtigte, aber potentiell interessante Aspekte geleitet (Details hierzu finden sich im nachfolgenden Unterabschnitt).

3.2.2 Sonstige Unterschiede und Erweiterungen zur GTM nach Glaser und Strauss

Neben der im letzten Abschnitt vorgestellten klareren Strukturierung des Kodierprozesses und der in diesem Zusammenhang vorgeschlagenen Verwendung des paradigmatischen Modells existieren weitere Unterschiede zwischen der GTM nach Strauss und Corbin und der ursprünglichen Version nach Glaser und Strauss. Hierbei handelt es sich um konkretere Aussagen zum Umgang mit Vorwissen und Literatur sowie damit zusammenhängend um nähere Erläuterungen zur Rolle der theoretischen Sensibilität. Darüber hinaus werden eine Reihe den Erkenntnisprozess maßgeblich unterstützende Praktiken und Werkzeuge weitaus detaillierter dargestellt.

Im weiteren Verlauf dieses Abschnittes werden diese Unterschiede und Erweiterungen soweit vorgestellt, wie sie einen Einfluss auf die in den nächsten Kapiteln dargestellten Untersuchungen hatten:

1. **Theoretische Sensibilität/Umgang mit Vorwissen:** In Abschnitt 3.1.6 wurde erläutert, dass Glaser und Strauss [76] vom Forscher so genannte theoretische Sensibilität erwarten, diese Erwartung aber nicht wirklich mit ihrem Anspruch auf „Emergenz“ der Theorie aus den Daten in Einklang bringen. Insbesondere bleibt bei ihnen unklar, welche Rolle das Vorwissen des Forschers in diesem Zusammenhang genau spielt und wie es zum Einsatz gebracht werden kann, ohne den Daten eine Theorie aufzuzwingen.

   Strauss und Corbin versuchen sich einer Lösung dieses Problemkreises anzunähern. Dazu betonen sie viel entschiedener die Wichtigkeit der Integration von Vorwissen und nennen als Quellen der theoretischen Sensibilität explizit Literatur sowie die berufliche und persönliche Erfahrung [199, S. 25ff.]. Nachfolgend stellen sie dann drei Verhaltensregeln auf [199, S. 28f.], die ein Zurückfallen auf reine Verifikation verhindern sollen:

(a) Der Forscher soll in regelmäßigen Abständen immer wieder einen Schritt zurücktreten und sich fragen, ob das, was er zu sehen glaubt (und z.B. bereits in Hypothesen formuliert hat), die Wirklichkeit der Daten (für alle beobachtbaren Phänomene) widerspiegelt.

(b) Er soll „alle theoretischen Erklärungen, Kategorien, Hypothesen und Fragen über die Daten, egal ob sie direkt oder indirekt aus Vergleichen, aus Literatur oder der Erfahrung stammen“, als provisorisch ansehen. Er soll beachten, dass speziell „geliehene“ Kategorien in der Regel immer kontextspezifisch und nie allgemeine Tatsachen sind.

(c) Der Forscher darf die Regeln der GTM nicht aufs Geratewohl verwenden, da diese einer Studie methodische Strenge verleihen und helfen, „vorurteilsartige Verzerrungen zu durchbrechen“.

Strauss und Corbin führen hier als Beispiel den Wechsel zwischen Erhebung und Analyse auf, der „nicht nur eine Datenerhebung auf der Basis von Konzepten, die sich für [die] bestimmte Forschungssituation als bedeutsam heraus gestellt haben [erlaubt], sondern [...] auch das Verifizieren von Hypothesen, während sie entwickelt werden [fördert], so daß [diese] die Wirklichkeit der untersuchten Situation erfassen.“ Ein Verzicht an dieser Stelle begünstigt das Einsickern von Vorurteilen und kann somit die Güte der Ergebnisse negativ beeinflussen.

Strübing [201, S.57] fasst zusammen, dass es Strauss und Corbin darum geht, das „Vorwissen kreativ und phantasievoll“ zu nutzen, aber gleichzeitig den systematischen Bezug zu den Daten im Blick zu behalten. Vorwissen soll nicht „als gültige Aussage über die Welt (oder das interessierende empirische Phänomen), sondern als Anregung zum Nachdenken über die untersuchten Phänomene aus verschiedensten Blickwinkeln [genutzt werden], also als Fundus *sensibilisierender Konzepte* in Blumers [21] Sinne.“

Um die theoretische Sensibilität, die nach Kelle [106, S.38] die Verfügbarkeit brauchbarer heuristischer Konzepte zur Identifizierung theoretisch relevanter Phänomene im Datenmaterial bezeichnet, zu erhöhen, schlagen Strauss und Corbin [199, S. 57ff.] eine Reihe von Techniken vor. Zu ihnen gehören:

– *Das Fragestellen:* Beim Fragestellen handelt es sich um eine zentrale Praktik der GTM, der im Rahmen der Erhöhung der theoretischen Sensibilität eine besonders wichtige Rolle zukommt. Ihr vorrangiges Ziel ist ein weiteres Aufbrechen der Daten. Es sollen *potentielle* Kategorien, ihre Eigenschaften und Dimensionen betrachtet werden, indem zunächst von bestimmten allgemeinen Fragen ausgegangen wird, „die gleichsam automatisch an die Daten gestellt werden können“: Wer? Wann? Wo? Was? Wie? Wie viel? und Warum?

Im Zusammenhang mit bereits entwickelten Kategorien (inkl. ihrer Eigenschaften und Dimensionen) können dann aus den allgemeinen

Fragen spezifischere abgeleitet werden. Strauss und Corbin geben hier zur Verdeutlichung ein Beispiel aus einer Untersuchung über den Umgang von Arthritispatienten mit ihren Schmerzen an. In dieser Studie wird die bereits identifizierte Kategorie Schmerzerleichterung betrachtet und es werden Fragen gestellt wie „Wer verschafft Menschen mit Arthristis Schmerzerleichterung?“, „Sorgen diese Menschen immer selber dafür oder tut das auch jemand anderes?“, „Begeben sie sich zum Beispiel jemals in eine Arzt-Praxis [...] um Schmerzerleichterung zu bekommen?“ etc.

Die Fragen erleichtern das Nachdenken über die Daten und können darüber hinaus, z.B. falls die vorliegenden Daten keine Antworten liefern, zur Steuerung der weiteren Datenerhebung verwendet werden.

- *Die Analyse eines einzelnen Wortes, einer Phrase oder eines Satzes:* Diese Technik dient dazu, den Forscher bez. bestimmter konkreter Aspekte für das zu sensibilisieren, was evtl. noch in den Daten zu finden ist bzw. ihm umgekehrt bewusst zu machen, welche Bedeutungszuweisungen er diesbezüglich jetzt oder in Zukunft in die Analyse mitbringt. Hierzu sollte er nach der Analyse eines Datensatzes einen Teil (Wort, Phrase oder Satz) heraus greifen, der ihm besonders interessant erscheint und über den er „eingehender nachdenken möchte“ [199, S. 62]. Er sollte dann frei über die Bedeutung dieses Teils assoziieren und sich Fragen zu weiteren Bedeutungen (auch konzeptueller Natur, z.B. bez. Bedingungen oder Konsequenzen) stellen. Obwohl er dabei vom gegebenen Kontext ausgehen sollte, ist es sinnvoll, diesen auch zu verlassen. Nur so kann sich dem Forscher die mögliche Bandbreite von Bedeutungen eröffnen und er kann sensibler an die Analyse weiterer Daten gehen.

- *Das Ziehen von Vergleichen:* Strauss und Corbin geben drei verschiedene Vergleichstechniken an, um auf der einen Seite Konzepte bzw. Kategorien besser verstehen bzw. weiter differenzieren zu können und es auf der anderen Seite zu ermöglichen, dass Vorannahmen durchbrochen werden. Bei der Flip-Flop-Technik[26] werden beispielsweise Kategorien unter kontrastierenden Kontexten angesehen. Diese Betrachtungen werden zunächst theoretisch gemacht. Dabei wird dem aktuell in den Daten betrachteten Kontext ein sich möglichst stark unterscheidender entgegengehalten, über dessen Eigenschaften durch das Stellen von Fragen reflektiert wird. Die Ergebnisse dieser Überlegungen werden dann in Bezug auf die interessierende Kategorie beleuchtet („vergleichendes Denken“) und somit z.B. potentielle neue Eigenschaften oder Fragestellungen in den Fokus der Betrachtungen gebracht. Der

[26] Die anderen beiden Techniken werden mit „Systematischer Vergleich von zwei oder mehr Phänomenen“ und „Weithergeholte Vergleiche“ bezeichnet (siehe [199, S. 66ff.])

Forscher kann dann „mit mehr Sensibilität für das Thema als vorher“ zu seinen Daten zurückkehren.

- *Das Schwenken der roten Fahne:* „Auch diese Vorgehensweise soll dem Analysierenden helfen, hinter das Offensichtliche in den Daten zu schauen“ [199, S. 70]. Niemals soll etwas für selbstverständlich genommen werden (denn es ist es oft nicht), speziell auch dann, wenn es mit Aussagen wie „Nie“, „Immer“ oder „Es kann unmöglich so sein“ unterstrichen wird. „Diese Wörter und Phrasen können als Signale angesehen werden, genauer hinzuschauen“ [199, S. 71]. Strauss und Corbin weisen darauf hin, dass der Forscher sich an solchen Stellen Fragen stellen sollte wie z.B. „Was passiert hier?“, „Was meinen sie mit *nie*? „Wie wird der Zustand von *nie* aufrechterhalten?“, „Was sind seine Konsequenzen?“ etc. Sie bezeichnen diese Form des Aufmerkens als Schwenken der roten Fahne.

Die von Strauss und Corbin vorgeschlagenen Techniken zur Erhöhung der theoretischen Sensibilität verdeutlichen noch einmal, dass die Autoren es für notwendig erachten, dass der Forscher sein Vorwissen gezielt und hinterfragend in den Analyseprozess einbringt. Dies gilt sowohl für den Beginn der Analyse wie auch für Momente des analytischen Stillstandes.

Durch das zeitweilige Verlassen der Untersuchungsdaten soll der Wahrnehmungshorizont des Forschers erweitert bzw. die „kreative Vorstellungskraft“ [199, S. 73] verbessert werden. Der Forscher wird für Indikatoren in den vorhandenen Daten sensibilisiert bzw. erkennt, dass er weitere Daten benötigt. Hierbei wird auch dafür gesorgt, dass Verzerrungen, Vorannahmen, Denkmuster und Wissen nicht die Sicht auf das blockieren, was in den Daten bedeutsam ist [199, S. 73]. Letztendlich muss der Forscher aber immer wieder zu den Daten zurückkehren, um Konzepte und Relationen anhand konkreter Belege zu finden, zu verifizieren bzw. auszudifferenzieren.

2. **Einsatz von Literatur:** Wie schon im Zusammenhang mit der theoretischen Sensibilität dargestellt sprechen sich Strauss und Corbin im Gegensatz zu Glaser und Strauss (siehe Abschnitt 3.1.3) explizit für die Verwendung von Literatur aus. Dies betrifft jede Art von Literatur, sowohl fachliche, z.B. Forschungsstudien, wie auch nichtfachliche, wie z.B. Biographien, Tagebücher, Manuskripte oder Medienveröffentlichungen. Nichtfachliche Literatur kann in den Augen der Autoren unter Umständen sogar als primäre Datenquelle verwendet werden, wobei ähnliches auch für fachliche Literatur gelten kann. Ihre Berücksichtigung kann nämlich neben der Verbesserung der theoretischen Sensibilität auch dazu führen, dass weiteres deskriptives Material erschlossen wird, welches ebenfalls mit Methoden der GTM ausgewertet werden kann. Darüber hinaus kann fachliche Literatur

   - den Einstieg in die Datenerhebung erleichtern (speziell wenn erste Interviews geplant werden müssen),

- bei Diskrepanzen zu den eigenen Daten zum Hinterfragen der eigenen Datenbasis bzw. deren Analysen führen,
- Ideen für bisher nicht berücksichtigte Situationen hervorbringen und
- letztendlich – wie schon bei Glaser und Strauss angesprochen – als ergänzender Gültigkeitsnachweis dienen.

Strauss und Corbin verorten den Einsatz von Literatur nicht in einer bestimmten Phase des Analyseprozesses. Er kann sich sowohl vor wie auch im Laufe der Studie als sinnvoll herausstellen. Sie befürworten sogar ein „echtes Wechselspiel zwischen Lesen von Literatur und Analysieren der Daten“ [199, S. 38]. Allerdings betonen sie, dass ein Forscher im Rahmen der GTM „Phänomene im Licht eines theoretischen Rahmens erklären [möchte], der erst im Forschungsverlauf selbst entsteht. [Er möchte] nicht dadurch eingeengt werden, daß [er] an einer vorab entwickelten Theorie festhalten [muss], die sich auf den untersuchten Wirklichkeitsbereich anwenden läßt oder auch nicht“ [199, S. 32]). Dementsprechend muss er darauf achten, dass „Kategorien und ihre Beziehungen an [den] primären Daten überprüft werden“ [199, S. 38]. Der Forscher sollte sich davor „hüten, ein Gefangener der Literatur zu werden“ [199, S. 38].

3. **Weitere Praktiken und Werkzeuge:** Strauss und Corbin explizieren eine Reihe weiterer Praktiken bzw. Werkzeuge, die bei Glaser und Strauss nur implizit oder am Rande dargestellt werden. Zu den im Analyseprozess unverzichtbaren gehören Memos und Diagramme.

   Bei *Memos* handelt es sich, wie schon im Zusammenhang mit der GTM nach Glaser und Strauss erläutert (siehe S.54), um Verschriftlichungen des (abstrakten) Denkens über die Daten, also um Aufzeichnungen, die (Zwischen-) Ergebnisse der Analyse enthalten. Strauss und Corbin unterscheiden drei grundsätzliche Typen [199, S. 169ff.]):

   (a) *Kodenotizen* (*code notes*): Hierbei handelt es sich um „Memos, die Ergebnisse der drei Formen des Kodierens beinhalten, wie zum Beispiel konzeptionelle Begriffe [...], paradigmatische Eigenschaften und Indikatoren für den Prozeß“ [199, S. 169]. Kodenotizen werden hauptsächlich im Zusammenhang mit dem offenen (siehe hierzu auch S. 62) und dem axialen Kodieren eingesetzt.

   (b) *Theoretische Notizen* (*theoretical notes*): Dies sind Memos, die theoretisch sensibilisierend oder zusammenfassend sind. Sie „enthalten die Produkte des induktiven und deduktiven Denkens über tatsächlich und möglicherweise relevante Kategorien, ihre Eigenschaften, Dimensionen, Beziehungen, Variationen [und] Prozesse“ [199, S. 169].[27] Theoretische

Notizen können in allen drei Kodierphasen eingesetzt werden.

(c) *Planungsnotizen* (*operational notes*): Diese Memos enthalten „Handlungsanweisungen für die eigene Person und das Forscher-Team [...]. [Sie] beziehen sich z.B. auf die Auswahl von Fällen, auf Interview-Fragen, mögliche Vergleiche, weiterzuverfolgende Ideen etc.“ [199, S. 169].

Strauss und Corbin betonen, dass sich die Memotypen speziell in Hinsicht auf die unterschiedlichen Kodierverfahren in eine Vielzahl von Untertypen differenzieren lasse. Diese unterscheiden sich dann aufgrund der verschiedenartigen Zielsetzungen bez. Länge, Form, Komplexität des Inhalts etc. Strauss und Corbin geben Beispiele für den Aufbau und Inhalt einzelner solcher Memotypen, legen ihre Form aber letztendlich nicht explizit fest. Sie stellen vielmehr heraus, dass ein Forscher bezüglich der Ausgestaltung der einzelnen Memotypen seinen eigenen Stil finden muss. Zur Unterstützung führen sie eine Reihe allgemeiner Hinweise auf[28]:

- Im Laufe des Forschungsprozesses steigt die Komplexität der Memos mit der Komplexität der Analyse. So sind z.B. Kodenotizen zu Beginn der Untersuchungen naturgemäß sehr einfach. Erst im Laufe der Untersuchung bekommen sie durch Vergleichen und Fragestellen eine komplexere Gestalt (z.B. in Form diverser Eigenschaften und ihrer Werte).
- In Bezug auf das offene Kodieren fangen „theoretische Notizen [...] dort an, wo [die] Kodenotizen aufhören“ [199, S. 178]. Hierbei kann es z.B. um potentielle weitere Eigenschaften bzw. deren Werte gehen, die provisorisch aufgelistet werden um sie nachfolgend zu validieren [199, S. 178].
- Bezüglich des axialen Kodierens reflektieren Memos die „erfolgreichen oder auch erfolglosen Versuche die beim offenen Kodieren aufgebrochenen Teile neu zusammenzufügen“ [199, S. 182].
- Im Rahmen des selektiven Kodierens sollten theoretische Notizen auch dazu verwendet werden, um eine erste deskriptive Wiedergabe dessen zu formulieren, worum es in der Forschung geht, also letztendlich um die Kernkategorie zu entdecken. Darüber hinaus werden die Memos

[27] Kodenotizen können durch Fortschreibung, Änderung und Erweiterung zu theoretischen Notizen werden. In diesem Zusammenhang weisen Strauss und Corbin darauf hin, dass ein Memo Aspekte bez. aller drei Untertypen enthalten kann, dass es aber speziell für Anfänger aus Gründen der Übersicht wichtig ist, zwischen den Typen zu unterscheiden.

[28] Strauss und Corbin geben auch ein Reihe von technischen Hinweisen, z.B. dahingehend, dass Memos (wie auch Diagramme) nach Typen klassifiziert, datiert, mit einem Verweis auf das Dokument von dem sie stammen versehen, in Bezug zu den betroffenen Kodes oder Kategorien gesetzt und ggf. untereinander verbunden sein sollten. Bei der Verwendung einer QDA-Software wird der Forscher bez. vieler dieser Anforderungen unterstützt.

in dieser Phase dazu verwendet, Beziehungen anderer Kategorien zur Kernkategorie zu formulieren.

Während Memos textuelle Beschreibungen unterschiedlichster Analyseergebnisse sind, handelt es sich bei *Diagrammen* im Wesentlichen um graphische Darstellungen von Beziehungen zwischen Konzepten. Sie entfalten ihren Nutzen, der ähnlich dem der Memos im Festhalten bzw. Erarbeiten von (Zwischen-) Ergebnissen und in der Verbesserung der theoretischen Sensibilität besteht, vorrangig im Laufe des axialen und selektiven Kodierens. Häufig (aber bei weitem nicht ausschließlich) werden hierzu Tabellen oder Matrizen verwendet, in denen die Beziehungen zwischen Konzepten und/oder Eigenschaften auch unter Berücksichtigung des prozedualen Charakters des Untersuchungsgegenstandes aufgetragen werden können (z.B. Eigenschaftsänderungen über zeitlich hintereinander angeordnete Phasen).

Memos und Diagramme sind somit zentrale Werkzeuge (bzw. Verfahren) bei einem Vorgehen nach der GTM, da erst mit ihnen der Analyseprozess bzw. die Analyseergebnisse protokolliert werden können. Bei ihrer Verwendung werden alle Zwischenergebnisse, Ideen bzw. provisorischen Hypothesen und evtl. auch Irrwege oder Lücken in Gedankengängen einbezogen. Darüber hinaus wirken Memos und Diagramme theoretisch sensibilisierend, helfen den Prozess der Datenerhebung zu steuern und bilden letztendlich den Ausgangspunkt für die Formulierung der Theorie [199, S. 192].

3.3 Aspekte der Auseinanderentwickung einzelner GTM Varianten

Wie bereits in der Einleitung zu Abschnitt 3.2 erwähnt, handelt es sich bei „The Discovery of Grounded Theory“ [76] um das erste und letzte Buch, welches Glaser und Strauss gemeinsam zum Thema GTM veröffentlichten. Danach stellten Glaser und Strauss ihre Sichtweisen zur bzw. auf die GTM nur noch getrennt voneinander vor. Spätestens in Folge der Veröffentlichung von „Basics of Qualitative Research: Grounded Theory Procedures and Techniques“ [198] kam es sogar zu einem „massiven Bruch“ zwischen Glaser und Strauss. Glaser warf Strauss in zwei nachträglich 1992 in „Emergence vs. Forcing: Basics of Grounded Theory Analysis“ [74] öffentlich gemachten Briefen „in rüdem Ton“ vor, „sich einseitig die Konzeption des gemeinsam entwickelten Forschungsstils der Grounded Theory angeeignet und sie zugleich in unzulässiger Weise verfälscht zu haben“ [202, S. 261]. Auf diese Kritik hat Strauss „nie [...] öffentlich geantwortet“ [202, S. 262]. Nach Strübing [202, S. 262] hat dies dazu geführt, dass „sich zwei ko-existierende Richtungen der Grounded Theory etabliert [haben], die beide das gleiche Label für sich beanspruchen.“ Strübing [202, S. 262] betont aber, dass diese These vom Entstehen von zwei „in wichtigen Punkten gravierend voneinander verschiedenen Verfahrensvorschlägen [...] durchaus umstritten ist.“ Er verweist hier auf

Mey und Mruck [139, S. 101]. Eine detaillierte Betrachtung der diesbezüglichen Standpunkte würde allerdings den Rahmen der vorliegenden Arbeit bei weitem sprengen. Es sei aber darauf hingewiesen, dass sich die von Strübing identifizierten Unterschiede weniger auf die „Oberfläche praktischer Verfahren“, sondern eher auf die „Intentionen und Zuschreibungen [...], mit denen Glaser einerseits und Strauss andererseits ihre Verfahren rahmen und die dazu jeweils geltend gemachten wissenschafts- und erkenntnistheoretisch fundierten Begründungen und Anschlüsse“ beziehen [202, S. 273]. Schließlich unterteilt auch Glaser den Prozess in ähnlicher Weise in Kodierprozeduren (*gegenstandsbezogenes Kodieren* (unterteilt in ebenfalls offenes und selektives Kodieren genannte Prozesse) und *theoretisches Kodieren* (Glaser [73] nach Mey und Mruck [138, S.26])) und schlägt in [73] gleich 18 „Kodierfamilien“ vor, mit „denen er den theoretischen Horizont der Forschenden erweitern [will]“ [202, S. 268]. Hierbei legt die „Kodierfamilie ‚The six C's‘ fast alle jene Heuristiken als theoretische Kodes nahe, die Strauss und Corbin im Kodierparadigma in Frageform vorschlagen: Ursachen, Kontext, Konsequenzen, Bedingungen“ [202, S. 269]. Somit ist es erstaunlich, dass Glaser [74] „Strauss und Corbin insbesondere den Vorschlag des Kodierparadigmas zum Vorwurf [macht], weil dieses dazu führe, den Daten eine theoretische Struktur überzustülpen, die die diesen möglicherweise nicht angemessen sei. [...] [Denn wo] das Kodierparadigma bei Strauss und Corbin nur den Charakter einer pragmatischen Heuristik hat, zielt Glaser [...] – und dies konterkariert die Idee von theoretischer Sensibilität als Selbstvergewisserung – auf die Rahmung der Kodierperspektive durch die Vorgabe einer umfangreichen Liste soziologischer Basiskonzepte““ [202, S. 269].

Neben diesem Dissens[29] über das „Aufstülpen“ von theoretischen Theorien ist besonders die Rolle der Verifikation zwischen Glaser und Strauss umstritten [202, S. 270]: „Soll [sich die Grounded Theory] darauf beschränken, auf Basis empirischer Daten Theorien zu entwickeln, oder sollen diese Theorien zugleich einer Überprüfung unterzogen werden?“ Strübing [202, S. 270 f] hält fest, dass die Position von Strauss „auf den Dreiklang von Induktion, Deduktion und Verifikation [setzt], wobei er die Verifikation im Sinne einer Überprüfung der Plausibilität und der praktisch-experimentellen Funktionsfähigkeit der an der Empirie entwickelten Theorie versteht, eine Überprüfung im Übrigen, die [...] von Strauss als Teil des Theoriebildungsprozesses und nicht als eine distinkte Arbeitsphase betrachtet wird. [...] Glaser hingegen lehnt die Vorstellung ausdrücklich ab, dass die Überprüfung einer Theorie untrennbar Bestandteil der Theoriegenerierung ist“. Strübing interpretiert die Haltung von Glaser dabei wie folgt [202, S. 270]: „Anstelle einer systematischen Überprüfung, ob die erarbeiteten Theorien auch wirklich leisten, was sie zu leisten vorgeben – also das fragliche Phänomen zu erklären – bietet Glaser die Einladung, den Ergebnissen schon deshalb einfach zu trauen, weil sie mit der Methode des ständigen Vergleichens erarbeitet wurden. Auf diese Weise re-etabliert er jenen objektivistischen Methodenglauben, der davon ausgeht, dass

[29] Das Wort Dissens ist an dieser Stelle genau genommen nicht ganz passend: Zum einen hat, wie bereits erwähnt, Strauss nie öffentlich auf Glasers Anschuldigungen reagiert und zum anderen stellt sich die Frage, ob dass, was Glaser als Dissens verhandelt, wirklich einer ist.

‚richtige' Methoden-‚Anwendung' praktisch automatisch zu korrekten Ergebnissen führt - ein Glaube, der seit der Wiederentdeckung qualitativ-interpretativer Methoden in den 1960er Jahren mit guten Gründen für überholt gelten sollte." Insgesamt kommt er zu dem Schluss, „dass Glasers Position nicht nur wissenschaftstheoretisch haltlos, sondern vor allem in sich inkonsistent ist, da die starke Betonung von Emergenz und die geforderte Vorwissens-Abstinenz von einem massiven Einbezug allgemein-sozialtheoretischer Konzepte konterkariert wird, mit dem Glaser sich implizit selbst widerlegt. Hinzu kommt, dass der Verzicht auf Verifikation eine unnötige und für ein wissenschaftliches Verfahren nicht akzeptable Beschränkung der Leistungsfähigkeit Grounded Theory-orientierter Analysen darstellt – eine Beschränkung, die ihre Begründung aus einem verfehlten, weil implizit hypothetiko-deduktiven Verständnis von Verifikation bezieht" [202, S. 263].

Die erläuterten Punkte machen deutlich, dass sich ein Forscher vor dem Hintergrund der Anschuldigungen von Glaser, den im Detail nicht vollständig identischen Vorgehensvorschlägen von Glaser auf der einen sowie Strauss und Corbin auf der anderen Seite und den verschiedenen Auffassungen in der Forschergemeinde bez. des Grades und der Ausrichtung des Unterschiedes zwischen den beiden Ausformulierungen der in „The Discovery of Grounded Theory: Strategies for Qualitative Research" [76] ursprünglich formulierten Ideen bei seinen Untersuchungen klar auf Glaser oder auf Strauss/Corbin beziehen sollte. Nur so wird sein Forschungsprozess eindeutig nachvollziehbar.

Kapitel

4 Daten und Datenerhebung

Boing Boom Tschak
Kraftwerk, Musik Non Stop

Im Folgenden wird sowohl die Art der Daten erläutert, die in den im vorliegenden Dokument dargestellten Untersuchungen analysiert wurden (Abschnitt 4.1), wie auch deren Herkunft (Abschnitt 4.2). Auf drei Paarprogrammierungssitzungen wird hierbei besonders eingegangen, da das zentrale Ergebnis der vorliegenden Arbeit – die so genannte Basisschicht (siehe Kapitel 6) – zu großen Teilen auf diesen fußt. Weitere in die Herleitung der Basisschicht eingegangene Sitzungen werden erst danach erörtert (siehe S. 102). Es handelt sich hierbei um solche, die primär im Rahmen von weiterführenden Untersuchungen bzw. Anwendungstests der Basisschicht betrachtet wurden. Eine Reihe von Phänomenen aus diesen Sitzungen half aber auch dabei, die Basisschicht weiter auszudifferenzieren. Um zu einer theoretischen Sättigung (siehe S. 55) zu kommen, ging insgesamt Datenmaterial aus sieben Paarprogrammierungssitzungen (mit einer Gesamtdauer von knapp über 13 Stunden) in die Herleitung der Basisschicht ein.

4.1 Art der erhobenen Daten

Um die Vorgänge bei der PP untersuchen zu können, wurden sowohl im universitären wie auch im professionellen Umfeld Paarprogrammierungssitzungen in Bild und Ton aufgezeichnet (siehe Abschnitt 4.2). Im Einzelnen wurde pro Sitzung Folgendes mitgeschnitten:

- **Videoaufzeichnungen der Entwickler:** Während der Sitzungen wurden die Entwickler mittels einer Webcam[1] aufgezeichnet (Auflösung: 176 x 144 oder 320 x 240 Bildpunkte; Bildwiederholungsrate: 4 oder 15 Frames/sec. (für Details siehe Tabelle 4.1)). Hierbei wurde einer der beiden folgenden Kamerablickwinkel gewählt (siehe auch Abbildung 4.1):
  - **Kamerablickwinkel 1 (KB1):** Die Kamera nimmt die Oberkörper der Entwickler frontal auf. Sie ist so positioniert, dass beide Oberkörper über weite Strecken des Aufzeichnungszeitraums in Vollansicht zu sehen sind, aber die Tastatur zu keinem Zeitpunkt im Bild ist. Vorteil: Körperhaltung, Gestik und in eingeschränkter Weise auch Mimik beider Probanden können, wenn die Entwickler sich nicht stark bewegen,

[1] Logitech QuickCam Pro 5000 (http://www.logitech.com/de-de/435/243 (Abruf: 06.03.2012))

verfolgt werden. Nachteil: Driverwechsel sind schwierig nachzuvollziehen.

- **Kamerablickwinkel 2 (KB2):** Die Kamera nimmt die Entwickler schräg in leichter Aufsicht auf. Sie ist so positioniert, dass die Tastatur im Bild ist, aber nicht sichergestellt ist, dass beide Oberkörper über den gesamten Aufzeichnungszeitraum in Vollansicht zu sehen sind. Vorteil: Driverwechsel sind einfach nachzuvollziehen. Nachteil: Körperhaltung, Gestik und Mimik der Probanden können über längere Zeiträume nicht verfolgt werden, da schon leichte Bewegungen der Probanden dazu führen, dass sich ihre Oberkörper nicht mehr vollständig im Bild befinden.

- **Audioaufzeichnungen der verbalen Kommunikation im Paar:** Die Äußerungen der Entwickler wurden entweder mit dem in der Webcam eingebauten oder zwei externen funkbetriebenen Ansteckmikrofonen[2] aufgezeichnet. Hierbei wurde darauf geachtet, dass auch Umgebungsgeräusche wie z.B. Stimmen und Geräusche im Hintergrund oder das „Klicken“ der Tastatur mit aufgezeichnet werden.

- **Bildschirmaufnahme (Screencasting):** Bei jeder Sitzung wurden die Bildschirminhalte (1024 x 768 Bildpunkte oder 1600 x 1200 Bildpunkte) als Video mit einer Bildwiederholungsrate von 4 Frames/sec. oder 5 Frames/sec. aufgezeichnet (für Details siehe Tabelle 4.1).

Zur Aufzeichnung des Videobildes, des Tons sowie des Bildschirminhaltes wurde die Software Camtasia Studio[3] eingesetzt. Die Framerate und die Größe des aufgezeichneten Videobildes wie auch die Framerate der Bildschirmaufzeichnung wurden an die Leistung der unterschiedlichen Aufzeichnungsrechner angepasst. Dies war besonders dann wichtig, wenn die Aufzeichnung direkt auf dem Entwicklungsrechner stattfand. Schließlich sollte die Aufzeichnung die Performance des Entwicklungsrechners und somit auch die Arbeit des Paares nicht beeinflussen. Die Video-, Audio- und Bildschirmaufzeichnungen wurden nachfolgend mittels Camtasia Studio zu synchronisierten Videos zusammengeführt (5 Frames/sec. oder 10 Frames/sec. (für Details siehe Tabelle 4.1)).[4] Hierbei wurde das Videobild rechts unten in das Video der Bildschirmaufzeichnung eingeblendet (siehe Abbildung 4.2). Die resultierenden Videos bildeten den zentralen Datenbestand, auf dem die GTM-Analysen durchgeführt wurden.

[2] audio-technica 700 Series Professional UHF Wireless Systems mit Lavaliermikrofon AT829 (http://eu.audio-technica.com/de/products/product.asp?catID=4&subID=30&prodID=3850 (Abruf: 14.03.2012))

[3] Camtasia Studio ist ein Produkt der TechSmith Corporation (http://http://www.techsmith.de/ (Abruf: 06.03.2012)). Es kam sowohl die Version 4 wie auch die Version 5 der Software zum Einsatz. Die Aufzeichnung wurde mittels der Teilapplikation Camtasia Recorder vorgenommen.

[4] Bei den resultierenden Videos kamen sowohl das „Audio Video Interleave (avi)“-Containerformat wie auch direkt der „Windows Media Video (wmv)“-Codec zum Einsatz.

Abb. 4.1: Beispiele für die zwei unterschiedlichen Kamerablickwinkel, die bei der Aufzeichnung von Paarprogrammierungssitzungen gewählt wurden. Das linke Bild zeigt Kamerablickwinkel 1. Es handelt sich um eine frontale Sicht auf die Oberkörper der Entwickler. Die Tastatur ist hier nie im Bild. Das rechte Bild zeigt Kamerablickwinkel 2. Hierbei handelt es sich um eine leicht schräge Aufsicht auf die Oberkörper der Entwickler, die zwar garantiert, dass die Tastatur immer im Bild ist, aber nicht, dass die (Oberkörper der) Probanden immer vollständig sichtbar sind. Ein Kamerablickwinkel, der die Vorteile beider hier beschriebenen Blickwinkel aufweist, konnte in den Aufzeichnungsumgebungen mit dem zur Verfügung stehenden Equipment nicht hergestellt werden. Im rechten Bild sind die verwendeten Ansteckmikrophone zu erkennen (siehe Halsausschnitt der Oberbekleidung der Probanden).

Sitzung (Kamera-blickwinkel)	**Auflösung/ Bildwiederhol-frequenz der Videoauf-zeichnung der Entwickler *(Bildpunkte/ Frames/sec.)***	**Auflösung/ Bildwiederhol-frequenz des Screencasts *(Bildpunkte/ Frames/sec.)***	**Format des zusammen-geführten Videos**	**Auflösung/ Bildwiederhol-frequenz des zusam-menge-führten Videos *(Bildpunkte/ Frames/sec.)***
ST1.1 (KB1)	176 x 144/5	1024 x 768/4	avi/MPEG-4	1024 x 768/5
PR1.1 (KB1)	176 x 144/5	1024 x 768/4	avi/MPEG-4	1024 x 768/5
PR2.1 (KB2)	320 x 240/15	1600 x 1200/5	wmv	1600 x 1200/10

Tab. 4.1: Übersicht über die technischen Daten der aufgezeichneten Videos. Es sind die Kennwerte derjenigen Videos aufgeführt, auf denen die in den Kapiteln 6 bis 9 erläuternden Ergebnisse primär fußen. Die darüber hinaus in den Analysen berücksichtigten Videos weisen ähnliche Kennwerte auf wie Sitzung PR2.1. Nähere Informationen zu den Sitzungen ST1.1, PR1.1, PR2.1 und den weiteren aufgezeichneten Sitzungen sind in Abschnitt 4.2 zu finden.

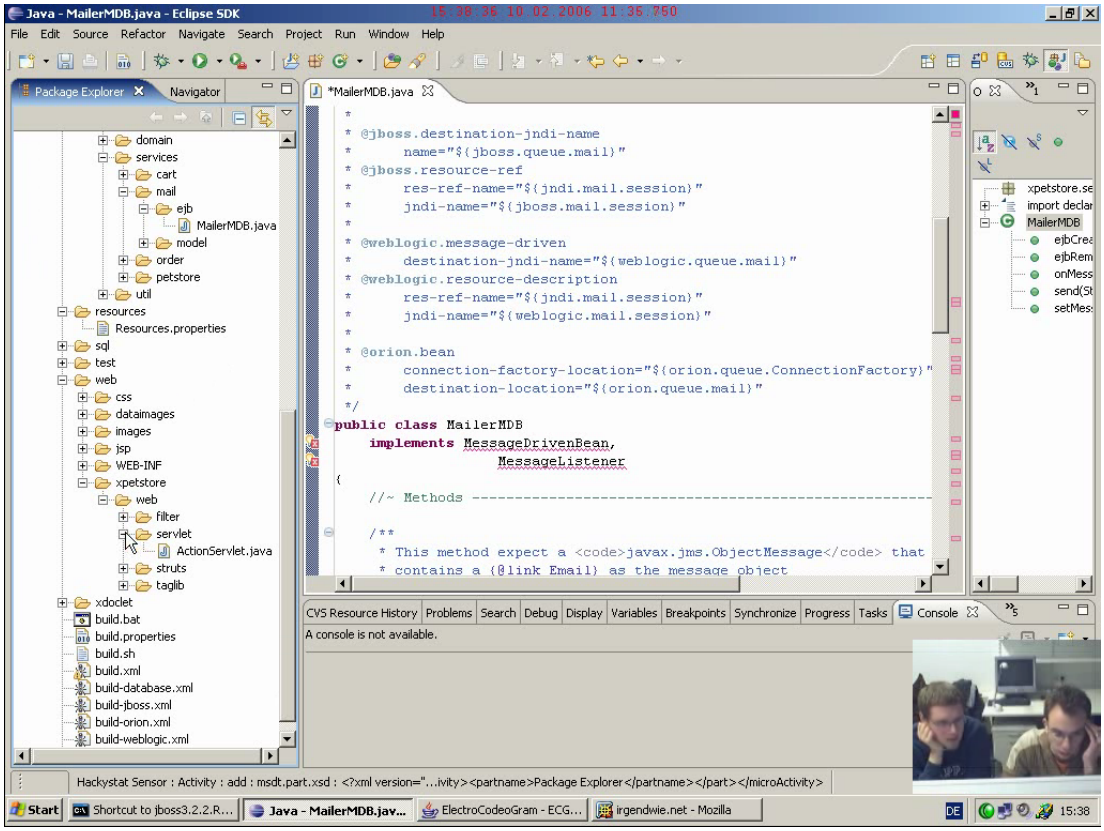


Abb. 4.2: Erläuterungen zum Aufbau der analysierten Videos anhand eines Standbilds: Zu sehen ist eine Bildschirmausgabe aus Sitzung ST1.1. Die Sitzung läuft zu diesem Zeitpunkt seit ca. 11 Minuten (Uhrzeit und Datum sowie Sitzungsdauer sind oben in rot eingeblendet). Die Entwickler haben die IDE Eclipse (siehe Fußnote 5 auf S. 85) im Vollbildmodus geöffnet – links ist der *Package Explorer* (siehe Fußnote *d* auf S. 193) zu sehen, in der Mitte im Editor die Datei `MailerMDB.java` und rechts der *Outline View* der Klasse `MailerMDB`. In der Statuszeile wird eine Meldung des ElectroCodeoGrams (siehe S. 85) angezeigt: „Hackstat Sensor: Activity [...]“ Unten ist die Taskleiste des Betriebssystems Windows zu sehen. Das Webcam-Video der Entwickler wurde mittels Camtasia Studio zeitlich synchronisiert unten rechts in den Screencast eingeblendet. Die zu allen anderen Sitzungen erstellten Videos weisen einen im Prinzip identischen Aufbau auf (siehe z.B. Abbildung 7.13 oder Abbildung 7.15). Achtung: Den Entwicklern war es erlaubt, im Rahmen der Sitzungen jedwede Applikation zu verwenden, die sie zu benötigen glaubten. Somit ist in den Videos keineswegs immer eine Entwicklungsumgebung zu sehen. In der Sitzung ST1.1 wurde z.B. ein Webbrowser geöffnet.

Neben den Sitzungsaufzeichnungen wurden in der Regel auch Fragebögen an die Entwickler herausgegeben. Auf diese wird im Abschnitt 4.2 eingegangen.

Bei den meisten Sitzungen (unter anderem bei ST1.1, PR1.1 und PR2.1) kam darüber hinaus das Eclipse-Plugin[5] ElectroCodeoGram[6] (ECG) [185] zum Einsatz. Dieses Werkzeug ermöglicht es, IDE-Ereignisse auf Mikroprozessebene aufzuzeichnen. Hierzu gehören sowohl Editier- wie auch Bedienschritte, z.B. das Eingeben von Zeichen oder Fokuswechsel[7]. Das ECG kennt verschiedene Klassen von Fokuswechseln, z.B. solche auf die Views der IDE Eclipse (Ereignistyp `msdt.partactive.xsd`), solche auf Editorfenster der IDE Eclipse (Ereignistyp `msdt.fileactive.xsd`) oder solche auf andere Applikationen[8] (Ereignistyp `msdt.windowactive.xsd`). Editiervorgängen werden vom ECG als so genannte Editierschritte protokolliert. Diese ergeben sich aus Ereignissen, die von der IDE Eclipse gemeldet werden. Es wird also in der Regel nicht jede Eingabe eines Zeichens als einzelner Editierschritt vermerkt. Die Anzahl der Editierschritte bildet somit ein nicht sehr genaues Maß für die Menge an Kode, die erzeugt, geändert oder gelöscht wird – nicht zuletzt auch dadurch, dass Kodeänderungen, die durch Wizards (wie z.B. *Refactoring*-Funktionalitäten der IDE) erzeugt werden, nicht eingehen.

Die resultierenden Daten wurden vor den GTM-Analysen dazu verwendet, einen groben Überblick über die Mikroereignisse einzelner Sitzungen zu bekommen (z.B. welche Dateien für wie lange in welcher Reihenfolge geöffnet wurden). Um dies zu erleichtern, wurde vom Autor des vorliegenden Dokumentes ein R-Programm[9] mit dem Namen *stripes* entwickelt, mit welchem ECG-Daten visualisiert werden können (Beispiele für solche Visualisierung sind in Anhang C zu finden). Für die eigentliche GTM-Analyse wurden die ECG-Daten nicht herangezogen.

[5] Bei dem Programm Eclipse (http://www.eclipse.org/ (Abruf: 26.01.2011)) handelt es sich um ein überwiegend als Entwicklungsumgebung (*integrated development environment* (IDE)) genutztes Open-Source-Framework, welches dem Programmierer im Rahmen von konfigurierbaren Perspektiven (*perspectives*) unterschiedlichste Informationen in Form von so genannten *Views* zur Verfügung stellt. Die Funktionalitäten der IDE Eclipse lassen sich durch so genannte Plugins erweitern. Über Plugins wird es beispielsweise ermöglicht, komfortabel in den unterschiedlichsten Programmiersprachen entwickeln zu können.

[6] Das Eclipse-Plugin ElectroCodeoGram wurde 2005 im Rahmen einer Diplomarbeit in der Arbeitsgruppe Software Engineering am Institut für Informatik der Freien Universität Berlin entwickelt (http://developer.berlios.de/projects/ecg/ (Abruf: 06.03.2012)).

[7] Der Begriff „Fokus“ bezieht sich hier auf Elemente von grafischen Benutzerschnittstellen und nicht etwa auf den Aufmerksamkeitsfokus einzelner Personen. Ein Teil einer Benutzerschnittstelle ist im Fokus, wenn er derart aktiviert ist, dass er als einziger Benutzereingaben (außer Fokuswechsel) empfangen kann.

[8] Mit der ersten Version des ECG konnte nur aufzeichnet werden, ob der Fokus auf Eclipse gelegt wird oder von Eclipse genommen wird. Erst nachfolgende Versionen konnten auch andere Programme diesbezüglich überwachen.

[9] R ist eine freie Programmiersprache für Statistik und Visualisierung (http://www.r-project.org/ (Abruf: 26.03.2012)).

4.2 Herkunft der verwendeten Daten

In den folgenden Unterabschnitten wird detailliert beschrieben, wo die im Rahmen der vorliegenden Arbeit analysierten Daten erhoben wurden. Hierbei wird in Abschnitt 4.2.1 auf diejenigen Daten eingegangen, die im Rahmen von studentischen Paarprogrammierungssitzungen und in Abschnitt 4.2.2 auf diejenigen, die im professionellen Umfeld bei Unternehmen erfasst wurden.

4.2.1 ST1: Experiment im Rahmen der Lehrveranstaltung „Bau betrieblicher Informationssysteme mit Java2 Enterprise Edition (J2EE)“

Gegen Ende der Vorlesungszeit des Wintersemesters 2005/2006 wurden die Teilnehmer einer Lehrveranstaltung am Institut für Informatik der Freien Universität Berlin vom Autor des vorliegenden Dokumentes[10] gebeten, an einem von ihm konzipierten Experiment teilzunehmen. Bei der Lehrveranstaltung handelte es sich um eine Vorlesung des Hauptstudiums mit dem Titel „Bau betrieblicher Informationssysteme mit Java2[11] Enterprise Edition (J2EE)“[12] (kurz BISJ2EE). Bei dem Experiment ging es darum, vor Ort zum Teil als Paare in einer Paarprogrammierungssitzung, eine Programmieraufgabe zu bearbeiten. Die Teilnahme an dem Experiment war für die Studierenden freiwillig.[13]

Das Ziel des Experimentes bestand grob gesprochen darin, Datenmaterial für die qualitative Analyse von Paarprogrammierungssitzungen zu sammeln. Es wurden folgende Randbedingungen festgelegt:

– Die zu lösende Programmieraufgabe sollte Aufgaben ähneln, die für gewöhnlich in der kommerziellen Softwareentwicklung zu bearbeiten sind. Dies sollte dadurch erreicht werden, dass nicht nur ein „Stand-Alone“-Programm zu

[10] Der Autor des vorliegenden Dokumentes war Übungsleiter der Lehrveranstaltung.

[11] Java ist eine objektorientierte Programmiersprache (http://www.java.com (Abruf: 13.03.2012)).

[12] „Die Vorlesung stellt die Aufgabenstellungen (nichtfunktionalen Anforderungen) vor, die beim Bau von BIS zu lösen sind, und behandelt die diversen Teiltechnologien von J2EE aus dem Blickwinkel, was sie jeweils dazu beitragen und wie (und wie nicht) sie eingesetzt werden sollten“ (https://www.inf.fu-berlin.de/w/SE/VorlesungBISJ2EE2004 (Abruf: 13.03.2012)). Mit BIS werden hier betriebliche Informationssysteme bezeichnet. Dies sind im Sinne der Vorlesung solche „Softwaresysteme, die direkt die Geschäftsabläufe (im Bürobereich) von Firmen unterstützen oder abwickeln. In der Praxis sind BIS heute groß, komplex, heterogen und ständigem Wandel unterworfen.“ J2EE ist eine nicht mehr gebräuchliche Abkürzung für das, was heute Java Platform, Enterprise Edition (Java EE) genannt wird.

[13] Um einen Leistungsnachweis für die Veranstaltung erhalten zu können, war es allerdings notwendig, die Aufgabe auch dann zu bearbeiten, wenn man sich nicht dazu entschied, an dem Experiment teilzunehmen. In diesem Fall konnte die Aufgabe innerhalb einer Woche unbeobachtet bearbeitet und beim Übungsleiter abgegeben werden.

entwickeln war, sondern auch eine Erweiterung eines den Programmierern in gewissem Maße bekannten komplexeren Softwaresystems.

- Für die Bearbeitung der Aufgabe sollte nur solches theoretisches Fachwissen von Nöten sein, von dem anzunehmen ist, dass es die Studierenden bereits besitzen. Die Aufgabe sollte also dahingehend konzipiert sein, dass sich die Studierenden nicht in neue Themenbereiche einarbeiten müssen. Dies sollte dadurch erreicht werden, dass in der Aufgabe ein in der Vorlesung und in den vorlesungsbegleitenden Übungen behandeltes Thema in den Vordergrund gestellt wird. Forciert werden sollte allerdings, dass es bei der Bearbeitung notwendig wird, in dem zu erweiternden Softwaresystem unbekannte Teile zu erkunden.

- Die Entwickler sollten in einer vertrauten Entwicklungsumgebung, das heißt wenn möglich auf ihrem eigenen und nicht auf einem für das Experiment gestellten Rechner arbeiten.

- Im Vorfeld sollte das Thema Paarprogrammierung nicht explizit erörtert werden. Insbesondere sollte es den Studierenden überlassen werden, wie sie ihre Zusammenarbeit im Detail gestalten. Dabei sollte nur insofern im klassischen Paarprogrammierungsstil gearbeitet werden, dass den Paaren jeweils nur ein Rechner mit einer Maus und einer Tastatur zur Verfügung steht.

- Zum Vergleich sollten auch Soloprogrammierer, die die gleiche Aufgabe bearbeiten, aufgezeichnet werden.

Eine Woche vor dem Experiment wurde eine Anleitung herausgegeben, in der erläutert wurde, wie ein evtl. vorhandenes Notebook bis zum Experiment zu konfigurieren sei, damit es im Experiment verwendet werden kann (in Tabelle 4.2 ist die zu verwendende Entwicklungsumgebung im Detail beschrieben). In dem Dokument wurde in erster Linie die Installation des Treibers für die Webcam, die Installation der Aufzeichnungssoftware Camtasia Studio und das Einspielen des ElectroCodeoGramm-Plugins beschrieben. Damit die Installationen vorgenommen werden konnten, wurden an die Studierenden Webcams[14] ausgegeben. In der Anleitung wurde auch darauf hingewiesen, dass sich auf dem Rechner eine lauffähige Version des Anwendungsservers (*application server*) JBoss[15] befinden muss. Auf entsprechende Dokumentationen wurde verwiesen. Da nicht alle Studierenden ein leistungsstarkes Notebook besaßen, wurden darüber hinaus Desktoprechner des Institutes vorbereitet. Auf ihnen wurde nicht nur die Beobachtungssoftware

[14] Siehe Fußnote 1 auf S. 81.

[15] Für Informationen zum JBoss Application Server siehe http://www.jboss.org/jbossas/ (Abruf: 04.04.2011). Eine JBoss-Installation war notwendig für die Bearbeitung der meisten Übungsaufgaben der Lehrveranstaltung. JBoss war den Studierenden also bereits bekannt. Es kam die Version 3.2.2RC2 zum Einsatz.

installiert, sondern auch von den Nicht-Laptopbesitzern im Vorfeld an den Veranstalter gesendete Eclipse Workspaces. Durch diese Maßnahme sollte erreicht werden, dass den Studierenden die Experimentumgebung vertraut ist und sie die für das Experiment relevanten, im Laufe der Veranstaltung von ihnen selbst entwickelten Programme vorfinden. Dies war insbesondere deshalb wichtig, da es in der Aufgabe darum ging, ein bestehendes System zu erweitern. Die Teilnehmer ohne eigenen Laptop sollten dieses System möglichst genau so vorfinden, wie auf dem Rechner, auf dem sie normalerweise entwickelten. Die vorbereiteten Recher waren also in gewisser Weise personalisiert. Es muss aber darauf hingewiesen werden, dass es bei Paaren trotz dieser Maßnahme dazu kommen konnte, dass nur einem der beiden Studierenden die Umgebung wirklich bekannt war. Diese Einschränkung galt aber auch für die Paare, die auf einem eigenen Laptop arbeiteten, und stellt in gewisser Weise eine Normalsituation bei der Paarprogrammierung dar.[16]

Da für die Durchführung des Experiments bis zu vier Stunden Zeit veranschlagt wurden und kein gemeinsamer Termin gefunden werden konnte, an dem alle Studierenden über ein solches Zeitintervall hinweg zur Verfügung stehen konnten, fand das Experiment über mehrere Tage verteilt statt.[17] Die Paarbildung erfolgte auf freiwilliger Basis. Dies geschah allerdings schon drei Wochen vor dem Experiment. Hier wurden die Studierenden aufgefordert, sich einen Partner zu suchen, um die aktuelle Übungsaufgabe mit diesem zusammen an einem Rechner zu bearbeiten.[18] Die Studierenden wussten zu diesem Zeitpunkt weder, dass sie mit dem hier gewählten Partner evtl. zusammen in einem Experiment arbeiten werden, noch dass es in dem Experiment um eine Umstellung des von ihnen in dieser Übung erstellten Kodes gehen wird. Zudem erfuhren die Entwickler erst am Tag des Experiments, ob sie im Paar oder alleine arbeiten sollen. Die Entscheidung hierüber wurde im Vorfeld ausgelost. Allerdings mussten zwei Paare neu zusammengestellt werden, da einzelne Studierende trotz Zusage nicht zum Experiment erschienen. Insgesamt nahmen sieben Paare und vier Soloprogrammierer an dem Experiment teil. Von den sieben Paaren arbeiteten zwei auf einem gestellten Rechner, von den Soloprogrammierern waren es ebenfalls zwei. Ein Soloprogrammierer hatte sich anderweitig einen Rechner ausgeliehen.

An den einzelnen Durchführungstagen wurde zuerst kommuniziert, wer im Paar und wer allein arbeiten sollte. Danach wurde der Aufgabenzettel ausgege-

[16] Natürlich sind auch Kontexte vorstellbar, in denen beide Entwickler eines Paares die Umgebung gleich gut kennen bzw. kennen können – z.B. wenn sie regelmäßig zusammen im Paar arbeiten oder in einem Unternehmen „genormte“ Umgebungen vorgeschrieben sind.

[17] Die Studierenden wurden aufgefordert, drei Wochen nicht miteinander über das Experiment, insbesondere die Aufgabe zu sprechen. Da kein Entwickler beobachtet werden konnte, der zu Beginn des Experimentes den Anschein machte, zumindest ungefähr zu wissen, wie die Lösung ausschaut, kann wohl davon ausgegangen werden, dass sich die Studierenden an diese Bitte gehalten haben.

[18] Es wurde nicht kontrolliert, ob diese Zusammenarbeit tatsächlich stattfand. In dem Fragebogen, der vor dem Experiment an die einzelnen Entwickler herausgeben wurde, wurden Paare allerdings gefragt, ob sie im Vorfeld schon einmal zusammen programmiert haben.

Betriebssystem	Microsoft Windows (in der Regel Windows XP)
Java Development Kit	J2SDK 1.4.2 oder neuer sowie JDK 1.5
IDE	Eclipse 3.1
Buildwerkzeug	Apache Ant 1.6[a]
Eclipse Plugin	ElectroCodeoGram
J2EE Applikationsserver	JBoss 3.2.2RC2[b]
Beispielanwendung 1	XPetstore[c] (inkl. Quellkode)
Beispielanwendung 2	Im Vorfeld von den Studierenden selbst geschriebene Java-Applikation, die den Email-Versende-Mechanismus von XPetStore nutzt, um Emails abzusetzen (inkl. Quellkode)
Dokumentation	XDoclet[d], JDK-API, J2EE-API, Vorlesungsfolien zu JMS[e], Übungsblatt 2 der Veranstaltung (enthält detaillierte Installationsanleitungen zu allen Komponenten)
Beobachtungssoftware	Camtasia Studio und Treiber für die Webcam

[a] Siehe http://ant.apache.org/ (Abruf: 19.03.2012).
[b] Siehe auch Fußnote 15 auf S. 87.
[c] Siehe auch Fußnote 19 auf S. 90.
[d] Siehe auch Fußnote 23 auf S. 90.
[e] Siehe auch Fußnote 21 auf S. 90.

Tab. 4.2: Verwendete Software im Experiment BSJ2EE. Es sind die in jedem Fall auf jedem Experimentrechner installierten/vorhandenen Komponenten aufgeführt. Da eine Reihe von Studierenden beim Experiment auf ihren eigenen Rechnern arbeitete, ist es möglich, dass sich weitere, im Experiment genutzte Komponenten auf diesen Rechnern befanden.

ben (siehe Anhang D). Er bestand grob gesprochen aus zwei Teilen. In Teil 1 wurde in den Ablauf des Experimentes eingeführt und (noch einmal) die Inbetriebnahme der Experimentumgebung auf dem Rechner erläutert. In Teil 2 wurde die zu bearbeitende Programmieraufgabe beschrieben. Die Teilnehmer waren dazu aufgerufen, sich Teil 2 erst dann anzusehen, wenn sie die Inbetriebnahme der Umgebung abgeschlossen haben und die Aufzeichnung von Ton, Video, Bildschirm und Mikroereignissen wie unter Abschnitt 4.1 beschrieben läuft. Die Entwickler wurden darauf hingewiesen, dass sie eine Lösung abgeben sollen, sobald sie davon ausgehen eine solche fertiggestellt und getestet zu haben, dass sie aber weiter arbeiten müssten, falls die Lösung – die dann sofort überprüft werden würde – fehlerhaft sei. Vor und nach dem Experiment wurden Fragebögen herausgegeben (Anhang D S. 502 ff. und S. 507 ff.).

In der Aufgabe stand die schon im Übungsbetrieb der Vorlesung im Zentrum stehende Webapplikation XPetStore[19], ein Shopsystem, über das Haustiere erworben werden können, im Fokus. Bei XPetStore handelt es sich um eine freie Neuimplementierung der Applikation PetStore, die von Sun Microsystems als Demonstrator für J2EE-Applikationen bereitgestellt worden war. So werden in XPetStore beispielsweise Technologien wie JSP[20], JMS[21] oder EJB 2.0[22] verwendet. XPetStore wurde aber vor allem deshalb entwickelt, um die Leistungsfähigkeit von XDoclet[23] zu demonstrieren.

Im Experiment ging es darum, eine vorher im Rahmen des Übungsbetriebs erstellte in Java geschriebene Zusatzapplikation zu XPetStore zu modifizieren. Die Applikation, die bisher den Email-Versende-Mechanismus von XPetStore nutzte, um selbst Emails abzusetzen, sollte nun keine eigenen Emails mehr versenden, sondern stattdessen mitprotokollieren, falls über XPetStore eine Email, zum Beispiel eine Bestätigungsmail für die Bestellung eines Haustieres, versendet wird. Hierzu musste in erster Linie mit Funktionalitäten des JMS gearbeitet werden. Es wurde davon ausgegangen, dass die Studierenden einigermaßen sicher im Umgang mit JMS sind. Schließlich mussten schon bei der Implementierung der ursprünglichen Version der externen Applikation JMS-Funktionalitäten, wenn auch nicht die selben, verwendet werden. Die Aufgabe war darüber hinaus derart konzipiert, dass nicht nur die externe Applikation sondern XPetStore selbst zu modifizieren war, um die gewünschte Funktionalität bereitstellen zu können (für Details siehe Anhang D).

Da dem Veranstalter die Lösung der Aufgabe bekannt war, wurden im Rahmen des Experimentes Sitzungen aufgezeichnet, die das Potential hatten, gut nachvollziehbar und beurteilbar zu sein.

Schon zu Beginn des Experiments stellte sich heraus, dass einige der Probanden ihre Rechner nicht wie gefordert vollständig vorbereitet oder getestet hatten – insbesondere in Hinsicht auf die (verbleibende) Performance. In Folge dessen hatten sowohl einige Soloprogrammierer wie auch einige Paare unterschiedliche Probleme mit ihrem Setup, was die Konzentration auf die eigentliche Aufgabe und das Vorankommen negativ beeinflusste. Letztendlich konnten – zum Teil auch wegen Setup-Problemen – nur drei Paare und ein Soloprogrammierer die Aufgabe

[19] Siehe http://xpetstore.sourceforge.net/ (Abruf: 13.03.2012).

[20] JSP steht für die „JavaServer Pages Technology“ (http://www.oracle.com/technetwork/java/javaee/jsp/index.html (Abruf: 31.10.2011)), eine auf Java basierende Methode zur dynamischen Erzeugung von Webseiten.

[21] JMS ist der „Java Message Service“ (http://www.oracle.com/technetwork/java/index-jsp-142945.html (Abruf: 26.01.2011)), eine genormte Programmierschnittstelle, um nachrichtenorientierte Middleware anzusprechen.

[22] Bei EJBs (Enterprise JavaBeans) handelt es sich um standardisierte Java-Komponenten. Sie werden innerhalb eines J(2)EE-Servers verwendet. Im XPetStore kamen Komponenten gemäß der Version 2.0 der EJB-Spezifikation zum Einsatz (http://java.sun.com/products/ejb/2.0.html (Abruf: 15.03.2012))

[23] XDoclet ist ein Werkzeug zur attributgesteuerten Erzeugung von Kode (http://xdoclet.sourceforge.net/xdoclet/index.html (Abruf: 26.01.2011)).

in der zur Verfügung stehenden Zeit erfolgreich fertigstellen. Zwei dieser Paare hatten auf ihrem eigenen Rechner gearbeitet.

Im Rahmen der im vorliegenden Dokument beschriebenen Untersuchungen wurde schließlich nur eine der aufgezeichneten Paarsitzungen verwendet. Sie nimmt bei den hier erörterten Analysen eine besondere Stellung ein, da es sich um diejenige Sitzung handelt, mit der die Untersuchungen zur Basisschicht begonnen wurden. Außerdem ist sie die einzige hier verwendete Sitzung, die nicht aus einem professionellen Umfeld stammt. Sie wird im Weiteren mit ST1.1 bezeichnet. Es handelt sich um eine Sitzung, die von zwei männlichen Probanden auf einem gestellten Rechner durchgeführt wurde. Sie hatte ein Länge von 2 Stunden und 54 Minuten und wurde aus folgenden Gründen ausgewählt:

- Die Entwickler bearbeiteten die Aufgabe erfolgreich. Somit konnte ein vollständiger „Entwicklungsprozess“ beobachtet werden.
- Während der Sitzung gab es kaum nennenswerte Probleme mit der Arbeitsumgebung oder der Aufzeichnungssoftware. Es konnte also eine Sitzung analysiert werden, in der es in erster Linie um das Lösen der Aufgabe ging.[24].
- Die von den Entwicklern abgegebene Lösung war im ersten Anlauf richtig. Es konnte also eine vollständige Sitzung analysiert werden, in die zu keinem Zeitpunkt aktiv relevante Informationen von außen hineingetragen wurden.
- Die beiden Entwickler hatten schon im Vorfeld häufiger zusammen programmiert. Somit war weitgehend sichergestellt, dass nicht in erster Linie ein Paarfindungsprozess (*pair-jelling* [219]) analysiert wurde.

Detailinformationen zu den in den ausgegebenen Fragebögen gegebenen Antworten sind in Tabelle 4.3 und Tabelle 4.4 zu finden. Es ist zu erkennen, dass sich Proband 1 (*P1*) als sehr kompetenter Entwickler sieht, wenn auch mit leichten bis großen Schwächen in den für die Aufgabe relevanten Technologien. Letztendlich empfand er die Aufgabe trotzdem als leicht. Entwickler 2 (*P2*) offenbart bezüglich dieser Technologien größere Schwächen und schätzt sich insgesamt bei weitem nicht als so kompetent wie sein Partner ein. Nach dem Experiment bewertet *P1* die Mitarbeit seine Partners trotzdem als hilfreich, insbesondere in Hinsicht auf die Strukturierung des Arbeitsprozesses. Beide beurteilen die Zusammenarbeit als harmonisch. Sieht man sich die Sitzung genauer an, fällt auf, dass die Rollen zwischen den Entwicklern äußerst ungleich verteilt wurden. So nahm *P1* zu über 95% der direkt am Rechner stattfindenden Nettoarbeitszeit die Rolle des Drivers ein.

[24] Kleinere technische Probleme ereigneten sich auch in dieser Sitzung. So funktionierte der automatische Eclipse-Syntax-Check nur eingeschränkt (siehe Tabelle 4.4). Außerdem stellte sich zu Beginn der Sitzung heraus, dass die IDE Eclipse (im vorliegenden Setting) nicht zur Bearbeitung von XML-Dokumenten verwendet werden kann. Das Paar musste an dieser Stelle auf einen externen Editor zurückgreifen (siehe Abbildung C.1)

In Abbildung C.1 ist der Verlauf der Sitzung als Abfolge von ECG-Ereignisintervallen visualisiert. Um die Übersicht zu erhöhen, wird in der Abbildung mit Spuren gearbeitet, die jeweils einen ECG-Ereignistyp adressieren. Die Abbildung enthält zusätzlich eine Kommentarspur. Sie wird dazu genutzt, eine Unterteilung der Sitzung in Phasen zu erläutern. Diese Phasen lassen sich grob wie folgt zusammenfassen:

1. Die Entwickler lesen das Aufgabenblatt und erörtern erste Fragen und Strategien (ca. 7min).

2. Das Paar führt notwendige Konfigurationen am Applikationsserver JBoss durch (ca. 68min).

3. Das Paar führt notwendige Änderungen an der Applikation XPetStore durch (ca. 26min).

4. Die Entwickler beheben einen Fehler in der vorliegenden Version der Applikation: Einem Nutzer des Webshops ist es nicht möglich, sich als Kunde zu registrieren, da beim Aufruf der Registrierungsseite nur eine Fehlermeldung erscheint (ca. 31min).[25]

5. Das Paar schließt die notwendigen Änderungen an der Applikation XPetStore ab (ca. 2min).

6. Das Paar entwickelt die gefragte externe Applikation (auf Basis der im Vorfeld vom Paar entwickelten Applikation `XpetstoreEmailSpy` - ca. 40min).

[25] Die Implementierung von XPetStore ist an dieser Stelle korrekt. Allerdings verwendete das Paar nicht einen neu installierten Shop bzw. den originalen Datenbestand, sondern eine bereits vorher bei einem der beiden Studierenden im Einsatz befindliche Installation, die im Vorfeld auf den Experimentrechner übertragen wurde.

Fragen	Antworten Entwickler 1 (P1)	Antworten Entwickler 2 (P2)
Studienfach	Informatik	Informatik
Angestrebter Abschluss	Diplom	Diplom
Hochschulsemester/Fachsemester	7/7	7/7
Programmiererfahrung in Java (Jahre)	3	3
Anzahl der Programmiersprachen, in denen anwendbare Kenntnisse existieren	6	4
Selbstbenotung: Erfahrung mit der IDE Eclipse (Schulnote)	3	3
Selbsteinschätzung: Ich kann besser Software entwickeln als x% aller Programmierer weltweit.	x≈83	x≈35
Selbsteinschätzung: JMS-Kenntnisse (Schulnote)	4	5
Selbsteinschätzung: Bewertung der Kenntnisse über XPetStore (nur bezogen auf den Bereich, der im Rahmen der Vorlesung bzw. der vorlesungsbegleitenden Übungen behandelt wurde; Schulnote)	4	5
Selbsteinschätzung: Bewertung der Kenntnisse über XDoclet (Schulnote)	5	5
Welche Teilaufgaben machen mir beim Programmieren am meisten Spaß?[a]	Design, Coding, eigene Ideen umsetzen, Bibliotheken verwenden	Anforderungsanalyse, eigenen Ideen umsetzen, Träumen
Häufigkeit der Zusammenarbeit im Paar (nur auf das Programmieren und nur auf den Experimentpartner bezogen, Freitext)	Bei den Übungsaufgaben; 6 Stunden im Monat in den letzten 12 Monaten	Eine Stunde pro Woche in den letzten 2 Monaten
Selbsteinschätzung: Wie eingespielt ist die Zusammenarbeit mit dem Experimentpartner (Schulnote)?	3	2
Selbsteinschätzung: Welche Stärken habe ich im Vergleich zu meinem Partner (Mehrfachauswahl aus vorgegebenen Antwortmöglichkeiten)?	Schnelle Auffassungsgabe, Erfahrung im Softwaredesign, Erfahrung mit Softwarearchitekturen	Genauigkeit

[a] Bei dieser Frage waren Antwortmöglichkeiten vorgegeben. Es konnte aber auch eine eigene Antwort ergänzt werden. Hier gab Entwickler 2 „Träumen“ an. Leider wurde nicht nachgefragt, was er mit dieser Antwort ausdrücken wollte.

Tab. 4.3: Selbstauskünfte und -einschätzungen der Probanden in Sitzung ST1.1 (BISJ2EE). Die Angaben wurde von den Studierenden in vor dem Experiment herausgegebenen Fragebögen gemacht (siehe S. 502 ff.). Jeder Studierende erhielt einen Fragebogen. Dieser wurde ausgefüllt, ohne dass Absprachen mit dem Partner erfolgen konnten.

Fragen	Antworten Entwickler 1	Antworten Entwickler 2
Wie schwierig/leicht fanden Sie die zu lösende Aufgabe? (Auswahl aus vorgegebenen Antwortmöglichkeiten)	leicht	meinem Können/ Wissen angemessen
Beschreiben Sie kurz die größten Schwierigkeiten der Aufgabe. (Freitext)	„Technische Dinge: Eclipse Syntax-Check war defekt; 'Das Struts Problem'. Herausfinden/Erinnern, dass die jms-config nicht automatisch verteilt wird.“	„Mit Eclipse, XPetStore rumärgern. Zu Hause/auf eigenem Rechner wäre dies viel unproblematischer gewesen.“
Haben Sie die Zusammenarbeit insgesamt als hilfreich empfunden?	Ja	Ja
Welche speziellen Aspekte fanden Sie an der Zusammenarbeit hilfreich? (Freitext)	„Mich zu bremsen, wenn ich Probleme unstrukturiert lösen will.“	“Er kennt sich gut mit Eclipse aus. Ich leider nicht so.“
Welche speziellen Aspekte fanden Sie an der Zusammenarbeit gestört? (Freitext)		„Keine.“
Welche Stärken haben Sie bei der Bearbeitung der Aufgabe bei Ihrem Partner erkannt? (Mehrfachauswahl aus vorgegebenen Antwortmöglichkeiten)	Strukturiertes Vorgehen, gute Kenntnisse in J2EE, Kooperationswille	Strukturiertes Vorgehen, schnelle Auffassungsgabe
Als wie harmonisch haben Sie die Zusammenarbeit mit Ihrem Partner empfunden? (Auswahl aus vorgegebenen Antwortmöglichkeiten)	harmonisch	sehr harmonisch

Tab. 4.4: Bewertung der Sitzung ST1.1 durch die Probanden. Die Angaben wurden von den Entwicklern in einem direkt nach dem Experiment herausgegeben Fragebogen (siehe S. 507 ff.) gemacht. Die Entwickler füllten den Fragebogen getrennt voneinander aus, ohne dass Absprachen mit dem Partner erfolgen konnten.

4.2.2 Beobachtungen von professionellen Entwicklern im Feld

Im Rahmen der in den Abschnitten 6 bis 9 erläuterten Untersuchungen wurden vor allem zwei im professionellen Umfeld stattfindende Paarprogrammierungssitzungen detailliert betrachtet. Auch bei ihnen handelte es sich – wie im Experiment BISJ2EE – um Sitzungen mit einem Rechner, einer Tastatur und einer Maus. Sie werden in den folgenden beiden Unterabschnitten (siehe S. 95 und S. 99) detailliert erläutert. Im letzten Unterabschnitt (siehe S. 102) wird dann auf diejenigen im professionellen Umfeld aufgezeichneten Paarprogrammierungssitzungen eingegangen, die darüber hinaus in die hier geschilderten Analysen eingegangen sind.

PR1.1: Paarprogrammierung in einem Unternehmen im Bereich *Social Media*

Die erste in einem professionellen Umfeld aufgezeichnete Paarprogrammierungssitzung (PR1.1) fand bei einem mittelständischen Unternehmen (im Weiteren auch als Firma 1 bezeichnet) statt, welches eine eigene Webapplikation im Bereich *Social Media* entwickelte und betrieb. Das Unternehmen war zum Zeitpunkt der Aufzeichnung von starkem Wachstum geprägt und erprobte schon mehrere Monate agile Methoden, insbesondere Praktiken des Extreme Programmings. Unter anderem wurde – wenn auch bei weitem nicht durchgängig – Paarprogrammierung praktiziert. Die Aufzeichnung der Sitzung wurde vom Autor des vorliegenden Dokumentes vorbereitet und unter Mithilfe einer Kollegin[26] durchgeführt. Folgende Schritte wurden durchlaufen:

1. Im Vorfeld wurde mit dem Entwicklungsleiter des Unternehmens ein Termin vereinbart, an dem es das Unternehmen ermöglichen würde, ein freiwilliges Entwicklerpaar (bestehend aus fest angestellten Entwicklern) bei der Arbeit in einer Paarprogrammierungssitzung zu beobachten. Die Auswahl der Entwicklungsaufgabe wurde dem Unternehmen überlassen, ebenso die Programmiersprache, in der diese umzusetzen war. Es wurde lediglich vereinbart, dass es sich um einen Task handeln sollte, der ohnehin im Rahmen der aktuellen Entwicklungen anstünde und für dessen Umsetzung die IDE Eclipse eingesetzt werden könnte. Da in dem Unternehmen zu diesem Zeitpunkt in erster Linie in PHP[27] programmiert wurde, wurde durch die Entwickler, die an der Aufzeichnung teilnehmen wollten (beide männlich), eine in PHP umzusetzende Aufgabe ausgewählt.

2. Am Tag der Aufzeichnung wurde die Aufzeichnungssoftware auf einem Rechner des Unternehmens installiert. Bei der Maschine handelte es sich

[26] Laura Plonka, zu diesem Zeitpunkt Mitarbeiterin der Arbeitsgruppe Software Engineering am Institut für Informatik der Freien Universität Berlin.

[27] Bei PHP handelt sich um eine Skriptsprache, die hauptsächlich bei der Erstellung von Webanwendungen zum Einsatz kommt (http://www.php.net/ (Abruf: 04.04.2012)).

nicht um eine der beiden, auf denen die beteiligten Entwickler normalerweise arbeiteten, sondern um eine in einem gesonderten Raum stehende. Sie war mit den im Unternehmen standardmäßig verwendeten Softwarekomponenten ausgestattet (siehe auch Tabelle 4.5) und bot somit eine relativ vertraute Umgebung.

3. Nachdem die Aufzeichnungssoftware installiert worden war, wurde ein erster Fragebogen herausgegeben (siehe Anhang E S. 512), der von den Entwicklern separat zu bearbeiten war. Hierüber sollten vor allem folgende Informationen eingeholt werden:

   – Klassifizierung der zu bearbeitenden Aufgabe
   – Beschreibung der zu bearbeitenden Aufgabe
   – Erfahrung der beteiligten Entwickler mit Paarprogrammierung

4. Danach wurde die Aufzeichnungssoftware gestartet, und die Entwickler begannen mit dem Bearbeiten der Aufgabe. Es gab keinerlei zeitliche Beschränkungen. Die Sitzung dauerte 1 Stunde und 48 Minuten.

5. Nachdem die Entwickler ihre Arbeit abgeschlossen hatten, wurde jedem von ihnen ein zweiter Fragebogen ausgehändigt. Mit diesem sollte in Erfahrung gebracht werden, wie die Entwickler die Sitzung erlebt hatten und wie sie diese beurteilten (siehe Anhang E S. 513 ff.) Sie wurden darüber hinaus aufgefordert, die Sitzung in Phasen aufzuteilen.

In den Antworten zu Fragebogen 1 charakterisierten die Entwickler[28] den zu bearbeitenden Task wie folgt:

– Eine bereits bestehende Funktionalität – das an Ereignisse gekoppelte Übermitteln von Daten aus der Webanwendung (bzw. des „Back-Ends“ der Webanwendung) des Unternehmens an eine Partnerfirma – soll erweitert werden.

– Das Hauptziel der Erweiterung besteht darin, die übertragene Datenmenge zu reduzieren.[29]

– Die zu erweiternde Funktionalität ist (wie fast alle Teile der Gesamtapplikation) in PHP programmiert und verwendet JSON[30].

[28] Genau genommen beschreibt nur *P1* die Aufgabe. Sein Partner verweist lediglich auf diese Beschreibung.

[29] Im Gespräch mit seinem Partner gegen Ende der Sitzung korrigiert *P1* seine Aussage zum Hauptziel dahingehend, dass es nicht nur um eine Reduktion der übertragenen Datenmenge, sondern auch um eine Reduktion der schreibenden Datenbankzugriffe auf Seiten der Partnerfirma geht.

[30] Die JavaScript Object Notation (JSON) ist ein Format um Daten zwischen Anwendungen austauschen zu können (http://json.org/ (Abruf: 04.04.2012)).

Betriebssystem	Microsoft Windows
IDE	Eclipse
Webbrowser	Firefox
Secure Shell-, Telnet- und Rlogin-Programm	PuTTY[a]

[a] http://www.chiark.greenend.org.uk/~sgtatham/putty/ (Abruf: 09.04.2012)

Tab. 4.5: In Sitzung PR1.1 im Zentrum der Entwicklung stehende Softwarekomponenten.

Aussagen innerhalb der Sitzung zeigten darüber hinaus, dass zumindest *P1* das prinzipielle Vorgehen zur Bewältigung der Aufgabe vor Beginn der Sitzung bekannt war. Allem Anschein nach hatte er im Vorfeld sogar schon diesbezüglich am Kode gearbeitet. Außerdem wurde ersichtlich, dass die Aufgabe optionale Anforderungen enthielt bzw. nicht so detailliert vorlag, dass eindeutig festgemacht werden konnte, welche Teilaufgaben auf jeden Fall im Rahmen der Sitzung zu erledigen waren.

Aus den Antworten der Entwickler in Fragebogen 1 konnte auch abgelesen werden, dass diese in der Vergangenheit schon häufig (evtl. auch in anderen Konstellationen) Paarprogrammierung praktiziert hatten. Die Entwickler vergaben folgende Schulnoten in Hinsicht auf die „Eingespieltheit“ mit ihrem Partner:

- Entwickler 1 (*P1*): 1-
- Entwickler 2 (*P2*): 2

Die wichtigsten Antworten zu Fragebogen 2 sind in Tabelle 4.6 zusammengefasst. Es fällt auf, dass beide Entwickler die Sitzung sehr ähnlich beurteilen. So betrachten beide Wissenstransfer als wichtigen Aspekt der Sitzung. Aus den Antworten von Entwickler 1 (*P1*) wird darüber hinaus ersichtlich, dass er sich zu Beginn der Sitzung mit dem primär zu modifizierenden Skript besser auskannte als sein Partner. Schließlich vermerkt er, dass er das Skript entwickelt hatte. Aus einer Äußerung von *P2* am Anfang der Sitzung wird aber deutlich, dass auch für ihn das Skript nicht vollständig neu war. *P2* erklärt nämlich, dass er sich das Skript im Vorfeld der Sitzung angesehen hat. Beide Entwickler bewerten die Zusammenarbeit im Paar als hilfreich, wobei zu bemerken ist, dass *P1* annähernd 100% der Zeit die Rolle des Drivers einnahm.[31]

In Abbildung C.2 im Anhang ist der Verlauf der Sitzung als Abfolge von ECG-Ereignisintervallen visualisiert. Wie schon in Abbildung C.1 gesehen, wird auch hier die Übersicht durch die Verwendung von Spuren, die jeweils einen ECG-Ereignistyp adressieren, erhöht. Es ist zu erkennen, dass folgende Phasen durchlaufen wurden:

[31] Zwischenzeitlich bot *P1* seinem Partner einmal die Driver-Rolle an. Dieser lehnte aber ab.

Fragen	**Antworten Entwickler 1**	**Antworten Entwickler 2**
Bewerten Sie, inwiefern es hilfreich war, die Aufgabe im Paar zu lösen (Schulnote).	2	2
Grund?	„≈ Historisch bedingt handelt es sich um mein Skript. Es gehört aber jetzt zum Projekt meines Partners (Übergabe).“	„Im Gespräch wurden Konzepte schneller durchdacht.“
Sind Sie mit der Bearbeitung der Aufgabe schneller vorangekommen als erwartet?	wie erwartet	wie erwartet
Spielte die Vermittlung von Wissen eine Rolle?	Ja	Ja
Wer von Ihnen hat bei der Vermittlung von Wissen mehr gewinnbringende Beiträge gemacht?	ausgeglichen	ausgeglichen
Spielte die Entwicklung einer (Lösungs-)Strategie eine Rolle?	Ja	Ja
Wer von Ihnen hat bei der Entwicklung einer (Lösungs-)Strategie mehr gewinnbringende Beiträge gemacht?	ausgeglichen	überwiegend ich
Spielte „Bugfixing“ eine Rolle?	Nein	Nein
Spielte die Entwicklung einer Architektur eine Rolle?	Nein	Nein
Spielte die Entwicklung von Algorithmen eine Rolle?	Nein	Nein
Spielten Kenntnisse von APIs eine Rolle?	Ja	Ja
Wer von Ihnen hat in Hinsicht auf Kenntnisse von APIs mehr gewinnbringende Beiträge gemacht?	überwiegend mein Partner	ausgeglichen
Spielte es eine Rolle, „den entscheidenden Einfall zu haben“?	Nein	Ja
Wer von Ihnen hat in Hinsicht auf entscheidende Einfälle mehr gewinnbringende Beiträge gemacht?	-	ausgeglichen

Tab. 4.6: Bewertung der Sitzung PR1.1 durch die Probanden. Die Angaben wurden von den Entwicklern in einem direkt nach der Sitzung herausgegebenen Fragebogen (siehe S. 513 ff.) gemacht. Die Entwickler füllten den Fragebogen getrennt voneinander aus, ohne dass Absprachen mit dem Partner erfolgen konnten. Paraphrasiert wiedergegebene Freitextantworten sind mit dem einleitenden Zeichen „≈“ markiert.

1. Vorbereitungen treffen: Das Paar passte ein PHP-Skript auf die Entwicklungssituation an, konfigurierte Eclipse etc. (ca. 15 Minuten).
2. Das Paar entwickelte die gefragte neue Funktionalität (insgesamt ca. 56 Minuten).
3. Die Entwickler fügten Funktionsaufrufe in die Gesamtapplikation ein, die mit der entwickelten neuen Funktionalität zusammenhingen (ca. 23 Minuten).
4. Das Paar führte abschließende Tests durch (ca. 13 Minuten).
5. Das Paar tauschte sich kurz darüber aus, ob nun ein günstiger Zeitpunkt wäre, die Aufzeichnung zu beenden (ca. 1 Minute)

Details zum Ablauf werden im Anhang in Abschnitt C.2 erläutert.

PR2.1: Paarprogrammierung in einem Unternehmen im Bereich Geoinformationssysteme

Die zweite professionelle Paarprogrammierungssitzung (PR2.1), die im Rahmen der in den Abschnitten 6 bis 9 erläuterten Untersuchungen detailliert analysiert wurde, ist in einem mittelständischen Unternehmen aufgezeichnet worden, welches auf die Entwicklung von Geoinformationssystemen spezialisiert ist (im Weiteren auch als Firma 2 bezeichnet). Der Entwicklungsleiter dieses Unternehmens propagierte zum Zeitpunkt der Aufzeichnung agile Methoden. Beispielsweise wurden die Entwickler ermutigt, PP zu praktizieren. Vorgeschrieben war diese Praktik allerdings nicht. Die Entwickler konnten vielmehr selbst entscheiden, wann und in welchen Konstellationen sie es für angebracht erachteten, auf diese Vorgehensweise zurückzugreifen.

Insgesamt wurden in diesem Unternehmen 6 Paarprogrammierungssitzungen mit insgesamt 8 unterschiedlichen Teilnehmern aufgezeichnet. Die Paarkonstellationen waren in allen 6 Sitzungen verschieden. Die Teilnahme an den Aufzeichnungen war freiwillig. Wie in Firma 1 wurden die im Rahmen der Aufzeichnungen zu entwickelnden Funktionalitäten auch in Firma 2 von den Probanden ausgesucht. Im Vorfeld war – ebenfalls wie in Firma 1 – nur festgelegt worden, dass es sich bei den Aufgaben um solche Programmiertätigkeiten handeln sollte, die ohnehin im Rahmen der Produktentwicklung anstanden. Außerdem sollte bei der Umsetzung die IDE Eclipse zum Einsatz kommen. Es war weiterhin vereinbart worden, dass sich die Entwickler einen Tag nach der jeweiligen Sitzung noch einmal zusammensetzten, um mit der Person, die für die Datenerhebung verantwortlich war[32], über den Sitzungsverlauf zu sprechen. Die Ergebnisse dieser so genannten Reflexionen wurden in den hier geschilderten Analysen allerdings nicht verwendet.[33]

[32] Siehe Fußnote 26 auf S. 95.

[33] Die Reflexionen waren neu eingeführt worden und hatten vor allem den Zweck, den Probanden die Möglichkeit zu geben, aus ihren Sitzungen zu lernen. Somit bekamen die Firmen einen Gegenwert dafür, dass sie es ermöglichten, Paarprogrammierungssitzungen aufzuzeichnen.

Der Ablauf der Sitzungen entsprach (abgesehen von den Reflexionen) weitgehend dem in Firma 1 (bzw. Sitzung PR1.1). Auch die Fragebögen, die vor und nach der Sitzung an die Entwickler ausgegeben wurden, waren weitgehend identisch mit den in Firma 1 verwendeten. In Fragebogen 2 wurde allerdings auf die Bewertung von unterschiedlichen, in der Sitzung potentiell eine Rolle spielenden Aspekten verzichtet. Dafür wurden mehr Informationen zur Bewertung von Phasen eingeholt. Anhand der Antworten zu Fragebogen 1 lässt sich der in Sitzung PR2.1 (von zwei männlichen Entwicklern) zu bearbeitende Task wie folgt charakterisieren:

- Das Ziel der zu bearbeitenden Aufgabe bestand darin, eine zusätzliche Funktionalität für ein bereits bestehendes (in Java implementiertes) Geoinformationssystem zu entwickeln. Es sollten so genannte "virtuelle Attribute" in einer Attributtabelle (Teil der graphischen Oberfläche) visualisiert werden.

- Hierzu musste unter anderem eine Klasse mit dem Namen `TableModel` erweitert werden, die von beiden Entwicklern als komplex eingestuft wurde.

- Diese Komplexität war auch der Grund dafür, dass den Entwicklern PP als Praktik geeignet erschien.

- Beide Entwickler schätzten, dass die Umsetzung insgesamt ca. 8 Stunden in Anspruch nehmen wird. Dies bedeutete unter anderem, dass von vorn herein absehbar war, dass die Aufgabe im Rahmen der geplanten Sitzung nicht fertiggestellt wird.

Entwickler 2 betonte darüber hinaus, dass er und sein Partner sich "nicht mit *dem* Kode" auskennen. Anhand der Aufzeichnungen lässt sich allerdings nachweisen, dass *P1* schon vor der Sitzung die Aufgabe adressierende Änderungen am Kode vorgenommen hatte (zu Beginn der Sitzung erläuterte er seinem Partner die von ihm vorgenommenen Modifikationen).

Die Antworten zu Fragebogen 1 lieferten neben den gerade wiedergegebenen Aussagen zur Aufgabe auch solche über die Entwickler bzw. das Paar. Diese sind in Tabelle 4.7 zusammengefasst.

Nachdem das Paar die Bearbeitung von Fragebogen 1 abgeschlossen hatte, begann es unverzüglich mit der Entwicklungsarbeit. Diese fand unter Microsoft Windows mit der IDE Eclipse statt. Die Entwickler saßen dabei direkt am Aufzeichnungsrechner, arbeiteten aber via Remote Desktop[34] auf einem ihrer eigenen Arbeitsrechner. Nach 1 Stunde und 15 Minuten beendeten sie die Sitzung. Zu diesem Zeitpunkt war die zu entwickelnde Funktionalität – wie erwartet – nicht fertiggestellt. Die Sitzung verlief grob gesprochen wie folgt (weitere Details sind im Anhang in Abschnitt C.3 zu finden):

[34] Fernzugriff auf die graphische Benutzeroberfläche eines Rechners.

1. Das Paar begann die Sitzung damit, zu überprüfen, ob die lokalen Sourcen den aktuellen Stand im SVN und CVS wiedergeben (ca. 1 Minute).[35]

2. Danach erläuterte *P1* seinem Partner die von ihm im Vorfeld bereits durchgeführten ersten Arbeiten. In diesem Zusammenhang versuchten die Entwickler, ihre Vorstellungen in Hinsicht auf Design und Vorgehen abzustimmen. Hierbei stellte *P2* Vorarbeiten des Partners in Frage (ca. 12 Minuten).

3. Entsprechend den durchgeführten Absprachen führte das Paar dann eine Reihe von „Refactorings“ auf dem Kode durch (ca. 21 Minuten)

4. Anschließend plante das Paar kurz das weitere Vorgehen (ca. 1 Minute).

5. Nach der Planung testete das Paar die vorgenommenen Änderungen, behob einen Defekt, testete erneut und führte dann ein *Commit*[36] durch (ca. 10 Minuten).

6. Danach begann das Paar damit, neue Funktionalität zu implementieren. Nachfolgend testete es diese, nahm Fehlerbehebungen vor, testete erneut und führte abschließend wieder ein *Commit* durch (ca. 16 Minuten).

7. Nachfolgend erörterten die Entwickler das weitere Vorgehen sowohl auf inhaltlicher (Softwaredesign) wie auch strategischer Ebene (Planung des weiteren Vorgehens; insgesamt ca. 12 Minuten).

8. Da ein “Stand-up meeting“[37] anstand beendeten die Entwickler die Sitzung. Vorher erörterten sie, wann sie die Bearbeitung fortsetzen wollen (ca. 2 Minuten).

Nach der Paarprogrammierungssitzung wurde den Entwicklern der zweite Fragebogen ausgehändigt. Hier beantworteten sie die Frage danach, als wie hilfreich sie die Bearbeitung der Aufgabe im Paar empfunden haben, unterschiedlich. So vergab *P1* die Schulnote 3, sein Partner die Schulnote 1. Die Einschätzung von *P1* überrascht vor allem deshalb, weil es sein Partner war, der den Anstoß für die umfangreichen „Refactorings“ gegeben hatte, die im Rahmen der Sitzung durchgeführt wurden. Die Antwort spiegelt allerdings auch die Verteidigungshaltung wider, die von *P1* in manchen der Designdiskussionen geäußert wurde – oft unter Verwendung der Phrase „Da war ich mir noch nicht so sicher“. Interessanterweise beurteilte *P2* das Vorankommen in der Sitzung schlechter als *P1*. Denn während

[35] Bei den Softwaresystemen SVN (Subversion, http://subversion.apache.org (Abruf: 26.01.2011)) und CVS (Concurrent Versions System, http://www.nongnu.org/cvs (Abruf: 26.01.2011)) handelt es sich um Versionsverwaltungssysteme.

[36] Als *Commit* bezeichnet man das Einspielen geänderter Programmartefakte in ein Versionsverwaltungssystem wie z.B. Subversion (siehe auch Fußnote 35 auf S. 101).

[37] Bei einem „Stand-up meeting“ handelt es sich um ein regelmäßiges, zumeist tägliches kurzes Zusammentreffen des gesamten Teams. Der agile Softwareentwicklungsprozess Scrum [188] definiert Ziele und Regeln einer solchen Besprechung.

Frage	Antworten Entwickler 1	Antworten Entwickler 2
Seit wie vielen Monaten/Jahren arbeiten Sie als Entwickler?	Mehr als 20 Jahre	9 Jahre und 2 Monate
Seit wie vielen Monaten/Jahren praktizieren Sie regelmäßig (mehrere Male im Monat) Paarprogrammierung?	20 Jahre[a]	ca. 6 Jahre
Für wie eingespielt halten Sie sich mit ihrem Partner (in Schulnoten)?	4	3

[a] Bei dieser Antwort scheint es sich nicht um einen Schreibfehler zu handeln, denn der Entwickler hatte zuerst 16 Jahre notiert, diese Antwort aber nachfolgend korrigiert. Es wurde versäumt, diesbezüglich nachzufragen.

Tab. 4.7: Selbstauskünfte, die die Entwickler in Sitzung PR2.1 in Fragebogen 1 gegeben haben.

sein Partner angab, dass das Paar wie erwartet vorangekommen sei, beurteilte *P2* das Vorankommen als schlechter als erwartet. Insgesamt kann man festhalten, dass diese Antworten das leicht angespannte Verhältnis zwischen den Entwicklern in der Sitzung widerspiegeln.

Weitere Datensätze

Eine vom Autor des vorliegenden Dokumentes betreute Masterarbeit [231], in der es primär darum ging, Beeinflussungen der Aktivitäten eines Entwicklers durch seinen Partner zu untersuchen, diente auch dazu, die Einsatzfähigkeit der so genannten Basisschicht zu erproben und die Basisschicht nötigenfalls zu ergänzen. Hierbei kam eine schon sehr weit fortgeschrittene Version der Basisschicht zum Einsatz. In der Studie wurden neben den Sitzungen ST1.1, PR1.1 und PR2.1 noch vier weitere Sitzungen analysiert. Es handelte es sich um drei weitere Sitzungen aus Firma 2 (PR2.2, PR2.3, PR2.4), sowie eine Sitzung, die in einem mittelständischen Unternehmen aufgezeichnet wurde, dessen Kerngeschäft die Entwicklung und Vermarktung einer CRM-Software[38] bildet (PR3.1). Das Vorgehen bei der Aufzeichnung dieser Sitzung war identisch zum Vorgehen bei den Aufzeichnungen in Firma 2 (siehe hierzu auch [167]). Die in den Sitzungen PR2.2, PR2.3, PR2.4 und PR3.1 beobachteten Phänomenen wurden auch dazu herangezogen, die Basisschicht auszudifferenzieren (hierauf wird noch einmal in Kapitel 10 eingegangen). Im Gegensatz zu den Sitzungen ST1.1, PR1.1 und PR2.1 wurden PR2.2, PR2.3, PR2.4 und PR3.1 aber nicht vollständig mit der so genannten Basiskonzeptmenge (Teil der Basisschicht) kodiert. Die wichtigsten Charakteristika dieser Sitzungen sind Tabelle 4.2.2 zusammengefasst.

[38] Customer-Relationship-Management

Aspekt	PR2.2	PR2.3	PR2.4	PR3.1
Länge der Sitzung (hh:mm)	01:57	02:05	01:23	02:22
Geschlecht Entwickler 1	männlich	männlich	weiblich	männlich
Geschlecht Entwickler 2	männlich	weiblich	männlich	männlich
Entwickler 1: Tätig als Entwickler in einer Firma seit? (Selbstauskunft)	9 Jahre und 3 Monate	10 Jahre	10 Jahre	2,5 Jahre
Entwickler 2: Tätig als Entwickler in einer Firma seit? (Selbstauskunft)	4 Jahre und 2 Monate	2 Jahre und 8 Monate	6 Jahre	3 Monate
Entwickler 1: Regelmäßiger[a] Einsatz von PP seit? (Selbstauskunft)	6 Jahre	unleserliche Antwort	12 Jahre	keine Angabe
Entwickler 2: Regelmäßiger Einsatz von PP seit? (Selbstauskunft)	2 Jahre	6 Jahre und 7 Monate	0 Jahre	3 Monate
Entwickler 1: Eingespieltheit mit dem Partner? (Schulnote; Selbstauskunft)	3	2	4	erstes Mal
Entwickler 2: Eingespieltheit mit dem Partner? (Schulnote; Selbstauskunft)	3	keine Angabe	4	erstes Mal
Sonstiges		2 Tastaturen/ 2 Mäuse	2 Tastaturen/ 2 Mäuse	

[a] Mehrere Male im Monat.

Tab. 4.8: Charakteristika der Sitzungen PR2.2, PR2.3, PR2.4 und PR3.1

4.2.3 Theoretisches Sampling

Wie bereits erläutert fußen die in den Kapiteln 6 bis 9 erläuterten Ergebnisse zu großen Teilen auf den Sitzungen ST1.1, PR1.1 und PR2.1. Sie wurden unter Berücksichtigung der Methode des theoretischen Samplings (siehe S. 52) aus der Menge der sukzessiv erhobenen Datensätze ausgewählt. Hierbei wurden folgende Kriterien angewendet:

1. Initial wurde Sitzung ST1.1 analysiert. Dies geschah vor allem aus einem pragmatischen Grund: Die Videos aus dem Experiment BISJ2EE waren die ersten zur Verfügung stehenden, in denen die Entwickler Aufgaben bearbeiteten, die denen in der professionellen Softwareentwicklung zumindest ähnelten. Kontaktaufnahmen zu Firmen befanden sich zu diesem Zeitpunkt noch in der Anbahnungsphase. Es war also noch nicht abzusehen, wann erste Aufzeichnungen aus Sitzungen in einem professionellen Umfeld zur Verfügung stehen würden. ST1.1 im Speziellen wurde ausgewählt, weil die-

se Sitzung zu den wenigen gehörte, in denen die Entwickler die Aufgabe vollständig bearbeitet hatten. Für ST1.1 sprach darüber hinaus, dass das Paar bereits im Vorfeld in dieser Konstellation umfangreichere Erfahrungen mit Paarprogrammierung gesammelt hatte (siehe auch S. 91 f).

2. Die Sitzung PR1.1 wurde ausgewählt, weil sich nach der Analyse von Sitzung ST1.1 die Frage stellte, ob die umfangreichen Ergebnisse auf professionelle Sitzungen übertragbar sind. PR1.1 war die erste Sitzung, die für das Herangehen an eine solche Fragestellung zur Verfügung stand. Sie hatte darüber hinaus den Vorteil, dass in ihr eine andere Programmiersprache als in Sitzung PR1.1 eingesetzt wurde.

3. Nachdem beobachtet worden war, dass die Paare in Sitzung ST1.1 und PR1.1 sehr harmonisch zusammen gearbeitet hatten, stellte sich die Frage, ob sich bei der Analyse weniger harmonischer Paare neue Einsichten ergeben würden. Aus diesem Grund wurde Sitzung PR2.1 ausgewählt. Außerdem konnten in den Sitzungen ST1.1 und PR1.1 jeweils nur sehr wenige Driver-Wechsel beobachtet werden. Da in Sitzung PR2.1 die Driver-Rolle gleichmäßiger auf die beiden Probanden verteilt war, schien sie auch aus diesem Grund ein geeigneter Kandidat zu sein. Darüber hinaus schien es interessant zu sein, eine Sitzung zu beobachten, in welcher den beiden Entwicklern von Anfang an klar war, dass nicht genügend Zeit zur Verfügung stehen würde, die Aufgabe fertigzustellen.

In Kapitel 10 wird erläutert, dass sich die Resultate der Analysen von Sitzung ST1.1 im Rahmen der weiterführenden Untersuchungen zwar in vielfältiger Weise detaillieren und ergänzen ließen, aber keine fundamentalen Änderungen vorgenommen werden mussten. Dies mag daran liegen, dass im Rahmen der Analysen nur solche Sitzungen betrachtet wurden, deren primäres Ziel darin bestand, bestehenden Kode zu erweitern. Dies geschah aus der Überlegung heraus, dass es sich bei diesem Typ von Aufgabe um einen im professionellen Umfeld häufig vorkommenden handelt. Weiterführende Untersuchungen müssen ggf. im Auge behalten, inwieweit sich die hier dargestellten Ergebnisse auch auf andere Sitzungstypen (z.B. Reviews, Schreiben von komplett neuem und unabhängigem Kode etc.) übertragen lassen.

Achtung: Die Auswahl der im Rahmen der Herleitung der BS untersuchten Paarprogrammierungssitzungen zielte nicht darauf, möglichst viele der potentiell denkbaren Einsatzszenarien der PP – wie z.B. Einarbeiten neuer Entwickler oder Beheben von Defekten – für sich allein analysieren zu können. Vielmehr wurden Sitzungen herangezogen, die potentiell ein breites Spektrum von Phänomenen versprachen und darüber hinaus häufig anzutreffende Entwicklungssituationen widerspiegelten.

Kapitel

5 Erweiterungen der GTM

„Siehst du nicht auch, wie das Auge sich spannt und den Willen darauf lenkt,
Wenn es begonnen den Blick auf zarte Gebilde zu richten?
Ohn' ein solches Bemühn ist deutliches Sehen nicht möglich.
Kann man doch selbst erfahren, daß deutlich erkennbare Dinge,
Wenn sie der Geist nicht beachtet, so gut wie dem Blicke entrückt sind
Während der ganzen Zeit und in weiteste Ferne verschlagen.
Weshalb soll es nun wunderbar sein, daß dem Geiste das andre
Alles verloren geht, nur das nicht, worauf er sich einstellt?"
Lukrez, Über die Natur der Dinge

In Kapitel 3 wurde detailliert in die GTM eingeführt. Im vorliegenden Kapitel wird nun beschrieben, in welcher Weise diese Methode im Rahmen eines ersten Versuches, die Vorgänge bei der PP zu analysieren, zum Einsatz gebracht wurde, warum dieser Versuch nicht erfolgreich war und welche Schlussfolgerungen aus dem Fehlschlag gezogen wurden.

Das Kapitel beginnt mit der Erörterung der Probleme, die dazu führten, dass der erste explorative Analyseversuch abgebrochen wurde (Abschnitt 5.1). Nachfolgend werden dann Praktiken vorgestellt, die vom Autor des vorliegenden Dokumentes entwickelt wurden, um den beobachteten Schwierigkeiten bei der Analyse begegnen zu können (Abschnitt 5.2). Diese Praktiken ergänzen die GTM nach Strauss und Corbin, ohne diese im Kern zu verändern. Eine erste Version dieser Praktiken wurde erstmals 2007 auf dem Workshop der *Psychology of Programming Interest Group* (*PPIG*) präsentiert [180] und nachfolgend in der Fachzeitschrift Human Technology veröffentlicht [181]. Die Darstellungen im vorliegenden Kapitel folgen in weiten Teilen den Darstellungen in diesen Veröffentlichungen, ohne dass dies jeweils explizit herausgestellt wird. Achtung: Die Praktik 3 „Modell der Analysemethoden und -objekte" wird hier in einer stark erweiterten Version vorgestellt.

5.1 Probleme bei der Anwendung der GTM

Der erste Versuch, den Prozess der Paarprogrammierung qualitativ und in explorativer Weise zu untersuchen, begann mit der Festlegung sowohl eines kurzwie auch eines mittelfristigen Untersuchungsziels bzw. der Formulierung von Fragen, denen diesbezüglich nachgegangen werden sollte. Beide Ziele waren darauf ausgerichtet ein Verständnis des Paarprogrammierungsprozesses zu erlangen und wurden – um den explorativen Ansatz der geplanten Untersuchung nicht zu gefährden – nur grob formuliert:

- **Kurzfristiges Analyseziel (Charakterisierung der Vorgänge bei der Paarprogrammierung):** Welche Geschehnisse formen eine Paarprogrammierungssitzung?
- **Mittelfristiges Analyseziel (Identifizierung von sich wiederholenden (Verhaltens-)Mustern):** Welche sich wiederholenden (Verhaltens-) Muster gibt es und wie lassen sich diese klassifizieren – z.B. nach Kriterien wie hilfreich, behindernd oder neutral?

Es wurde davon ausgegangen, dass die Unterscheidung zwischen kurz- und mittelfristig in gewisser Weise artifiziell und zudem vage ist. Durch sie sollte nur deutlich gemacht werden, dass Muster aller Wahrscheinlichkeit nach nur dann entdeckt werden können, wenn zumindest annähernd klar ist, woraus sich der Prozess der PP (im Wesentlichen) zusammensetzt.

Nach der Festlegung dieser Ziele wurde unmittelbar mit dem offenen Kodieren von Videomaterial begonnen. Initial wurde hierbei eine Sitzung betrachtet, in welcher zwei mit PP vertraute Studierende[1] beim Entwickeln eines JSP-Tags[2] beobachtet werden konnten (zu diesem Zeitpunkt standen die Daten aus dem Experiment BISJ2EE noch nicht zur Verfügung – die verwendete Aufzeichnungsmethodik war aber weitgehend mit der später im BISJ2EE-Experiment verwendeten identisch). Hierbei wurde – wie bereits angedeutet – der GTM nach Strauss und Corbin gefolgt. Auf eine Transkription der Aufzeichnung wurde verzichtet, da eine solche – zumindest im ersten Stadium der Untersuchungen – weder praktikabel noch vernünftig erschien. Zu umfangreich war die Menge der durch das Video zur Verfügung gestellten potentiell relevanten oder nützlichen Informationen, als dass eine Filterung der Daten, wie sie sich zwangsläufig aus einer Transkription ergeben hätte, hinnehmbar gewesen wäre. Es wurde entschieden, dass die Analysen direkt auf dem Videomaterial durchgeführt werden sollten. Hierzu wurde auf die QDA-Software[3] ATLAS.ti[4] zurückgegriffen. ATLAS.ti erlaubt unter anderem die persistente Markierung von Videoausschnitten (zusammenhängende Intervalle – in ATLAS.ti als *Quotations* bezeichnet). An diese Ausschnitte, die zur Kennzeichnung von Phänomenen verwendet werden können, lassen sich mittels ATLAS.ti beliebige Kodes oder Memos annotieren. Auf alle zu einem Kode (oder Memo) gehörenden Videoausschnitte kann dann – z.B. im Rahmen des ständigen Vergleichens (siehe S. 54 f) – direkt zugegriffen werden. ATLAS.ti kann also als Werkzeug für die GTM-Analyse von Videomaterial eingesetzt werden.

[1] Bei den Probanden handelte es sich um Teilnehmer einer Hauptstudiumsveranstaltung. Sie hatten sich freiwillig bereit erklärt, an einem kleinen Experiment teilzunehmen, in dem sie sowohl eine algorithmische wie auch eine auf die Verwendung einer Bibliothek zentrierte Aufgabe zu bearbeiten hatten.

[2] Die JSP-Technologie (siehe Fußnote 20 auf Seite 90) erlaubt es, JSP-Seiten parametrisierbare Funktionalitäten über so genannte Tags zur Verfügung zu stellen.

[3] Qualitative Data Analysis

[4] Der Großteil der im vorliegenden Dokument beschriebenen Untersuchungen wurde mit ATLAS.ti Version 5 (http://www.atlasti.com (Abruf: 21.04.2012)) durchgeführt. Darüber hinaus kamen auch Version 6 und Version 7 zum Einsatz.

Aus dem offenen Kodieren des Videos resultierten 194 verschiedene Konzepte sowie die Einsicht, in den Daten „verloren gegangen zu sein“. Insbesondere wurden folgende, mit einander zusammenhängende Probleme identifiziert:

- **Beobachtungsfokus:** Zu Beginn der Untersuchung standen keinerlei Kriterien zur Verfügung, anhand derer es möglich gewesen wäre, zu entscheiden, welche Formen von Phänomenen/Beobachtungen (verbale Äußerungen, Gesichtsausdrücke, Gesten, Körperhaltungen, Blickrichtungen, Computereingaben, Eingabemethoden etc.) zu kodieren oder – zumindest vorläufig oder partiell – zu ignorieren sind. Diese Situation änderte sich auch im weiteren Verlauf nicht. Über den gesamten Untersuchungszeitraum hinweg wurde somit versucht, möglichst alle Phänomentypen im Auge zu behalten. Dies führte zu einem Effekt, der in der oben erwähnten Veröffentlichung in der Fachzeitschrift Human Technology [181] als „overwhelmed by the data“ bezeichnet wird. Denn auch wenn es einem Beobachter möglich ist, „mehr als einer Sache auf einmal Aufmerksamkeit zu schenken“, sind dieser „Vielseitigkeit Grenzen gesetzt“ [38, S. 35 f]. Diese Grenzen wurden im Laufe der Analyse erreicht bzw. überschritten.

- **Granularität:** Da es keine Richtlinien gab, über die sich das Detailniveau bei der Kodierung steuern ließ, wurden Kodes auf ganz unterschiedlichen Niveaus generiert – z.B. gröbere wie *handle problem* und feinere wie *test defect fix*. Dies hatte zur Folge, dass es zu Abgrenzungsproblemen zwischen Kodes kam.

- **Subjektivität:** Kodes können sowohl aus Beobachtungen (genau genommen Deutungen von Beobachtungen) extrahiert werden, denen jeder Beobachter mehr oder weniger zustimmen würde, wie auch aus solchen, die von den meisten eher als vage Vermutung oder gar „Wunschdenken“ bezeichnet werden würden. Anders ausgedrückt: Die GTM stellt keine Kriterien für das zur Verfügung, was unter dem „in Daten gegründet sein“ im Einzelnen genau verstanden werden soll – zumindest keine einfachen, in einem frühen Stadium der Analyse anwendbaren. Dies hatte zur Folge, dass zwei unterschiedliche Formen von Kodes entstanden – auf der einen Seite „objektiv-deskriptive“, wie z.B. *uses documentation*, auf der anderen „subjektiv-bewertende“ wie z.B. *gains knowledge of detail*. Im Verlauf der Untersuchungen nahm die Verunsicherung darüber zu, ob und in welcher Weise hier steuernd eingegriffen werden sollte. Achtung: Hier soll nicht kritisiert werden, dass „die Forschenden [in der GTM] nie allein neutraler Beobachter, sondern zwangsläufig als Interpret ihrer Daten und als Entscheider über den konkreten Gang der theoretischen Argumentation immer

auch Subjekt des Forschungsprozesses“ sind [201, S. 16].[5] Es wird vielmehr nur darauf hingewiesen, dass es nicht immer einfach ist, mit der eigenen Subjektivität angemessen umzugehen.

- **Breites Spektrum von Phänomenen:** Die generierten Kodes deckten eine große Anzahl unterschiedlichster Interessenfelder und Themen ab. Dies hatte zur Folge, dass es nicht gelang, in zumindest einem der angeschnittenen inhaltlichen Komplexe eine gewisse Dichte zu erreichen. Dies wäre in diesem Stadium der Untersuchung sicherlich akzeptabel, vielleicht sogar erwartbar gewesen. Allerdings stellte sich auch keine Einsicht dahingehend ein, welche Aspekte (zumindest vorläufig) von besonderem Interesse sein könnten. Als Grundlage für eine Spezialisierung oder ein theoretisches Sampling schien die resultierende Menge an Kodes somit nicht geeignet zu sein.

- **Gruppierung von Konzepten:** Die große Menge von unterschiedlichen Interessenfeldern und Themen, die im Lauf der Untersuchung adressiert wurde, machte es schwierig zu Kategoriebildungen zu kommen.

- **Priorisierung/Auswahl von Phänomenen:** Gegen Ende der Untersuchung stellte sich die Frage, ob die stetig wachsende Zahl unterschiedlicher Interessenfelder und Themen, die im Rahmen der Analyse im Auge behalten werden mussten, dazu geführt haben könnte, dass zu Gunsten letztendlich wenig interessanter oder unwichtiger Aspekte wichtige übersehen wurden.

Die Einsicht in die Problematiken führte dazu, dass der erste Analyseversuch abgebrochen wurde. Stattdessen wurde damit begonnen, ergänzende Praktiken zu entwickeln, mit deren Hilfe sich zumindest einige der Schwierigkeiten abmildern lassen.

5.2 Die GTM ergänzende Praktiken

In den nachfolgenden Unterabschnitten werden Praktiken erläutert, die vom Autor des vorliegenden Dokumentes entwickelt wurden, um den im letzten Abschnitt dargestellten Problemen bei der Analyse von Videoaufzeichnungen von Paarprogrammierungssitzungen begegnen zu können.

[5] Mühlmeyer-Mentzel und Schürmann [141] weisen explizit darauf hin, dass „Lehrbücher zur GTM [...] Verfahrenstechniken [vermitteln], die beispielsweise aufzeigen, wie man von der Deskription der Daten hin zu deren analytischer Beschreibung kommt oder zu einer dichten im Gegensatz zu einer oberflächlichen Theorie. Die Ausführung dieser verfahrenstechnischen Anteile führt aber nicht notwendigerweise zu guten Ergebnissen. Jedem verfahrenstechnischen Schritt gehen inhaltliche Überlegungen voraus, die in *Entscheidungen* münden müssen, um einen verfahrenstechnischen Schritt durchführen zu können. [...] Ob ein verfahrenstechnischer Schritt zu einem qualitätsvollen Ergebnis führt, liegt [...] an der Kompetenz des Forschenden und nicht allein am Verfahrensschritt.“

5.2.1 Praktik 1: Blickwinkel auf die Daten

Wie in Abschnitt 3.1.4 ausgeführt, gehen sowohl Glaser und Strauss [77] wie auch Strauss und Corbin [199] davon aus, dass der Forscher durch eine Forschungsfrage geleitet wird. Diese sollte „zu Beginn des Forschungsprojektes sehr offen formuliert“ sein [209], wobei Strauss und Corbin [199, S. 23] betonen, dass sie weder „so offen [sein sollte], daß sie das ganze Universum von Möglichkeiten einbezieht“ noch „so eingegrenzt und fokussiert, daß Entdeckungen und neue Erkenntnisse ausgeschlossen werden.“ Die „Fragestellung in einer Untersuchung mit der Grounded Theory“ sollte „eine Festlegung [sein], die das Phänomen bestimmt, welches untersucht werden soll.“ Dieses Verständnis wird durch Praktik 1 „Blickwinkel auf die Daten“ präzisiert und operationalisiert, indem empfohlen wird, die folgenden drei Fragen zu klären:

1. **Ausrichtung des Interesses:** In welcher Hinsicht sollen die Daten Einblicke verschaffen?

2. **Art der zu berücksichtigenden Phänomene:** Welche Formen von Phänomenen sollen als identifizierbar und die Analyse mitbestimmend zugelassen werden?

3. **Ergebnistyp:** Welche Art von Analyseergebnis wird angestrebt?

Im Rahmen der ersten Frage soll – wie schon von Strauss und Corbin gefordert – sowohl geklärt werden, „was man schwerpunktmäßig untersuchen“ will wie auch “was man über den Gegenstand wissen möchte“ [199, S. 23]. Hierbei geht es wohlgemerkt nicht darum, festzulegen was, sondern nur in welcher Hinsicht etwas entdeckt werden soll. Der erste Teil des Blickwinkels fungiert also als Kriterium, das dabei hilft, zu entscheiden, welchen Ereignissen potentiell mehr und welchen potentiell weniger Beachtung geschenkt werden sollte.

Mit der zweiten Fragestellung soll der Umgang mit der eigenen Subjektivität erleichtert werden, indem vorgeschlagen wird, die Formen und das Ausmaß der in die Analyse einfließenden subjektiven Einschätzungen systematisch zu begrenzen. So kann beispielsweise festgelegt werden, dass nur im behavioristischen Sinne [213] beobachtbare Phänomene berücksichtigt und konzeptualisiert werden dürfen. Weniger strenge Festlegungen können darüber hinaus den Umgang mit Konzepten erlauben, die auch bestimmte nicht direkt beobachtbare Vorgänge (z.B. innere Einstellungen oder Denkprozesse der Probanden), Vorhersagen (z.B. „hilfreich, um Ziel X zu erreichen“) oder Bewertungen (z.B: „gut“ oder „schlecht“) adressieren.

Genauso wie die zweite dient auch die dritte Frage dazu, den Blick auf die Daten fokussieren zu können – wenn auch in gänzlich anderer Weise. Denn während mittels des zweiten Teilaspektes Phänomene in gewisser Weise anhand ihrer Evidenz gefiltert werden, geschieht dies beim dritten Teilaspekt anhand ihrer Nützlichkeit für das Erreichen einer bestimmten Ergebnisform. So sind für die Herleitung eines allgemeinen Kodierschemas in einem breiteren Interessenfeld andere –

evtl. sogar weniger oder weniger detaillierte – Phänomene zu berücksichtigen als für das Erstellen einer vollständigen Theorie in einem spezifischen Teilbereich. Achtung: Auch wenn das Festlegen eines Ergebnistyps Auswirkungen darauf haben kann, welche Kodierformen der GTM nach Strauss und Corbin zum Einsatz kommen werden, ist das Festlegen auf einen Ergebnistyp nicht gleichzusetzen mit einer Beschränkung der GTM auf bestimmte Praktiken. Denn selbst wenn beispielsweise entschieden wird, dass „nur“ ein Kodierschema herzuleiten ist, muss dies nicht bedeuten, dass eine Beschränkung auf das offene Kodieren sinnvoll ist. So kann beispielsweise auch der (partielle) Einsatz des axialen Kodierens von Nutzen sein, z.B. um das Material besser zu verstehen.

Der Blickwinkel steckt also nicht nur den inhaltlichen Rahmen ab, sondern definiert auch einen Filter auf den Ereignissen – z.B. um zu entscheiden, welche Teile eines (reichhaltigen Datensatzes) man sich zuerst ansehen möchte. Ein initial festgelegter Blickwinkel sollte deswegen nicht als unveränderbar angesehen werden. Er muss vielmehr im Laufe der Untersuchung regelmäßig in Hinsicht auf seine Brauchbarkeit überprüft und nötigenfalls modifiziert werden. Insbesondere die im Rahmen von Frage zwei getroffenen Entscheidungen sollten nicht als strikte Regel, sondern als grobe Orientierungshilfe betrachtet werden, die je nach Erkenntnisstand zu ändern ist. Insbesondere dann wenn sich herausstellt, dass die auferlegte Beschränkung das Vorankommen behindert, muss über eine Modifikation nachgedacht werden.

Der Blickwinkel, mit dem die weitere Analyse der Paarprogrammierung begonnen wurde, wird in Abschnitt 6.1 erläutert.

5.2.2 Praktik 2: Syntaktische Regeln für Konzeptnamen

Die GTM macht keinerlei Vorgaben oder Anmerkungen dahingehend, wie Konzepte zu benennen sind. Dieser Freiheitsgrad führte im Rahmen des ersten Analyseversuchs dazu, dass unterschiedlichste Formen von Konzeptnamen verwendet wurden. Diese Diversität machte es im Laufe der Untersuchung zunehmend schwierig, sich bei einzelnen Kodierungen an alle potentiell geeigneten Konzepte zu erinnern oder Konzepte bzw. Gruppen von Konzepten miteinander zu vergleichen. Hieraus wurde der Schluss gezogen, dass es – zumindest in der ersten Phase der Analyse – hilfreich sein kann, mit einem Namensschema zu arbeiten. So wurde auf Basis der im ersten Versuch gesammelten Erfahrungen und dem daraus resultierenden Blickwinkel (siehe Abschnitt 6.1) für die im Weiteren beschriebenen Untersuchungen folgendes Schema festgelegt:

```
code = <actor>.<description>
actor = P1 | P2 | P
description = <verb>_<object>[_<criterion>]
```

`P1` und `P2` stehen für Entwickler bzw. *Programmierer* 1 und 2, wobei die Bezeichner pro Sitzung fest zuzuweisen sind. Mit dem Bezeichner `P` wird das

Paar als Gesamtes referenziert.[6] Mit einer `description` wird sowohl beschrieben, welcher Aktivität eine Person nachgeht (Verbbestandteil) wie auch worauf sich diese Aktivität bezieht (Objektbestandteil). Beispielsweise referenziert das Konzept *P1.ask_knowledge* Äußerungen, in denen *P1* seinen Partner nach Wissen fragt und *P2.write_sth*[7] Schreibhandlungen von *P2*. Der Zusatz `criterion` kann dazu verwendet werden, Konzepte weiter zu spezifizieren.[8]

Das hier gewählte Schema entstand nicht losgelöst von anderen Aspekten der Untersuchung, sondern baut auf dem zu Beginn der Analyse festgelegten Blickwinkel auf (siehe Abschnitt 6.1), indem es berücksichtigt, dass in erster Linie (individuelle) Aktivitäten zu eruieren sind und eine Ausdifferenzierung von Konzepten mittels Eigenschaften nicht angestrebt wird. Bei der Anwendung des Schemas konnten folgenden Vorteile beobachtet werden:

- Phänomene bzw. Konzepte wurden schon zum Zeitpunkt ihrer Einführung besser verstanden, da das Schema das strukturierte Nachdenken über Phänomene und Konzepte erleichtert.

- Auch mit größeren Konzeptmengen konnte konsistent umgegangen werden, da die Struktur der Namen als „Wegweiser“ durch die Menge fungierte.

- Eine Reihe von Beziehungen zwischen Konzepten trat zügig hervor.

- Die Anwendung der Kernpraktik des ständigen Vergleichens wurde erleichtert, da durch die Konzeptnamen (zumindest zwei) unterschiedliche Aspekte der einzelnen Konzepte sofort sichtbar waren und auf ihre Angemessenheit im betrachteten Fall überprüft werden konnten.[9]

- Ursachen für Abgrenzungsprobleme zwischen Phänomenen bzw. Konzepten konnten leichter identifiziert werden.

- Grundsätzliche Schwächen der gewählten Konzeptualisierung konnten einfacher ausgemacht und korrigiert werden.

[6] Im Rahmen der im vorliegenden Dokument vorgestellten Konzepte wird der Bezeichner `P` nicht verwendet. Aktivitäten werden also nie als solche des Paars, sondern immer nur als solche von Einzelpersonen angesehen.

[7] In Kapitel 8 wird erläutert, warum hier ein unspezifisches Objekt wie *sth* (*something*) zum Einsatz kommt.

[8] Bei den im vorliegenden Dokument vorgestellten Konzepten kommt die Ergänzung `criterion` nicht (mehr) zum Einsatz. Sie wurde im Rahmen der hier beschriebenen Untersuchungen nur temporär verwendet – z.B. um kurzfristig unterschiedliche Formen von Strategievorschlägen unterscheiden zu können (siehe S. 217).

[9] Genau genommen geht es hierbei nicht ausschließlich um die Überprüfung auf Angemessenheit. Schließlich ist das Verständnis über das, was unter einem bestimmten Verb oder Objekt bzw. der Kombination von einem Verb mit einem Objekt verstanden werden soll, ein evolutionärer Prozess.

Viele dieser Vorteile lassen sich z.B. – zumindest für die hier gewählte Syntax – durch folgende Überlegung plausibilisieren: Wenn im Laufe des offenen Kodierens allem Anschein nach ein neues Konzept entdeckt wird (aus den Daten „emergiert“), bieten sich zwei Möglichkeiten, um dieses zu benennen:

1. Entweder kann der Name des Konzeptes durch eine noch nicht verwendete Kombination von Elementen der bereits festgelegten Verb- und Objektmenge gebildet werden, oder

2. es muss mindestens ein neues Verb oder ein neues Objekt identifiziert werden, um dieses für den Namen des neuen Konzeptes zu verwenden.

Im ersten Fall wird das Verständnis des verwendeten Verbs und des verwendeten Objekts in Hinsicht auf den gewählten Blickwinkel verbessert, weil beide Teile in einem im Allgemeinen neuen, bisher nicht berücksichtigten Kontext betrachtet werden (gilt auch für den ggf. wiederverwendeten Bestandteil im zweiten Fall).

Im zweiten Fall muss genau überlegt werden, warum und wie sich das neue Verb bzw. das neue Objekt von den bestehenden unterscheidet. Hierdurch werden einerseits neue Konzepte (bzw. Verben und Objekte) von vornherein besser verstanden und andererseits bestehende Konzepte (bzw. ihre Bestandteile) regelmäßig bez. neuer Aspekte beleuchtet.

Auswirkungen auf das Ergebnis der Untersuchungen, die sich unter anderem auch aus der Wahl dieses Schemas ergeben haben, werden im weiteren Verlauf z.B. in den Abschnitten 9.1 und 9.2 diskutiert.

5.2.3 Praktik 3: Modell der Analysemethoden und -objekte

Im Rahmen des ersten Analyseversuchs tauchten auch solche Probleme auf, die unmittelbar mit Eigenschaften der Analysemethode und des Analysewerkzeugs zusammenhingen:

- Unterschiedliche Begriffsbildungen in der GTM und in der verwendeten Analysesoftware ATLAS.ti sorgten für Irritationen (siehe Tabelle 5.1) – ein Phänomen, das auch von Mühlmeyer-Mentzel und Schürmann im Rahmen ihrer Kurse zur GTM beobachtet wurde [141].

- Der dem GTM-Prozess inhärente fließende Übergang zwischen Phänomenwelt („reale Welt“) und konzeptueller Welt erschwerte in vielen Situationen sowohl die Standortbestimmung (z.B.: Worüber rede ich/reden wir gerade?) wie auch die Orientierung (z.B.: Welche Form von Erkenntnis strebe ich/streben wir gerade in welcher Weise an?).

- Es wurde eine Leitlinie vermisst, mit der Ideen aus der GTM systematischer zur Lösung von Problemen, z.B. bei der Einführung neuer Kodes oder bei der Abgrenzung von Kodes und/oder Eigenschaften zueinander, eingesetzt werden können.

Um diesen Problemen begegnen zu können und Optionen im Analyseprozess sowie die Beziehungen der dabei zu verwendenden Objekte zu verdeutlichen, wurde das *Modell der Analysemethoden und -objekte* entwickelt. Es bezieht alle Kodierphasen der GTM nach Stauss und Corbin [199] ein und ist in Abbildung 5.1 in Form eines UML-Klassendiagramms[10] dargestellt.[11] Für die einzelnen dort aufgeführten Klassen und Assoziationen gilt das Folgende:

- **`quotation/phenomenon`**: Bei der Analyse von Videos mittels ATLAS.ti können beobachtete Phänomene (`phenomena`) unter Zuhilfenahme von so genannten `quotations` markiert werden. Bei `quotations` handelt es sich um zusammenhängende Ausschnitte aus einem Video, die einen Start- und einen Endzeitpunkt besitzen.

- **`code/conceptualisation/annotation`:** Zur Konzeptualisierung (`conceptualisation`) von Phänomenen stehen in ATLAS.ti Objekte vom Typ `code` zur Verfügung. Hierbei wird die Konzeptualisierung, sprich das Heften eines `codes` an eine `quotation`, als `annotation` bezeichnet. ATLAS.ti stellt eine entsprechende Operation zur Verfügung. Objekte vom Typ `annotation` existieren in ATLAS.ti hingegen nicht – zumindest nicht über die Benutzerschnittstelle zugreifbare. Ein Videoausschnitt kann mit einem (oder auch mehreren) Objekten der Klasse `code` annotiert werden. Über einen `code` kann auf jede zugehörige, dass heißt mit diesem Konzept annotierte `quotation`, zugegriffen werden – und zwar derart, dass der entsprechende Videoausschnitt unmittelbar abgespielt wird. ATLAS.ti unterstützt somit direkt das ständige Vergleichen.

- **`property/value`:** Eigenschaften (`properties`) und ihre Werte (`values`) müssen in ATLAS.ti als Objekte vom Typ `code` modelliert werden. Eigene Objekte für Eigenschaften oder deren Werte gibt es in ATLAS.ti nicht.[12] Achtung: Obwohl durch die Verwendung von `values` – wie in der Abbildung dargestellt – `conceptualisations` näher beschrieben werden sollen, können in ATLAS.ti `values` nur an `quotations` und nicht an `conceptualisations` „gehängt“ werden.

- **`category`:** Das Gleiche gilt im Prinzip auch für Kategorien. Auch sie müssen in ATLAS.ti als `codes` modelliert werden.[13]

[10] Die Unified Modeling Language (UML) ist eine von der Object Management Group (http://www.omg.org (Abruf: 29.07.2012)) entwickelte Modellierungssprache.

[11] Bei den in Abbildung 5.1 dargestellten Klassen sind aus Platzgründen nicht alle im Rahmen einer GTM-Analyse notwendigen Eigenschaften angegeben. So sollten viele der Objekte, z.B. `concepts`, eine Beschreibung besitzen, die im Rahmen einer GTM-Analyse auszuarbeiten ist.

[12] In ATLAS.ti können allerdings Relationen zwischen `codes` definiert werden. Beispielsweise kann die Beziehung zwischen einem `code` und einer Eigenschaft des `codes` über eine Relation `is property of` festgehalten werden.

[13] ATLAS.ti bietet die Möglichkeiten, `codes` zu gruppieren. Auf eine Erörterung dieser Funktionalität in Bezug auf den Umgang mit Kategorien wird hier verzichtet, da sie im Rahmen der hier geschilderten Untersuchungen nicht derart verwendet wurde.

– **`conceptualisationRelation/conceptRelation`:** Strauss und Corbin weisen (z.B. im Rahmen von Erläuterungen zum axialen Kodieren) darauf hin, dass es notwendig ist, sich sowohl Beziehungen zwischen konzeptualisierten Phänomenen, im Weiteren als `conceptualisationRelations` bezeichnet, wie auch solche zwischen Konzepten, im Weiteren als `conceptRelations` bezeichnet[14], anzusehen – am Besten in einer Art Wechselspiel. Sie schlagen hier die Verwendung des paradigmatischen Modells vor (siehe Abbildung 3.3). Durch `conceptualisationRelations` ist es möglich, das Zusammenspiel von Phänomenen, also Zusammenhänge in der realen Welt im Blick zu behalten, mit `conceptRelation`, Hypothesen aufzustellen bzw. die mittels `conceptualisationRelations` verifizierten Zusammenhänge in die im Entstehen begriffene Theorie einzubetten. In ATLAS.ti können beide Formen von Beziehungen über frei definierbare Relationen festgehalten werden, was unter anderem auch bedeutet, dass mit `conceptRelations` über das hinausgegangen werden kann, was im paradigmatischen Modell adressiert wird.

– **`track`:** Neben den sich direkt aus der GTM ergebenden Klassen wurde eine weitere definiert, der so genannte `track`. Objekte der Klasse `track` sollen dabei helfen, sich einen Überblick, z.B. über Verläufe, zu verschaffen. Hierzu „sammelt“ man in einem `track` auf ein Thema fokussierte `conceptualisations` – und zwar solche, die sich anordnen lassen, z.B. entlang einer Zeitachse. Im Anschluss visualisiert man den `track` (evtl. zusammen mit anderen). Im Rahmen der hier geschilderten Untersuchungen wurde zur Visualisierung von `tracks` die selbst entwickelte Software *stripes* (siehe S. 85) eingesetzt. Hierbei wurden entlang interessierender Aspekte (z.B. eine Gegenüberstellung der Anteile der Äußerungen von Entwickler 1 und Entwickler 2) mittels *Code Families* in Spuren gruppierte Videoannotationen als XML aus ATLAS.ti exportiert, in CSV-Daten konvertiert und mit *stripes* dargestellt (siehe Abbildung 5.2).

[14] Die Termini `conceptualisationRelation` und `conceptRelation` wurden weder der GTM noch ATLAS.ti entliehen – auch wenn derartige Relationen sowohl in der GTM wie in ATLAS.ti adressiert werden, sondern im Rahmen der ersten Analysen von Paarprogrammierungssitzungen eingeführt.

GTM	**ATLAS.ti**
Phänomen (*Phenomenon*)	*Quotation*[a]
Konzeptualisierung (*Conceptualization*)	*Annotation*
Konzept (*Concept*)	*Code*
Eigenschaft (*Property*)	*Code*
Kategorie (*Category*)	*Code*[b]
Beziehung (*Relationship*)	*Relationship/Relation*

[a] Genau genommen ist eine *Quotation* nur ein Datenausschnitt. Allerdings bilden Datenausschnitte in ATLAS.ti die einzige Möglichkeit, um Phänomene zu markieren. Dies bedeutet unter anderem, dass es, falls in einem Zeitintervall zwei unterschiedliche Phänomene ausgemacht werden, streng genommen angebracht ist, zwei identische *Quotations* anzulegen und dann entsprechend zu annotieren. Genau dies ist aber in ATLAS.ti nicht möglich, da sich zwei Datenausschnitte in Videos zumindest um einen Frame unterscheiden müssen.

[b] In eingeschränkter Weise können Kategorien in ATLAS.ti auch durch so genannte *Code Families* realisiert werden (siehe http://www.atlasti.com (Abruf: 21.04.2012)).

Tab. 5.1: Gegenüberstellung der in der GTM und der Analysesoftware ATLAS.ti (Version 5) verwendeten Begriffe. Nachfolgend wird in Abbildung 5.1 beschrieben, wie mit den Differenzen umgegangen wurde. Achtung: ATLAS.ti ist nicht speziell auf die GTM zugeschnitten. Die durch die ATLAS.ti-Terminologie referenzierten Objekte sind vielmehr derart generisch, dass sie im Rahmen unterschiedlichster qualitativer Analysemethoden verwendet werden können. Auf spezielle Objekte von ATLAS.ti wie z.B. *Super Codes* wird hier nicht eingegangen.

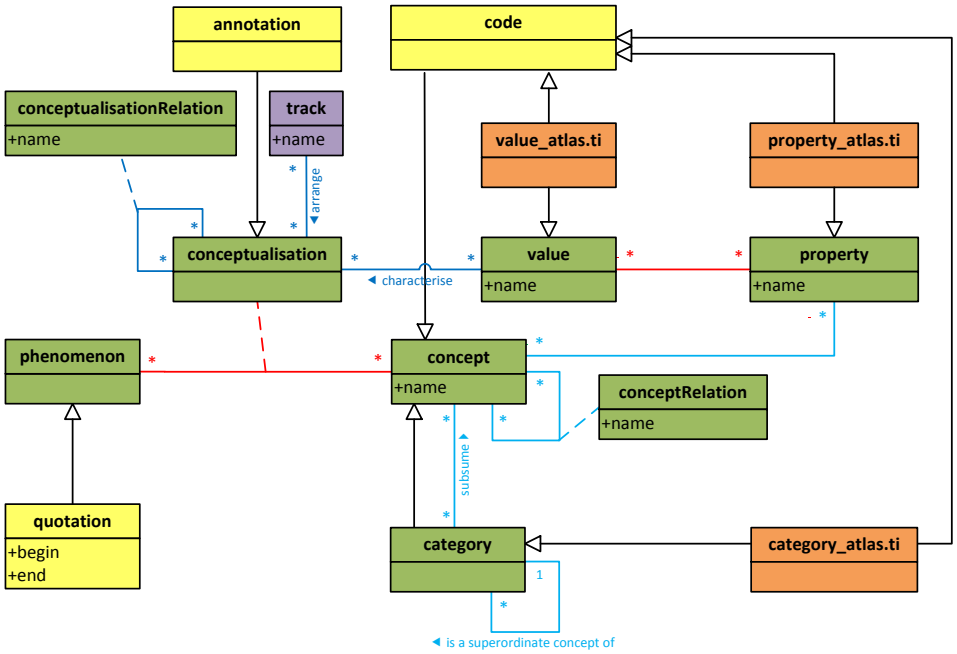


Abb. 5.1: Modell der Analysemethoden und -objekte in Form eines UML-Klassenmodells. Es fußt auf der GTM nach Strauss und Corbin [199]. Der Begriff Klasse wir hier im Sinne der objektorientierten Analyse verwendet. Die einzelnen Klassen sind auf S. 113 näher erläutert. In der Abbildung sind die grundlegenden Bestandteile der GTM-Analyse mit Grün markiert – mit Gelb Realisierungen dieser Bestandteile in ATLAS.ti. Orange sind solche Klassen eingefärbt, die zwar im Rahmen einer GTM-Analyse mit ATLAS.ti benötigt werden, für die es aber keine nativen Entsprechungen in ATLAS.ti gibt. Sie müssen unter Zuhilfenahme anderer Klassen realisiert werden – z.B. müssen Eigenschaften (`properties`) in ATLAS.ti als `codes` modelliert werden. In Lila ist nur die Klasse `track` markiert. Sie repräsentiert ein über die GTM hinausgehendes „Werkzeug“ der Analyse. Um besser zwischen GTM-Klassen und in ATLAS.ti existierenden oder zu modellierenden unterscheiden zu können, sind in der Abbildung einige der ATLAS.ti-Klassennamen mit dem Qualifier *_atlas.ti* versehen. Die Beziehungen zwischen den Klassen sind mittels dreier Farben gruppiert: Dunkelblau sind Beziehungen markiert, die sich in der „realen Welt“ zwischen Einzelfällen abspielen, Hellblau solche, die in der konzeptuellen Welt „zu konstruieren“ sind. Rot gefärbte Beziehungen schlagen eine Brücke zwischen der realen und der konzeptuellen Welt. Beispielsweise kann man Konzeptualisierungen von Phänomenen in der „realen Welt“ durch `values` näher beschreiben – z.B. in dem man eine Annotation einer Quotation mit dem Konzept „Strategie“ durch ein „Organisation von Arbeitspaketen“ näher spezifiziert. Später – z.B. dann, wenn man bereits unterschiedliche Ausdifferenzierungen von Strategiephänomenen benannt hat – kann man auf konzeptueller Ebene davon reden, dass sich Strategien mittels einer Eigenschaft, die man dann vielleicht „Typ“ nennt, in fundamentaler Weise ausdifferenzieren lassen und diskutieren, was dies z.B. in Hinsicht auf bestimmte `conceptRelations` bedeutet. Der Beantwortung der letzteren Fragestellung kann man sich z.B. dadurch nähern, dass man bestimmte `conceptualisationRelations` untersucht.

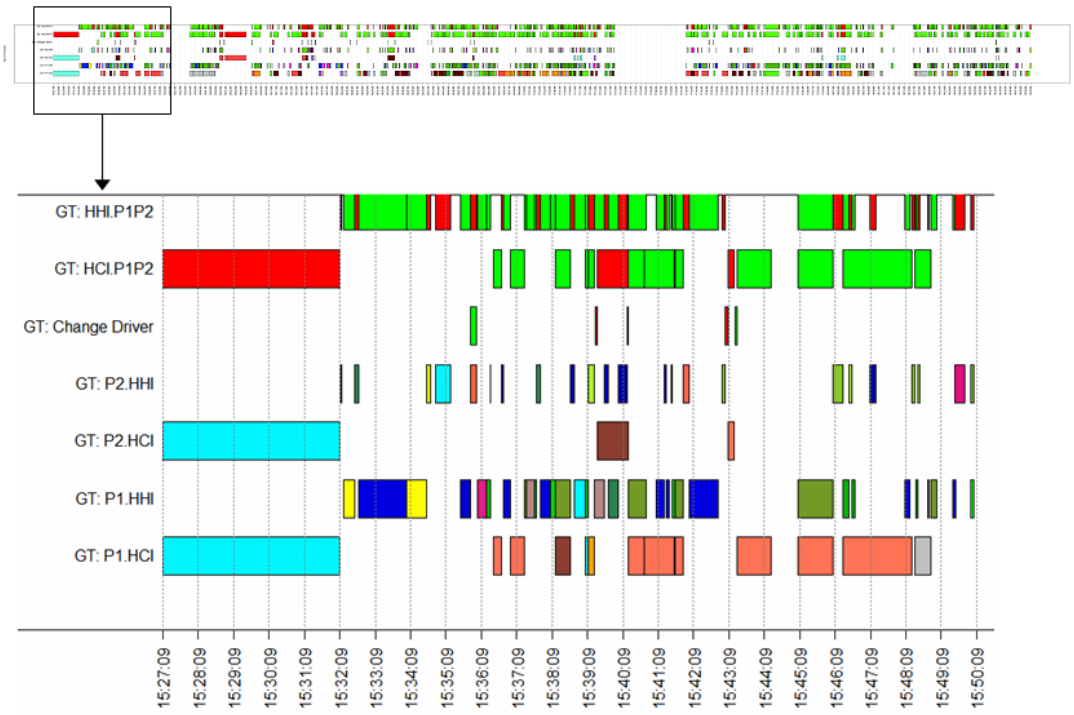


Abb. 5.2: Beispiel für die Visualisierung von Spuren (`tracks`) mittels *stripes*: Im oberen Teil der Abbildung ist – stark verkleinert – eine mittels *stripes* generierte Visualisierung eines Zwischenstandes der Kodierung von Sitzung ST1.1 zu sehen. Sie enthält sieben Spuren. Im unteren Teil ist ein Ausschnitt dieser Datenvisualisierung dargestellt. In der ersten Spur (mit GT:HHI:P1P2 bezeichnet) wird gezeigt, zu welchen Zeitpunkten sich die beiden Entwickler *P1* und *P2* verbal äußerten (HHI – *human-human interaction*). Hierzu wurde bei der Erstellung der Grafik festgelegt, dass in die Spur GT:HHI:P1P2 alle Annotationen aufzunehmen sind, die Äußerungen von *P1* oder *P2* markieren und dass Äußerungen von *P1* grün und solche von *P2* rot einzufärben sind. In der Spur darunter sind in gleicher Weise sonstige Handlungen der Entwickler dargestellt. Auch in den vier unteren Spuren sind Äußerungen bzw. Handlungen von *P1* und *P2* visualisiert – hier aber nach Personen sowie unterschiedlichen Formen von Äußerungen (GT.P2.HHI; GT.P1.HHI; z.B. einen Arbeitsschritt vorschlagen oder eine Frage stellen) bzw. Handlungen (GT.P2.HCI; GT.P1.HCI; z.B. Kode schreiben oder Kode ansehen) ausdifferenziert (pro Handlungsform eine eigene Farbe). Achtung: Vom Visualisierungsprogramm *stripes* werden Annotationen, die sich innerhalb einer Spur zeitlich überlagern, nicht gesondert verarbeitet. Es kann also dazu kommen, dass Rechtecke teilweise verdeckt oder komplett überschrieben werden. Da das Programm vor allem dazu diente, einen groben Überblick über Verläufe erlangen zu können, wurde es diesbezüglich nicht optimiert. Gab es Anlass zu der Vermutung, dass in einer Spur interessante Informationen durch Überdeckungen verloren gegangen sein könnten, wurde der Inhalt der Spur auf zwei Spuren verteilt. Die Abbildung wurde erstmalig in [181] veröffentlicht.

5.2.4 Praktik 4: Kodieren im Paar

Als zentrale Praktik wurde das Kodieren im Paar eingeführt. Sie fußt auf der Empfehlung von Glaser und Strauss, „die theoretischen Begriffe mit einem oder mehreren Teamkollegen zu diskutieren“ um „Verfehltes klarzustellen sowie Gesichtspunkte hinzuzufügen, auf die [die Kollegen] während der eigenen Kodierung und Datenerhebung gestoßen sind“ [77, S. 114], dehnt den Ratschlag aber auf alle Teile einer GTM-Untersuchung, insbesondere das offene Kodieren aus: Alle Formen der Kodierarbeit sollten – zumindest zu Beginn einer Untersuchung – von zwei Personen zusammen - und im Falle der Verwendung von ATLAS.ti oder einer anderen QDA-Software auch an einem Rechner – also analog der PP durchgeführt werden. Das Ziel beim Kodieren im Paar besteht darin, bei allen Entscheidungen einen Konsens herzustellen oder schriftlich festzuhalten, warum ein solcher nicht erreicht werden konnte.[15] Dies betrifft sowohl das Identifizieren von zu kodierenden Phänomenen wie auch Entscheidungen darüber, wann Phänomene beginnen bzw. wann sie enden und ob ein bereits bestehendes Konzept verwendet werden kann oder ein neues „kreiert“ werden muss. Durch die Praktik soll erreicht werden, dass

- Kodierentscheidungen kontinuierlich hinterfragt werden und
- beim schriftlichen Festhalten von Ergebnissen und Ideen – z.B. in Form von Kodenotizen — diszipliniert vorgegangen wird.

Folgenden Vorteile des Kodierens im Paar – im Gegensatz zum Solokodieren – konnten beobachtet werden:

- Die Beschreibungen (Definitionen) der Konzepte in Form von Kodenotizen wurden präziser, da sie direkt bei ihrer Entstehung (vom Partner) hinterfragt wurden – ein Effekt, der durch Praktik 2 (syntaktische Regeln für Konzeptnamen) forciert wurde.
- Unterschiede zwischen ähnlichen Konzepten wurden deutlicher herausgearbeitet, einerseits dadurch, dass die einzelnen Konzeptdefinitionen an und für sich präziser formuliert wurden, andererseits dadurch, dass zuverlässiger verhindert wurde, dass Konzepte auf unterschiedlichen Niveaus – z.B. auf verschiedenen Granularitätsstufen – entstanden bzw. ungeachtet dieser Eigenschaft miteinander verglichen wurden.
- Verbleibende Abgrenzungsprobleme zwischen Konzepten wurden nicht ignoriert sondern schneller bzw. früher gelöst. Dies führte zu weniger „falschen“, noch einmal zu überarbeitenden Konzeptualisierungen, minderte also den Arbeitsaufwand. Falls es nicht möglich war, ein bestimmtes

[15] Konsens darf hier nicht mit Kompromiss verwechselt werden. Es geht nicht darum, Entscheidungen zu fällen, denen beide gerade noch zustimmen können, sondern darum, zu verstehen warum es unterschiedliche Ansichten gibt, und diese notfalls auch (erst einmal) nebeneinander stehen zu lassen.

Abgrenzungsproblem zu lösen, half die Diskussion zumindest dabei, das Problem und somit auch die Daten besser zu verstehen.

- Der Blickwinkel auf die Daten (Praktik 1) wurde konsistenter eingehalten. Falls es zu einer Verletzung des Blickwinkels kam, wurde über den Grund dafür diskutiert und erörtert, ob der Blickwinkel anzupassen ist.

- Es wurden mehr relevante Phänomene entdeckt und kodiert.

Die hier vorgestellten Vorteile des Kodierens im Paar wurden nicht weiter empirisch validiert, passen aber zu Ergebnissen aus der psychologischen Forschung, die zeigen, dass Gruppen oft bessere Entscheidungen fällen bzw. Lösungen finden als Einzelpersonen (siehe z.B. [64, S. 315 ff]). Zwar kann die Zusammenarbeit von mehreren Personen unter bestimmten Umständen auch dazu führen, dass schlechtere Entscheidungen getroffen werden – ein Phänomen welches oft als Gruppendenken (*groupthink*) [99] bezeichnet wird – dies ist aber im Fall des Kodierens im Paar wenig wahrscheinlich. Denn Gruppendenken tritt nach Janis [99] vor allem in kohäsiven Gruppen auf, die strukturellen Mängeln ausgesetzt sind, z.B. dem Fehlen einer unabhängigen Gruppenleitung, und die in einem schwierigen Kontext, z.B. in einer Gefahrensituationen, agieren müssen.[16] Diese Bedingungen sind im Rahmen des Paarkodierens kaum relevant. Insbesondere auch deshalb, weil es zu den routinemäßigen Aufgaben eines Paarkodierers gehört, die Ideen des Partners – aber auch die eigenen – zu hinterfragen bzw. unterschiedliche Sichtweisen aufzuzeigen. Mit dieser Festlegung kann auch Ergebnissen aus der sozialpsychologischen Forschung begegnet werden, die besagen, dass – entgegen der landläufigen Meinung – Gruppen aufgrund von „basalen Gesetzmäßigkeiten des Diskutierens und Entscheidens in Gruppen“ beim Brainstorming weniger Ideen entwickeln als eine entsprechende Anzahl von Einzelpersonen in Summe [187]. Denn das Paar kann derartigen Phänomenen dadurch entgegenwirken, dass es sich regelmäßig vor Augen hält, dass es erst einmal nicht darum geht, Einigkeit zu erzielen sondern vielmehr darum, Optionen auszuloten.

[16] Die hier dargestellten Kausalitäten sind nicht unumstritten. Park weist in seiner 1990 veröffentlichen Literaturstudie [159] darauf hin, dass, obwohl das Modell des Gruppendenkens in den Sozialwissenschaften häufig zur Anwendung gebracht wird, bisher nur 16 empirische Studien über Gruppendenken publiziert wurden. Er hebt hervor, dass keine dieser Studien einen signifikanten Effekt der Kohäsion auf Gruppendenken nachweisen konnte.

Kapitel 6

Einführung in Basisschicht und Basiskonzeptmenge

Ich verneige mich vor den Begriffen.
Joseph Beuys

In der Einführung in Kapitel 1 wurde erläutert, in welcher Weise es hilfreich sein kann, qualitative Analysen des Paarprogrammierungsprozesses in partiell aufeinander aufbauenden Schichten (*layers*) durchzuführen. Außerdem wurde herausgearbeitet, dass in einem ersten Schritt eine so genannte Basisschicht (BS, *foundation layer* (FL)) entwickelt werden sollte, welche die grundlegenden Aktivitäten der PP mittels einer Konzeptmenge, der so genannten Basiskonzeptmenge (BKM, *base concept set*), beschreibt und somit einen Ausgangspunkt für das Verständnis der Abläufe des Paarprogrammierungsprozesses zur Verfügung stellt.[1] Die BS soll die Entwicklung aller weiteren Schichten dadurch erleichtern, dass bei den jeweils notwendig werdenden Konzeptionalisierungen nicht bei Null, sprich mit einer leeren Konzeptmenge begonnen werden muss, sondern stattdessen die Möglichkeit besteht, die BKM oder ihre Elemente, die *Basiskonzepte*, auszudifferenzieren, zu verallgemeinern, zu erweitern, zu beschneiden, zu ergänzen oder zum Zweck der Verbesserung der theoretischen Sensibilität einzusetzen.[2]

Das vorliegende Kapitel bildet den Einstieg in die BS bzw. BKM. Es beginnt mit einer ausführlichen Beschreibung des Zwecks der BS und einer präzisen Erörterung desjenigen Blickwinkels, unter dem sie aus den verwendeten Daten extrahiert wurde (Abschnitt 6.1). Hierbei wird auch der Begriff der *Basisaktivität* näher beleuchtet. In Abschnitt 6.2 wird dann erläutert, in welcher Weise die einzelnen Konzepte dargestellt sind, d.h. welche Aspekte von den Konzeptbeschreibungen adressiert werden. Anschließend wird der potentielle Einfluss des zu Beginn der Untersuchungen vorhandenen Vorwissens auf das Ergebnis diskutiert (Abschnitt 6.3). Außerdem werden Auswirkungen von Eigenschaften des Blickwinkels auf die Gestaltung des Analyseprozesses erörtert (Abschnitt 6.4).

[1] Ob bzw. in welchem Umfang die BS neben der BKM noch weitere Bestandteile (z.B. weitere Strukturbeschreibungen oder zusätzliche Regeln) besitzen sollte, war zu Beginn der Untersuchungen nicht klar. Erst im Verlauf der Untersuchungen zeigte sich, dass die Resultate über das hinausgehen werden, was man unter einer Menge versteht. So wird in den hier durchgeführten Erörterungen der Ergebnisse in der Regel von BS und nicht von BKM gesprochen. Nur wenn explizit die Menge der Konzepte Gegenstand der Betrachtungen ist, wird der Terminus BKM verwendet – auch in Form eines Zusatzes zum Kürzel BS (BS/BKM).

[2] Natürlich kann die BKM auch direkt, ohne Modifikationen vorzunehmen, eingesetzt werden.

Das Kapitel schließt mit der groben Gliederung der Elemente der BKM in drei Klassen[3] (Abschnitt 6.5).

Wie bereits in Abschnitt 4.2 ausgeführt, wurden für die Herleitung der BS primär die Paarprogrammierungssitzungen ST1.1, PR1.1 und PR2.1 verwendet. Hierbei kam die erweiterte GTM (siehe Kapitel 5) zum Einsatz – die Praktik der Paarkodierung (siehe S. 118) allerdings nur bei den ersten und in eingeschränkter Form bei den letzten (siehe Kapitel 10) Analyseschritten, das heißt vor allem im Zusammenhang mit ersten Untersuchungen der Sitzung ST1.1.[4]

Die Videos wurden, wie in Kapitel 5 erläutert, direkt kodiert, das heißt ohne sie vorher zu transkribieren.[5] Transkriptionen wurden allenfalls dann verwendet, wenn sich bestimmte Phänomene oder Episoden trotz intensiver Beschäftigung mit dem Videomaterial nicht konzeptuell erschließen ließen. In solchen Fällen kam ein eigenes, an die halbinterpretative Arbeitstranskription [55; 56; 57; 172] angelehntes Transkriptionssystem zum Einsatz (Details siehe Anhang B). Dieses System wird auch bei der Darstellung der in diesem Dokument aufgeführten Beispiele eingesetzt.

Insgesamt wurden über die Sitzungen ST1.1, PR1.1 und PR2.1 hinweg 3008 Kodeinstanzen an 2464 Videoausschnitte (Quotations) annotiert.

Eine weitaus weniger detaillierte Version von Teilen der Inhalte dieses Kapitels wurde 2008 auf dem Workshop der *Psychology of Programming Interest Group (PPIG)* vorgestellt [182].

Achtung: Der Aufbau der Namen der Elemente der Basiskonzeptmenge folgt – bis auf wenige Ausnahmen – der in Abschnitt 5.2.2 erläuterten Syntax. In den nachfolgenden Erläuterungen werden die Akteure (*P1* oder *P2*) allerdings in der Regel weggelassen.

6.1 Zweck und Blickwinkel der Basisschicht

Wie bereits in Kapitel 2 ausgeführt wurde, ist bisher noch nicht versucht worden, die bei der PP auftretenden Phänomene weitreichend oder gar vollständig zu erfassen und zu beschreiben. Genau genommen steht derzeit noch nicht einmal eine empirisch validierte Taxonomie für die elementaren/grundlegenden Aktivitäten der PP (Basisaktivitäten) zur Verfügung. Es gibt also bisher keine allgemein akzeptierte Terminologie,

– mit der sich die grundlegenden Vorgänge/Elemente, aus denen der Paarprogrammierungsprozess zusammengesetzt ist, formulieren lassen,

[3] Im Rahmen der Erörterungen der BS/BKM werden die Begriffe Menge und Klasse synonym verwendet.

[4] Diese Beschränkung hat nur formale, keine inhaltlichen Gründe. Schließlich sollte es sich bei einer Dissertation um eine weitgehend selbstständig durchgeführte Arbeit handeln. Qualitativen GTM-Analysen, die derartigen Restriktionen nicht unterworfen sind, sei angeraten, die Praktik der Paarkodierung im größeren Stil zu betreiben.

[5] Der Analyseprozess wurde unter Verwendung der QDA-Software ATLAS.ti durchgeführt (siehe auch S. 106).

- mit deren Hilfe relevante Blickwinkel für spezifische qualitative Studien ausgewählt werden können (siehe Abschnitt 5.2.1)
- und die darüber hinaus einen Ausgangspunkt für spezialisierte Konzeptualisierungen und nachfolgende Theoriebildungen liefert.

Diese Lücke soll mit der vorliegenden Arbeit geschlossen werden. Hierzu wird der Begriff Basisaktivität sukzessive mit Bedeutung gefüllt, unter anderem indem zwei zusammenhängende Fragen geklärt werden:

1. **Frage nach den Inhalten:** Welche Themenkreise oder Aspekte müssen auf welche Art und Weise durch Basiskonzepte abgedeckt werden?
2. **Frage nach der Granularität:** Wie feinkörnig soll dabei zwischen verschiedenen Konzepten differenziert werden?

Zu Beginn der Untersuchungen lagen die Antworten zu beiden Fragen weitgehend im Dunkeln. Weder war klar, in welche Richtung die inhaltlich orientierte Segmentierung des Prozesses zu verlaufen hat, noch auf welchem Detailniveau gearbeitet werden kann, muss oder sollte. Es schien auch nicht sinnvoll zu sein, die Granularität der Konzepte indirekt, z.B. durch eine „a-priori-Festlegung“ auf eine bestimmte Art der Datensegmentierung, zu steuern. Schließlich musste davon ausgegangen werden, dass Inhalt und Granularität in gewisser Weise interagieren. Somit war von Anfang an klar, dass die strukturellen Eigenschaften der Basisaktivitäten zusammen mit ihren inhaltlichen Schwerpunkten im Rahmen von in weiten Teilen explorativen Analysen aus den Sitzungsaufzeichnungen herausgearbeitet werden müssen.

Um hierbei nicht in den komplexen Daten, d.h. in den unzähligen über die Videos erfassten Details „unterzugehen“ bzw. nicht die Fähigkeit zu verlieren, erkennen zu können, welche Struktur zu einer hilfreichen Taxonomie führen kann, wurde ein aus drei Teilen bestehender Blickwinkel entwickelt (Praktik 1, beschrieben in Abschnitt 5.2.1). Er wurde wie folgt formuliert:

1. **Ausrichtung des Interesses:** Ziel der Untersuchungen soll es sein, die den Prozess der PP formenden grundlegenden Aktivitäten der *einzelnen* Paarmitglieder zu ermitteln und zu konzeptualisieren, um eine extensionale Beschreibung[6] eines auf die PP bezogenen Aktivitätsbegriffs zu erhalten. Dieser soll einen Ausgangspunkt für spezialisierte, theoriebildende Untersuchungen zur Verfügung stellen.

   Um eine Beeinflussung der Ergebnisse durch theoretisches Vorwissen, z.B. eine voreilige Schwerpunktsetzung, so gut es geht zu vermeiden, soll nur von

[6] Unter einer *extensionalen Definition* versteht man die „Aufzählung der Gegenstände oder Denotate, die unter einen Begriff fallen“ [119, S. 291]. Im Gegensatz dazu bezeichnet die *intensionale Definition* „die Angabe der Eigenschaften und Merkmale eines Begriffes, seine inhaltliche Bestimmung“ [120, S. 467].

einem umgangssprachlichen Verständnis des Begriffs Aktivität ausgegangen werden. Unter Aktivitäten soll grob das Handeln von Menschen, d.h. hier das in der Regel bewusste Agieren, Arbeiten oder Gestalten verstanden werden. Auf eine genauere Spezifikation wird verzichtet. Insbesondere soll nicht – zumindest nicht von Beginn an – auf spezielle Aktivitätsmodelle, z.B. aus der Aktivitätstheorie [118] bzw. der *Systemic-structural activity theory* [15; 16] oder auf Begriffe und Erkenntnisse aus der philosophischen oder soziologischen Handlungstheorie, zurückgegriffen werden.[7]

2. **Art der zu berücksichtigenden Phänomene:** Bei der Auswahl der zu analysierenden Phänomene soll der Sichtweise des klassischen Behaviorismus [213] gefolgt werden: Es dürfen nur direkt beobachtbare Phänomene, d.h. im wesentlichen Aspekte von tatsächlichem Verhalten [115, S. 41], berücksichtigt werden. Die handelnden Personen selbst sind als „Black Box" zu betrachten. Es darf nicht versucht werden, ihre innerpsychischen Vorgänge[8] bzw. mentale Abläufe oder kognitiven Prozesse, wie beispielsweise „Über eine Lösung nachdenken", zu erfassen und zu konzeptualisieren.[9] Außerdem soll keine Bewertung von Phänomenen, z.B. in Hinsicht auf Fortschritt bzw. Zielführung, vorgenommen werden.

   Es ist wichtig festzuhalten, dass aus der Festlegung auf beobachtbare Aspekte von Verhalten nicht folgt, dass es sich bei der Herleitung der BS und insbesondere der BKM um ein vollständig objektives Verfahren, wie es John B. Watson und viele andere Vertreter des Behaviorismus im Sinn hatten [115, S. 41], handeln kann. Schließlich werden im Rahmen des verwendeten GTM-Prozesses Phänomene interpretiert.[10] Diese Interpretationen sollten allerdings intersubjektiv nachvollziehbar sein.

   An dieser Stelle sei abschließend noch auf eine Aussage von Jörg Strübing [201, S. 79f] verwiesen. Dieser betont, dass in der GTM, obwohl das klassische Gütekriterium „Objektivität" überhaupt nicht thematisiert wird, „ein pragmatisches Konzept von Objektivität existiert [...]". Den Ausgangspunkt für diese Sichtweise bildet seine Feststellung, dass „gerade die in der grounded theory systematisch maximierte Vielzahl möglicher Lesarten [...]

[7] Obwohl die Herleitung der BS derart konzipiert war, dass sie möglichst unvoreingenommen stattfinden kann (siehe hierzu auch Abschnitt 6.3), ist nicht auszuschließen, dass das Ergebnis letztendlich nicht nur strukturell, sondern auch inhaltlich durch die Festlegung einer Syntax (siehe Abschnitt 5.2.2) beeinflusst wurde. Eine Erörterung dieser Frage sprengt allerdings den Rahmen der vorliegenden Arbeit und wird deshalb nicht vorgenommen.

[8] Der Begriff „innerpsychisch" (*intrapsychic*) stammt aus der Psychologie. Er bezeichnet die inneren psychologischen Prozesse von einzelnen Personen, z.B. moralische Konflikte.

[9] Wie später (siehe S. 250 f) diskutiert werden wird, adressiert die BKM trotz dieser Festlegung gewisse Ergebnisse kognitiver Prozesse explizit. Dies geschieht aber nur dann, wenn bestimmte Indizien vorliegen.

[10] An dieser Stelle sei bemerkt, das die Anfänge der GTM vermutlich dadurch gekennzeichnet waren, „dass Strauss und Glaser primär eine wissenschaftspolitische Manifestation gegenüber dem Behaviorismus und dem quantitativen Mainstream vornehmen wollten" [111].

das Ausgangsmaterial für diskursiv zu entwickelnde und wiederum empirisch zu überprüfende Theorieentwürfe“ bildet. Nach Strübing ist es „das Wechselspiel von Objektkonstruktion und sozialem Handeln mit diesen Objekten, das im sozialen Prozess des Forschens typischerweise (bekanntlich nicht immer) zu einer Schließung in Form einer weitgehend einheitlichen, im Meadschen Sinne ‚objektiven‘ Perspektive führt“ (vgl. [135]).

3. **Ergebnistyp:** Das Ergebnis soll aus einer Menge von Konzepten (den Basiskonzepten) bestehen, die weder unbedingt mittels Eigenschaften ausdifferenziert noch zwingend mit Relationen versehen sein müssen. Das Herausarbeiten von Eigenschaften soll weitgehend der Entwicklung anderer Schichten überlassen werden, da Eigenschaften erst vor dem Hintergrund eines konkreteren Interessenschwerpunktes wirkungsvoll ausgewählt bzw. ausgearbeitet werden können. Außerdem soll kein festes Kodierschema (bzw. vollständiges deskriptives System), sondern ein in zukünftigen Studien ggf. zu erweiterndes oder umzuschreibendes „Framework“ geschaffen werden. Insbesondere soll die BKM dabei helfen, spezielle Untersuchungsfokusse auswählen zu können.

   Obwohl es also bei der Entwicklung der BKM weder darum geht, bestimmte Eigenschaften von Konzepten auszuarbeiten noch darum, Konzepte in eine Theorie einzubetten, müssen die Elemente der BKM und ihre Bezüge zu konkreten Phänomen zumindest soweit präzisiert werden, dass ein weitgehend konsistenter Einsatz beim Kodieren möglich ist. In gewisser Weise wird es also nötig werden, auch Eigenschaften zu betrachten und die Beschreibungen der Konzepte bezüglich der hieraus resultierenden Erkenntnisse zu erweitern. Dies bedeutet allerdings nicht, dass die Beschreibungen eine Dichte erreichen müssen, wie sie in der Regel nur durch ein auf bestimmte Aspekte hin durchgeführtes theoretisches Sampling (siehe S. 52 f) zu Tage fördern kann (Stichwort: theoretische Sättigung (siehe S. 55)). Im Detail wird hierauf im nächsten Abschnitt eingegangen.

   Achtung: Bei der Darstellung der BS bzw. der BKM wird ausschließlich der Terminus Konzept verwendet. Die Begriffe Kode und Kategorie werden hingegen nicht eingesetzt. Hiermit soll zum Ausdruck gebracht werden, dass es sich bei den Elementen der BKM weder um reine Begriffsassoziationen noch um fertige Bestandteile einer nur noch auszuformulierenden Theorie handelt (siehe auch Abschnitt 3.1.2). Zusammenhängende Gruppen von Konzepten werden Klassen genannt.

Der auf diese Weise definierte Blickwinkel trägt den oben gemachten Überlegungen zum Detailierungsgrad der BKM Rechnung, indem er weder eine Aussage über die Form bzw. Granularität der zu verwendenden Datensegmentierung noch

über die angestrebte Granularität der BKM macht.[11] Beide Aspekte werden vielmehr als Ergebnisse der Analyse betrachtet. Hierbei wird davon ausgegangen, dass sich die Granularität(en) der Konzepte und die der Segmente über den Verlauf des Analyseprozesses hinweg und im Kontext der sich herausbildenden Theoriebestandteile – in Form von (vorläufigen) Konzepten, Beschreibungen (Memos), aber evtl. auch ersten Beziehungen sowie einer sich entwickelnden, tieferen Einsicht, worum es bei der Untersuchung genau geht – durchgehend gegenseitig beeinflussen.

Dies bedeutet u. a., dass es zumindest in den ersten Phasen der Analyse nicht möglich sein kann, verbindlich von einer Aktivität zur nachfolgend auftretenden überzugehen. Als Orientierungshilfe kann nur eine grobe, an Robillard et al. [175] angelehnte Regel verwendet werden, nach der ein neues Segment u. a. dann beginnen muss, wenn für die fortlaufend zu beobachtenden Phänomene die zuletzt vorgenommene Annotation nicht mehr passt. Diese (zugegebenerweise vage) Festlegung einer Granularität für Konzepte, wie auch die dadurch entstehende Größe der Segmente muss aber explizit als vorläufig angenommen werden. Jeder Zeit kann es notwendig werden, diese vor dem Hintergrund der aus den Daten heraus entstehenden Einsichten zu ändern.

Da die Granularität der Konzepte sowie die der Segmentierung also von den sich einstellenden Einsichten und den damit verbundenen, nicht zwangsläufig kanonischen Entscheidungen abhängt, kann das Ergebnis der Untersuchungen bzw. die resultierende Struktur nicht als kanonisch angesehen werden. Es liegt vielmehr in der Hand der Untersuchenden, ob und wie die identifizierten Teilbereiche ausgearbeitet werden. Es muss allerdings darauf geachtet werden, dass das im Blickwinkel formulierte Ziel der Analyse nicht aus den Augen verloren wird. Zur Unterstützung hierbei wurden zusätzliche, nicht explizit aus dem Blickwinkel hervorgehende Entscheidungsleitlinien formuliert. Sie haben weniger die Form einer konkret anwendbaren Regel, sondern sind Reflexionshilfen, die es ermöglichen sollen, den Entscheidungsprozess, speziell beim Kodieren im Paar[12] bezüglich des übergeordneten, im Blickwinkel formulierten Ziels, zu steuern. Sie besagen:

1. Die BKM sollte groß genug sein, um zwischen wesentlichen Phänomenen unterscheiden zu können, und klein/handlich genug, um eine konsistente Anwendung zu erlauben. Entscheidungen über Ausdifferenzierungen oder Zusammenlegungen von Konzepten müssen im Laufe des Prozesses immer wieder überprüft werden.

2. Die BKM sollte eine möglichst gleichförmige Struktur besitzen und deswegen ähnliche oder sogar dieselben Unterscheidungskriterien (bzw. Typen

[11] Wenn von der Granularität der Konzepte bzw. Phänomene gesprochen wird, soll keineswegs impliziert werden, dass über alle Konzepte oder Phänomene hinweg auf einer halbwegs identischen Ebene zu differenzieren ist. Es ist plausibel, dass es auch im Rahmen der BKM sinnvoll sein kann, verschiedene Teilaspekte unterschiedlich genau auszudifferenzieren.

[12] Eine Anwendung von Praktik 4 (Kodieren im Paar, siehe Abschnitt 5.2.4) kann die Situation verbessern, da die Wahrscheinlichkeit erhöht wird, dass schon frühzeitig unterschiedliche Varianten bez. Granularität bzw. Segmentierung zu Tage treten bzw. diskutiert werden.

von Entscheidungskriterien) in unterschiedlichen Teilbereichen der resultierenden Menge verwenden. Auch dies sollte immer wieder überprüft werden. Sollten in verschiedenen Bereichen unterschiedliche Kriterien bzw. Arten von Kriterien notwendig werden, muss herausgearbeitet werden, warum dies so ist.

In Abschnitt 11.1 wird diskutiert, in welcher Weise der Blickwinkel eingehalten und die darüber hinausgehenden Ziele erreicht wurden. Insbesondere wird erörtert, in welcher Form Eigenschaften von Konzepten zumindest implizit verwendet wurden.[13] Hierauf wird auch im nächsten Abschnitt eingegangen.

6.2 Darzustellende Aspekte der Basisschicht

Wie bereits erläutert, wird durch den Blickwinkel die anzuvisierende Ergebnisstruktur dahingehend festgeschrieben, dass sie im Wesentlichen aus einer Menge von Konzepten, der so genannten BKM, bestehen soll. Ein Aussage über Form und Umfang wird in diesem Zusammenhang aber nicht gemacht. Es ist zwar offensichtlich, dass Konzeptnamen für sich allein genommen nicht ausreichend sind und durch Beschreibungen ergänzt werden müssen (Kodenotizen, siehe S. 62 und S. 76), es liegt aber nicht, zumindest nicht vollständig, auf der Hand, welche Aspekte hierbei im Einzelnen adressiert werden sollten. Erst im Laufe der Untersuchungen zeigte sich, dass vor allem zwei Typen von Merkmalen zu berücksichtigen sind, primär charakterisierende Konzepteigenschaften sowie spezielle Eigenschaften verbaler Kommunikation:

1. **Primär charakterisierende Konzepteigenschaften:** Obwohl der Blickwinkel beinhaltet, dass Konzepte nicht (bzw. nicht zwingend) mittels Eigenschaften ausdifferenziert werden sollten, ist sofort einsichtig, dass es wenig sinnvoll ist, dieser Vorgabe ohne Ausnahmen zu folgen. Schließlich handelt es sich auch bei den Konzeptbeschreibungen im Wesentlichen um Erörterungen von Eigenschaften bzw. von Werten von Eigenschaften. Auf diese kann nicht verzichtet werden. Stattdessen muss zwischen darzustellenden und nicht darzustellenden Eigenschaften unterschieden werden. Hierzu wurde eine grobe Leitlinie verwendet. Sie besagt, dass zumindest all diejenigen, direkt mit der Semantik eines Konzeptes zusammenhängenden Eigenschaften[14] erläutert werden müssen, deren Kenntnis unverzichtbar dafür ist, dass das Konzept im intendierten, aus den Daten ermittelten Sinn verstanden

[13] Eigenschaften wurden z.B. dazu verwendet, um (wie in Abschnitt 3.2.1 erläutert) aus vorhandenen Konzepten neue Konzepte zu generieren (falls eine Aufsplittung an dieser Stelle als wesentlich bewertet wurde) oder um den Umfang bzw. die Bedeutung eines speziellen Konzeptes besser beschreiben zu können.

[14] Im Rahmen von Kommunikation kann man zwischen Konzepteigenschaften, also solchen, die die Bedeutung eines Konzeptes näher beschreiben, und Formulierungseigenschaften unterscheiden.

und verwendet werden kann. Dies bedeutet, dass ein Konzept zumindest soweit beschrieben werden muss, dass anhand dieser Beschreibung alle bei der Herleitung verwendeten bzw. diesbezüglich beobachteten Phänomene nachträglich als derartige identifiziert werden können. Solche Eigenschaften sollen im Weiteren *primär charakterisierend* heißen.

Achtung: Die auf diese Weise beschriebenen primär charakterisierenden Eigenschaften entsprechen im Allgemeinen nicht denjenigen Eigenschaften, die in der Erkenntnistheorie oder allgemeiner in der Philosophie als zur Natur einer Sache gehörend bzw. der Sache *wesentlich* bezeichnet werden (siehe z.B. [60, S. 28]). Denn zum einen ist eine Eigenschaft in den erwähnten Disziplinen in der Regel das, was im Rahmen der vorliegenden Arbeit als Wert einer Eigenschaft bezeichnet wird (siehe Abschnitt 5.2.3; wobei primär charakterisierende Eigenschaften im Rahmen der BKM nicht immer explizit benannt, sondern nur durch die Erläuterung ihrer Werte eingeführt werden), und zum anderen sind primär charakterisierende Eigenschaften bzw. ihre einzelnen Werte oft weder notwendig noch hinreichend dafür, dass es sich um ein bestimmtes Ding handelt. Vielmehr gilt Folgendes: Sei K ein Konzept, welches durch eine primär charakterisierende Eigenschaft E_K mit den Werten W_E^1 und W_E^2 beschrieben ist. Dann muss für ein Phänomen P zumindest W_E^1 oder W_E^2 gelten, um es mit K konzeptualisieren zu können.[15] Verschiedene Ausprägungen einer primär charakterisierenden Eigenschaft weisen dabei oft auf derart unterschiedliche Ereignisse hin, dass ohne die Beschreibung dieser Eigenschaft und ihrer Werte gar nicht zu erkennen wäre, dass es sich um denselben Typ von Phänomen handelt (siehe etwa die unterschiedlichen Strategietypen auf S. 216 f). Hat E_K nur einen möglichen Wert, so kann man E_K mit diesem assoziieren und von einer zur Natur der Sache gehörenden, also notwendigen Eigenschaft sprechen. Solche Eigenschaften lassen sich natürlich auch durch die Disjunktion von Werten von primär charakterisierenden Eigenschaften bilden. An dieser Stelle sei aber bemerkt, dass die Beschreibungen von primär charakterisierenden Eigenschaften immer als vorläufig zu betrachten sind. Sie entsprechen dem, was man (bisher) anhand der Daten über anscheinend wesentliche Aspekte der sich in Entwicklung befindlichen Konzepte bzw. der sich herausbildenden Konzeptmenge gelernt hat. So kann es sein, dass es im Laufe einer Analyse notwendig wird, den Wertebereich einer primär charakterisierenden Eigenschaft zu erweitern, im obigen Beispiel also einen neuen Wert W_E^3 von E_K zuzulassen. Außerdem handelt es sich bei den Werten nicht immer um hundertprozentig trennscharfe Kriterien (siehe z.B. S. 257).

[15] Man kann von einer *hinreichenden* primär charakterisierenden Eigenschaft E_K sprechen, wenn es für die Zuordung eines Phänomens zur Klasse K hinreichend ist, dass das Phänomen einem der Werte von E_K genügt. Im Allgemeinen ist dem aber nicht so. Vielmehr gilt: Genügt ein Phänomen einem Wert einer primär charakterisierenden Eigenschaft E_K, so ist dies nur ein Indiz dafür, dass es vom Typ K sein könnte.

***Exkurs 4:** Äußerungen*

Unter einer Äußerung wird in der Linguistik ein „Abschnitt in der Rede einer einzigen Person [...], vor und nach welchem die Person schweigt“ [31, S. 12 f] bzw. eine „sinnvoll-abgeschlossene sprachlich-kommunikative Einheit, die aus einem oder mehreren Sätzen, aber auch Teilen von Sätzen bestehen kann“ verstanden [119, S. 117]. Die BS folgt der zweiten Definition insoweit, dass die Eigenschaft sinnvoll-abgeschlossen als „mit einem Element der BKM zu annotieren“ interpretiert wird. Dies bedeutet unter anderem, dass zu Beginn der hier geschilderten Untersuchungen nur ein grobes Verständnis von dem existierte, was als sinnvoll-abgeschlossen angesehen werden soll. Schließlich war die BKM zu diesem Zeitpunkt noch leer. Neben dem Begriff Äußerung wird auch noch der Terminus Teiläußerung verwendet. Dies geschieht vornehmlich dann, wenn betont werden soll, dass es sich um eine Äußerung handelt, die als Teil einer anderen betrachtet werden kann.[a] Äußerungen werden dabei nie unabhängig von dem Kontext betrachtet, in dem sie gemacht werden. Wenn also von einer Äußerung X gesprochen wird, ist immer Äußerung X in einem bestimmten Beobachterkontext K gemeint.[b]

[a] Im weiteren Verlauf der vorliegenden Arbeit wird erläutert, dass die BKM Elemente enthält, die „ineinander geschachtelt“ eingesetzt werden können (siehe z.B. Abschnitt 7.7).

[b] Was genau unter dem Kontext (oder auch Beobachterkontext) einer Äußerung verstanden werden soll, wird in Abschnitt 9.3.3 erläutert. Vorerst reicht ein umgangssprachliches Verständnis des Begriffs als „Zusammenhang“.

2. **Spezielle Eigenschaften verbaler Kommunikation:** Wie im weiteren Verlauf erläutert werden wird (siehe Abschnitt 6.5), adressiert die BKM überwiegend verbale Äußerungen, d.h. Phänomene, die sich im Kern über natürliche Sprache zeigen (für Details zum Begriff Äußerung siehe Exkurs 4). Da mittels dieser vielerlei Möglichkeiten existieren, um ein und denselben Aspekt zu artikulieren, können gleichartige Phänomene in gänzlich unterschiedlicher Gestalt auftreten. Anders ausgedrückt: Es gibt weder ein 1:1-Verhältnis zwischen syntaktischer Form und Satztyp (Deklarativsatz (Aussagesatz), Interrogativsatz (Fragesatz), Imperativsatz (Aufforderungssatz), Exklamativsatz (Ausrufesatz) und Optativsatz (Wunschsatz)), noch zwischen Satztyp und ausgeführtem Sprechakt (siehe Exkurs 6). Vielmehr ist eine kontextfreie oder situationsenthobene Äußerung nach [122, S. 192] „im Grunde“ pragmatisch mehrdeutig.

Um die Verwendung der BS zu erleichtern bzw. eine konsistente Kodierung zu ermöglichen, sollten deshalb die im Rahmen von Äußerungen eines bestimmten Typs zu erwartenden Formulierungsphänomene durch die Konzeptbeschreibungen adressiert bzw. diskutiert werden – zumindest soweit,

wie es die Untersuchungsdaten hergeben.[16] Z.B.: Da ein Fragesatz nicht mit dem Sprechakt des Fragens verbunden sein muss, legt die BS fest, Vorschläge auch dann mit den entsprechenden Konzepten zu annotieren, wenn sie in Frageform formuliert sind (siehe S. 177).

Zum Abschluss dieses Abschnittes sei darauf hingewiesen, dass im Zusammenhang mit den gerade beschriebenen Eigenschaften im Weiteren oft Regeln angegeben werden, die es ermöglichen sollen, in einer bestimmten Situation zwischen zwei oder mehreren potentiell passenden Konzepten entscheiden zu können. Diese Regeln können z.B. folgende Form haben:

> Wenn ein Phänomen die Eigenschaft X hat und diese Eigenschaft darauf hindeutet, dass sowohl Konzept U wie auch Konzept V passend sind, ist im Rahmen der BS nur U zu annotieren.[17]

Die Festlegung solcher Regeln steht nicht im Widerspruch dazu, dass die BS und insbesondere die BKM als ein zu modifizierendes Framework konzipiert ist. Es ist nämlich explizit vorgesehen, dass diese Regeln – falls nötig – im Rahmen spezialisierter Untersuchungen geändert werden können, z.B. um bestimmte Phänomene sichtbar und somit untersuchbar zu machen. Im Zusammenhang mit der BS dienen sie lediglich als Orientierungshilfe und demonstrieren darüber hinaus, in welcher Weise die Elemente konsistent aus den Daten extrahiert wurden. Sie sollen dabei helfen, die BS/BKM besser zu verstehen. Ihre Angemessenheit muss aber immer unter Berücksichtigung des gerade aktuellen Kontextes hinterfragt werden. In diesem Zusammenhang beachte man auch die folgende Aussage des ungarischen Mathematikers George Pólya [170, S. 170]:

> „Eine Regel dem Buchstaben nach anwenden, starr, blind, in Fällen, wo sie paßt, und in Fällen, wo sie nicht paßt, ist Pedanterie. Manche Pedanten sind arme Toren; sie haben niemals die Regel verstanden, die sie so gewissenhaft und so unterschiedslos anwenden. [...] Eine Regel mit natürlicher Leichtigkeit anwenden, mit Urteil sie da anwenden, wo sie paßt, und ohne daß je die Worte der Regel den Zweck der Handlung oder die Gunst der Lage verbergen, ist Meisterschaft."

6.3 Potentieller Einfluss des Vorwissens

Wie bereits in Kapitel 3 dargestellt, kann nicht verhindert werden, dass Untersuchungen auf Basis der GTM durch das Vorwissen des Forschenden beeinflusst

[16] Wie in Abschnitt 7.1 erläutert werden wird, geht es bei der Identifikation von Konzepten auch darum, die Illokution (siehe Exkurs 6) von Äußerungen zu verstehen. Hierzu werden in der Regel vielfältige Merkmale betrachtet. Bei weitem nicht alle dieser Indizien sollten im Rahmen der Beschreibung der BKM diskutiert werden. Beispielsweise muss nicht erläutert werden, wie man im Normalfall erkennen kann, dass es sich bei einer Äußerung um eine Frage handelt.

[17] Z.B. wird auf diese Weise festgelegt, was zu tun ist, wenn ein Designvorschlag einen Vorschlag zum nächsten Arbeitsschritt beinhaltet (siehe S. 200).

werden. Strauss und Corbin heben sogar hervor, dass das Einbringen von Vorwissen notwendig ist, um die theoretische Sensibilität zu verbessern, d.h. um die Fähigkeit auszubauen, den Daten Bedeutung zu verleihen und konzeptionelle Beschreibungen zu formulieren (siehe Abschnitt 3.2.2). Ihre Anmerkungen bedeuten allerdings nicht, dass schon im Vorfeld versucht werden sollte, möglichst viele potentiell interessante Theorien oder Modelle zu sammeln. Schließlich soll Theorie entdeckt und nicht nur verifiziert werden.

So wurde bei den hier dargestellten Untersuchungen zunächst, zumindest was große Teile des Analyseprozesses, insbesondere die ersten Phasen betrifft, auf eine explizite Erweiterung des Vorwissens (z.B. bez. Tätigkeitstheorie, Lernpsychologie, kognitiver Psychologie etc.) verzichtet. Zu Beginn der Analysen beschränkte es sich im Wesentlichen auf folgende allgemeine und spezifische Fachkenntnisse:

Domainwissen: Beim Autor sowie den anderen an der Entwicklung der BS beteiligten Personen (siehe Praktik 4 in Abschnitt 5.2.4) handelt es sich um Informatiker mit dem Schwerpunkt Software Engineering und einem besonderen Interesse an agilen Softwareprozessen. Somit ist davon auszugehen, dass fachspezifische Kenntnisse Auswirkungen auf die Analysen gehabt haben.

Praktische PP-Kenntnisse: Die konkrete PP-Erfahrungen des Autors und der anderen an der Entwicklung der BS bzw. BKM beteiligten Personen beschränkten sich auf wenige explizit als solche angegangene und eine nicht zu bestimmende Anzahl von nicht explizit als solche angegangene Paarprogrammierungssitzungen. Bei Letzteren handelt es sich in der Regel um spontane bzw. ungeplante Zusammenkünfte am Rechner, wie sie beispielsweise zum Zwecke einer konkreten Defektbehebung stattfinden.

Keine dieser Sitzungen hatte das Ziel, Einblicke in den Ablauf des PP-Prozesses zu erhalten. Insbesondere wurden keine Sitzungen zum Zwecke von Introspektion durchgeführt. Trotzdem ist davon auszugehen, dass (unbewusst) Erfahrungen aus diesen Sitzungen in die Analysen eingegangen sind.

Theoretische PP-Kenntnisse: Im Rahmen der Vorbereitung auf die hier beschriebenen Untersuchungen wurde die PP-Literatur sondiert (siehe Kapitel 2). Dies hat mit großer Wahrscheinlichkeit den Blick auf die Paarprogrammierungssitzungen verändert. Außerdem waren dem Autor, als Mitglied der Gemeinde der Softwareentwickler, die in diesem Umfeld kursierende Vermutungen bez. PP bekannt – z.B. häufig geäußerte Vorbehalte.[18]

[18] Einer der häufig diskutierten Aspekte der PP ist der Wissenstransfer (z.B. von Projektwissen, technischem Know-How etc.). Dieser Aspekt nimmt auch einen großen Raum in der BKM ein. Ein Zusammenhang zwischen dem Vorwissen über diesen Aspekt und der Rolle dieses Aspektes innerhalb der BKM kann nicht ausgeschlossen werden.

6.4 Eingesetzte Praktiken der GTM

Durch den oben beschriebenen Blickwinkel wird festgelegt, dass die BS im Wesentlichen aus einer Menge von Konzepten, der BKM, bestehen soll. Die BS bildet also nur einen Teil von dem, was von Glaser und Strauss [76] (und auch allgemein) mit dem Begriff Theorie bezeichnet wird (siehe Kapitel 3). Hieraus folgt, dass die GTM, welche im Kern dahingehend konzipiert ist, Theorie zu entdecken und zu formulieren, nicht im vollen Umfang zum Einsatz gebracht werden muss. So ist z.B. sofort einsichtig, dass auf das selektive Kodieren verzichtet werden kann bzw. soll (siehe hierzu auch S. 69f).[19]

Bei anderen Analyseschritten, z.B. dem offenen Kodieren, ist die Lage weniger eindeutig. So ist zwar offensichtlich, dass die Herleitung von In-vivo-Konzepten unverzichtbar ist bzw. die zentrale Aktivität bei der Konstruktion der BKM darstellt, es ist aber nicht vollständig klar, inwieweit dabei auch Dimensionalisierungstechniken zum Einsatz kommen müssen. Denn einerseits ist die Ausdifferenzierung von Konzepten mittels Eigenschaften kein primäres Ziel der Analyse, andererseits kann auf eine solche aber nicht vollständig verzichtet werden, da sie benötigt wird, um Klarheit darüber zu erlangen, wo Konzeptgrenzen verlaufen sollen bzw. können (siehe z.B. die Diskussion über primär charakterisierende Eigenschaften in Abschnitt 6.2). Analoges gilt auch für das axiale Kodieren, unabhängig davon, welches heuristische Paradigma eingesetzt wird.[20] Denn schließlich ist plausibel, dass es, auch wenn keine vollständigen Analysen von Phänomenen bez. eines bestimmten Paradigmas angestrebt werden, notwendig werden kann, aufgebrochenen Strukturen in neuer Art zusammenzufügen oder das Handeln und Interagieren näher zu untersuchen, z.B. um einzelne Konzepte besser zu verstehen oder gegen andere abzugrenzen.

Zusammenfassend kann festgehalten werden, dass bei der Herleitung der BS nicht alle Methoden der GTM vollständig eingesetzt werden müssen. Diese Einschränkung ist eine gewollte Folge des Schichtenkonzeptes. Die Analyse vieler weiterer Schichten wird sich von ihr lösen müssen, d.h. die GTM im vollen Umfang zum Einsatz bringen.

6.5 Grobe Gliederung der Basiskonzeptmenge

Schon frühzeitig im Analyseprozess wurde erkannt, dass sich die Elemente der BKM auf zwei disjunkte Klassen (HHI und HCI/HEI) verteilen lassen.[21] Sie werden im Weiteren erläutert.

[19] Für den Fall der GTM nach Strauss und Corbin [198].

[20] Neben dem paradigmatischen Modell von Strauss und Corbin [198] existieren auch eine Reihe anderer Raster, z.B. das von Bogdan und Biklen [26].

[21] Neben den Konzepten der beiden Hauptklassen enthält die BKM noch eine geringe Anzahl von weiteren Elementen. Diese sind allerdings eher deskriptiver Natur und dienen vornehmlich dazu, zentrale Sitzungsereignisse wie z.B. einen Driverwechsel zu markieren (siehe Abschnitt 8.2).

6.5.1 HHI

Die Klasse *HHI* (*human-human interaction*) umfasst all diejenigen Konzepte, mit denen Interaktionen zwischen den Paarmitgliedern, also aufeinander bezogene Handlungen, beschrieben werden können. Es wurde festgelegt, dass diese Klasse – zumindest was die BS/BKM angeht – nur solche Elemente enthalten soll, die der Konzeptualisierung verbaler Äußerungen dienen (siehe Exkurs 5 auf S. 137, in dem die Bedeutung und Abgrenzung der Begriffe Interaktion und Kommunikation erläutert wird).[22] Dies hatte im Wesentliche zwei Gründe:

1. Während der Mitschnitt der hier betrachteten Sitzungen die verbale Kommunikation der Entwickler vollständig erfasst hatte[23], waren die nonverbale Kommunikation und die sonstigen Interaktionen zwischen den Partnern nur unvollständig aufgezeichnet worden. Dies ist auf folgende technische Randbedingungen zurückzuführen:

   - Zum einen sind die Entwickler aus einer festen Kameraposition heraus beobachtet worden. Bewegungen der Probanden führten dazu, dass sie bzw. ihre Gesichter nicht durchgehend vollständig „im Bild“ waren.
   - Zum anderen stand das Webcam-Bild bei den Analysen (der Sitzungen ST1.1, PR1.1 und PR2.1) lediglich in einer Auflösung von 176 x 144 bzw. 320 x 240 Bildpunkten und mit einer Bildwiederholfrequenz von 5 Frames/sec bzw. 15 Frames/sec zur Verfügung (siehe Tabelle 4.1). Somit war es nur eingeschränkt dafür geeignet, Details wie z.B. den Gesichtsausdruck, in jeder Situation zweifelsfrei deuten zu können.[24]

2. Nonverbale Kommunikation, speziell solche, die nicht offensichtlich verbale Äußerungen ersetzen soll (wie z.B. Nicken), schien nur selten stattzufinden und wurde darüber hinaus nie als eigenständige Basisaktivität im Sinne des Blickwinkels (siehe Abschnitt 6.1) interpretiert. Gleiches gilt für sonstige Interaktionen.

Achtung: Im Zusammenhang mit den HHI-Konzepten sollte Folgendes beachtet werden:

1. HHI-Konzepte, genau genommen die Objektbestandteile der HHI-Konzeptnamen (siehe Abschnitt 7.1), adressieren ausschließlich Elemente des Diskurses bzw. der *Diskurswelt*[25], d.h. Formulierungen von Gedanken, Wissen

[22] An dieser Stelle wird nicht unterschieden, ob eine verbale Äußerung wirklich für den Partner bestimmt ist oder aus anderen Gründen hervorgebracht wird.

[23] Nur in seltenen Fällen sprachen die Entwickler so leise oder so undeutlich, dass es in keiner Weise möglich war, ihre Äußerungen zu deuten.

[24] Die niedrige Qualität des Videobildes war technischen Restriktionen geschuldet. Bei weiterführenden Untersuchungen sollte versucht werden, diese zu überwinden, z.B. durch den Einsatz einer leistungsfähigeren Hardware.

[25] Der Begriff Diskurs soll an dieser Stelle nicht auf spezielle philosophische oder linguistische Diskurstheorien verweisen, sondern erst einmal nur hin und her gehende Gespräche bezeichnen.

oder Erkenntnissen über einen Sachverhalt und nie den Sachverhalt selbst. Zur Verdeutlichung betrachte man folgende Beispiele:

- Mit dem Konzept *propose_step* wird ein vom Sprecher erdachter und dann formulierter, in der nahen Zukunft potentiell auszuführender Arbeitsschritt referenziert, also die Sichtweise des Sprechers auf eine noch durchzuführende Tätigkeit[26], und nicht etwa die Menge der bei der späteren Umsetzung des Schrittes tatsächlich resultierenden Arbeiten (als Elemente der so genannten *Tätigkeitswelt*). Eine artikulierte Aktion, bzw. genau genommen artikulierte Vorstellung von einer Aktion, wird vor allem dadurch zu einem *step*, weil der Sprecher sie derart auffasst, und nicht weil das referenzierte Objekt, also die potenziell resultierende Aktion, über sprecher- bzw. formulierungsunabhängige Kriterien als *step* identifiziert werden kann.
- Mit dem Objekt *source of information* wird nicht eine Informationsquelle an und für sich, sondern das Wissen einer Person über eben diese adressiert. So werden mit *remember_source of information* Äußerungen annotiert, in denen ein Entwickler erkennbar die Erinnerung verbalisert, dass es eine bestimmte Informationsquelle gibt. Ob sie wirklich in der vom Sprecher geäußerten Weise existiert, spielt im Rahmen der BS/BKM keine Rolle.

Bei der Erörterung der prozessorientierten Konzepte (Abschnitt 7.5.1) wird hierauf noch einmal eingegangen.

2. Inwieweit durch HHI-Konzepte auch solche Aktivitäten berücksichtigt werden sollten, die äquivalent zu einer Äußerung sind (wie z.B. Nicken), wird durch die BS/BKM nicht festgeschrieben. Allerdings ist, wie bereits angedeutet, die Zuverlässigkeit der Kodierung solcher Gesten von der Qualität des Videomaterials abhängig.

3. Obwohl die HHI-Klasse – zumindest vorerst – keine Elemente enthalten soll, mit denen explizit nonverbale Aktivitäten zwischen den Entwicklern konzeptualisiert werden können bzw. sollen, spielen nonverbale Aktivitäten eine nicht unerhebliche Rolle bei der Annotation von HHI-Konzepten. Oft liefern sie nämlich zusätzliche Indizien. So kann z.B. die Tatsache, dass sich ein Entwickler nach seiner Äußerung dem Partner zuwendet, darauf hindeuteten, dass er eine Frage gestellt hat und nun eine Antwort erwartet. Eine solche Beobachtung ist besonders wertvoll, wenn der Frageaspekt nicht eindeutig aus der Äußerung abgelesen werden kann.

[26] Diese Aussage ist nicht ganz richtig, da das Konzept *propose_step* auch dann verwendet werden darf, wenn sich die entsprechende Tätigkeit bereits in Ausführung befindet. Hierauf wird in Abschnitt 7.5.1 genauer eingegangen.

4. Trotz der Tatsache, dass die HHI-Konzepte im Grundsatz nur auf Basis verbaler Phänomene entwickelt wurden, kann es in weiterführenden Studien sinnvoll sein oder sogar notwendig werden, ihren Anwendungsbereich auf bestimmte rein nonverbale Aktivitäten zu erweitern.

6.5.2 HCI/HEI

Neben den Elementen der Klasse HHI enthält die BKM auch solche, die die Interaktionen einer Person mit ihrer sonstigen Umgebung adressieren. Diese Konzepte bilden die zweite große Klasse. Sie wurde, da der Computer bei der PP eine herausragende Rolle spielt, in zwei Teilklassen aufgeteilt:

- *HCI* (*human-computer interaction*) und
- *HEI* (*human-environment interaction*)[27]

Die Klasse HCI umfasst alle Konzepte, die Entwickleraktivitäten adressieren, die sich auf den Computer bzw. auf ihm laufende Programme beziehen und in vielen Fällen unter Einsatz von Maus und/oder Tastatur durchgeführt werden, während die Klasse HEI alle Konzepte bündelt, die sich auf Aktivitäten beziehen, die in Hinsicht auf die sonstige Arbeitsumgebung geschehen. HCI und HEI sind keineswegs disjunkt, da bestimmte Konzepte sowohl HCI- wie auch HEI-Aktivitäten adressieren können. Beispielsweise adressiert das Konzept *explore_sth* sowohl das Durchsehen von unbekanntem Kode, also HCI-Phänomene, wie auch das Durchsehen von Dokumentationen in Papierform, also HEI-Phänome. Aus diesem Grund wird im Weiteren in der Regel auch zusammenfassend von HCI/HEI-Konzepten gesprochen.[28]

6.5.3 Ausblick

Im weiteren Verlauf wird dargestellt,

- welche Konzepte den Klassen HHI, HCI und HEI zugewiesen wurden,

[27] Die Begriffe HHI, HCI und HEI wurden nicht mit Bezug auf gleichnamige Forschungsfelder bzw. Theorien (z.B. im Zusammenhang mit Softwareergonomie), sondern als eigenständige, noch nicht mit Theorie belegte Begriffe ausgewählt und verwendet. Achtung: In wie weit es wirklich angemessen ist von Interaktionen zu sprechen, soll hier nicht weiter diskutiert werden.

[28] Ursprünglich (siehe [182]) waren HCI-Konzepte als diejenige definiert worden, die Aktivitäten adressieren, die sich auf den Computer beziehen *und* unter Einsatz von Maus und/oder Tastatur durchgeführt werden. Dementsprechend handelte es sich bei HCI-Aktivitäten um solche, die vom Driver ausgeführt werden. In weiteren Untersuchungen stellte sich aber heraus, dass diese Festlegung zu kurz greift. So wurden beispielsweise Durchsichten von gerade geschriebenem Kode beobachtet (so genannte *verify_sth*-Phänomene), die ohne Verwendung von Maus und Tastatur auskamen. Z.B. sahen einzelne Driver abschließend noch einmal die gerade von ihnen editierten Zeilen durch. Auf der Ebene der BS/BKM schien es nicht zielführend zu sein, derartige Durchsichten von solchen zu unterscheiden, bei denen beispielsweise mit der Maus „gescrollt“ wird. Dementsprechend wurde die Beschreibung von dem, was im Rahmen der BS unter HCI-Phänomene verstanden werden soll, geändert.

- ob und wie sich die Klassen HHI, HCI und HEI weiter in Unterklassen aufteilen lassen,
- was über diese Klassen hinaus im Rahmen des Blickwinkels erfasst wurde und
- inwieweit die Elemente der Klassen HCI und HEI Kontexte für die Elemente der Klasse HHI bereitstellen (bzw. umgekehrt).

Hierbei ist der letzte Punkt von besonderem Interesse, da die Bedeutung einer Äußerung oft nur in ihrem Kontext, also dem Gesprächsverlauf sowie dem Handlungsumfeld, verstanden werden kann. Nach Grice [85] hängt dies damit zusammen, dass ein Sprecher seine Äußerungen derart formuliert, dass sein Gegenüber diese bei Berücksichtigung des jeweiligen Kontext schon verstehen wird. Dies bedeutet umgekehrt, dass ein und dieselbe Aussage je nach Kontext evtl. unterschiedlich interpretiert werden muss. Ein Kontextverständnis bzw. eine Kontextbeschreibung fördert bzw. steuert also das Verstehen und somit die hier angestrebte Konzeptualisierung. In Abschnitt 9.3 wird hierauf näher eingegangen.

Hinweis zur Begriffsverwendung: Im Rahmen von HHI-, HCI- und HEI-Klassen wird auch von HHI-, HCI- und HEI-Aktivitäten gesprochen, obwohl es sich hierbei in gewisser Weise um einen Pleonasmus, eine verbale Redundanz, handelt. Dies geschieht, damit die Einführung weiterer Abkürzungen – wie HH-Aktivität – sowie Wortungetüme – wie Mensch-zu-Mensch-Aktivität – vermieden werden können.

Exkurs 5: *(Soziale) Interaktion und Kommunikation*

Der Begriff Kommunikation ist derart komplex, dass über unterschiedliche Disziplinen wie z.B. Informatik, Linguistik oder Soziologie hinweg kein Konsens bez. einer exakten Definition besteht [110]. Im Folgenden soll ein kurzer Einblick in unterschiedliche Definitionsversuche gegeben werden, um abschließend festzulegen, was im Rahmen der vorliegenden Arbeit unter Interaktion und Kommunikation verstanden werden soll.

Um sich dem Begriff *soziale Interaktion*, im Weiteren kurz *Interaktion* genannt, anzunähern, eignet sich die Beschreibung von Blickle [19, S. 57], in der von einer nicht notwendigerweise absichtsvollen oder bewussten Einwirkung verschiedener Personen aufeinander gesprochen wird. Dem Terminus *Kommunikation* kann man sich mit einer Definition von Krauss und Fussell [110] nähern, nach der Kommunikation die Übermittlung oder der Austausch von Informationen ist. Theis [206, S. 112] sagt in diesem Zusammenhang, dass im Gegensatz zur Interaktion die „Kommunikation eine wie auch immer geartete Systembildung“ voraussetzt. „Bedeutung, Verstehen oder Informationstransfer sind an das Vorhandensein von sprachlichen und sozialen Strukturen gebunden“. Diese werden häufig als Kommunikationsregeln bezeichnet. Somit stellt zwar jede Kommunikation zwischen Personen eine Interaktion da, aber nicht jede Interaktion ist auch eine Kommunikation [19, S. 58].

Folgt man Nerdinger et al. [151, S. 62], sollte der Begriff Kommunikation aber nur dann verwendet werden, wenn der Austausch von Informationen bewusst erfolgt. In allen anderen Fällen sollte auf den Terminus Interaktion zurückgegriffen werden. Diese Begriffsverwendung widerspricht der von Watzlawick et al. [214], welche die Termini Verhalten und Kommunikation im Rahmen ihrer psychologisch orientierten Beschreibungen von Kommunikation gleichbedeutend verwenden (S. 23) und dabei, zumindest was die Definition der Begrifflichkeiten angeht, nicht zwischen absichtsvollen und absichtslosen oder bewussten und unbewussten Handeln unterscheiden (S. 52).[a]

Darüber hinaus wird in der Regel zwischen *verbaler* und *nonverbaler* Kommunikation differenziert, wobei zur verbalen Kommunikation Lautsprache, Gebärdensprache und Schriftsprache zählen, während unter nonverbaler Kommunikation das Übermitteln von „Botschaften durch Mimik, Gestik, Körperhaltung und auch durch die Modulation der Stimme [151]“ verstanden wird.

Im Rahmen der vorliegenden Arbeit soll nun Folgendes gelten: Im Großen und Ganzen wird den Begriffsbildungen von Blickle bzw. Krauss und Fussel sowie Theis gefolgt. Der Begriff Interaktion soll aber nur im Zusammenhang mit direkt an den Partner gerichteten Aktivitäten verwendet werden. Indirekte Einwirkungen aufeinander werden hingegen – zumindest vorläufig – nicht als Interaktionen zwischen Personen bezeichnet, weil davon ausgegangen wird, dass sie in der Regel wesentlich schwieriger identifiziert werden können. Außerdem wird der Begriff Interaktion im Zusammenhang mit Einwirkungen einer Person auf die sonstige Umgebung verwendet. Bei der Benutzung des Begriffes Kommunikation wird nicht explizit zwischen bewussten oder unbewussten Akten unterschieden. Bisher wurde allerdings nur die verbale, und somit im Allgemeinen bewusste Kommunikation, in Form von *verbalen Äußerungen*, im Weiteren kurz *Äußerungen* genannt, adressiert. So enthält die Konzeptklasse HHI ausschließlich Konzepte, die auf Basis der Lautsprache entwickelt wurden. Dies schließt aber nicht aus, dass diese Konzepte in Zukunft auch für andere Kommunikationsformen eingesetzt werden können. Den einfachsten denkbaren Fall bilden hier die *lexikalisierten Gesten*, also die, die wie Elemente einer Lautsprache funktionieren.

---

[a] Für Watzlawick et. al. [214, S. 51] hat „alles Verhalten in einer zwischenmenschlichen Situation Mitteilungscharakter“. Sie schließen daraus, dass, da man sich nicht *nicht* verhalten kann, man auch nicht *nicht* kommunizieren kann.

Kapitel 7

Die HHI-Konzepte

What's in a name? That which we call a rose
By any other name would smell as sweet.
Shakespeare, Romeo and Juliet (II, ii, 1-2)

Im vorliegenden Kapitel werden die HHI-Konzepte, also diejenigen, mit denen verbale Interaktionen oder Aktionen beschrieben werden können, detailliert erklärt. Hierbei geht es vor allem darum, sie für Analysen konkreter Prozessaspekte praktisch nutzbar zu machen. Um dieses Ziel zu erreichen, werden die aus den Daten gewonnenen Erkenntnisse sowohl auf konzeptioneller Ebene erläutert wie auch durch vielfältige Phänomenbeschreibungen bebildert und somit „greifbar" gemacht. Um den Einstieg zu erleichtern, werden zuerst die extrahierten Objekte und Verben, also diejenigen Teile, aus denen sich die Namen der HHI-Konzepte zusammensetzen, in Übersicht dargestellt (Abschnitt 7.1). Nachfolgend werden zwei zentrale, den Umgang mit der BS/BKM prägende Prinzipien erläutert (Abschnitt 7.2). In den Abschnitten 7.3 bis 7.9 folgen dann die Beschreibungen der einzelnen in den Daten gegründeten Konzepte sowie Erläuterungen zu deren Verwendung. Hierbei werden sowohl Identifikationskriterien für einzelne Konzepte wie auch Zusammenhänge und Abgrenzungen zwischen verwandten Konzepten und Konzeptklassen erläutert. Im Sinne der GTM werden nur Konzepte berücksichtigt, für die tatsächlich Phänomene beobachtet werden konnten, und nicht etwa auch solche, deren Existenz (z.B. anhand bereits entdeckter Teile) naheliegend erscheint. Außerdem werden die einzelnen Konzepte nur soweit beschrieben, wie es anhand der jeweils identifizierten Indikatoren (siehe S. 49) möglich ist. Darüber hinaus wird erörtert, dass im Laufe der Analyse auch Relationen zwischen einzelnen Konzepten oder Konzeptklassen entdeckt wurden, die Untersuchung der Basisaktivitäten also, ohne dass es explizit (z.B. durch den Blickwinkel) gefordert war, eine partielle Struktur auf der Menge der Konzepte hervorgebracht hat.

Eine erste, noch nicht im Detail ausgearbeitete Version der HHI-Konzepte wurde 2008 auf dem Workshop der *Psychology of Programming Interest Group (PPIG)* vorgestellt [182].[1] Die hier dargestellte praktisch anwendbare Form wurde danach ausgearbeitet.

Achtung: In diesem Kapitel wird der Einfachheit halber auch dann von BKM bzw. BS/BKM oder BS gesprochen, wenn nur HHI-Konzepte adressiert oder betrachtet werden.

[1] Die erste Version enthielt vier HHI-Konzepte weniger. Die schon vorhandenen Konzepte waren bei weitem nicht so detailliert ausgearbeitet, wie im vorliegenden Dokument.

7.1 Objekte und Verben der HHI-Konzepte

In Abschnitt 5.2.2 wurde erläutert, warum es günstig ist, eine Syntax für Konzeptnamen zu besitzen. Nachfolgend wurde eine solche für Aktivitäten im Rahmen der PP, also für die Elemente der BKM, festgelegt. Sie schreibt im Wesentlichen die Verwendung eines Objektes und eines Verbes fest.

Die Verwendung dieser Syntax im Rahmen der hier beschriebenen Analysen führte dazu, dass bez. der verbalen Kommunikation 16 Objekte (die so genannten *HHI-Objekte*) und 13 Verben (die so genannten *HHI-Verben*) aus den Daten extrahiert wurden.[2] Sie werden in Tabelle 7.1 bzw. Tabelle 7.2 erläutert und bilden neben den Klassen HHI, HCI und HEI die zentralen Konzeptklassen der BKM (*Objektklassen* und *Verbklassen*), das heißt Mengen, die bez. bestimmter Aspekte gleichartige Konzepte enthalten. Die Namen der Objekte (oder auch Verben) werden somit im Weiteren sowohl dafür verwendet

- einen betrachteten Gegenstand (bzw. eine betrachtete Handlung) zu bezeichnen, wie auch dafür,

- eine Klasse von Konzepten zu benennen.

So bezeichnet beispielsweise *knowledge* einerseits eine bestimmte Form von Wissen und andererseits die Menge der Konzepte *explain_knowledge*, *agree_knowledge*, *disagree_knowledge*, *challenge_knowledge* und *ask_knowledge*.

Achtung: Die Tabellen 7.1 und 7.2 fassen nur die Kerncharakteristika der Objekte bzw. Verben zusammen. Weitere Eigenschaften, speziell solche, die der Abgrenzung der Konzepte gegeneinander dienen, werden erst im nächsten Abschnitt anhand der Basiskonzepte erläutert. Fünf zentrale, im Rahmen der Analysen der Videos herausgearbeitete, objekt- bzw. verbübergreifende Eigenschaften sollen aber schon vorweg erwähnt werden:

1. Bei weitem nicht alle Kombinationen aus Objekten und Verben finden sich in der BKM in Form von Basiskonzepten wieder. Dies hat vier Gründe:

   (a) Nicht jedes Verb kann sinnvoll mit jedem Objekt zu einem Konzept zusammengefügt werden. So ist es beispielsweise nicht möglich, ein *propose* bez. einer *activity* durchzuführen, da mit dem Begriff *activity* nur aktuell ablaufende und nicht etwa auch zukünftige Aktivitäten bezeichnet werden.

[2] Die HHI-Objekte und HHI-Verben wurden nicht separat, sondern ausschließlich in Form von Verb-Objekt-Paaren, also als Konzepte aus den Daten abgeleitet. Es wurde darauf geachtet, dass bereits festgelegte Objekte und Verben wiederverwendet werden, sofern dies möglich war. Dies hatte zur Folge, dass die Beschreibungen der Objekte und Verben kontinuierlich erweitert, präzisiert oder auch geändert werden mussten (siehe auch Abschnitt „Ständiges Vergleichen“ auf S. 54f).

Objekt	Beschreibung
activity	Eine gerade ablaufende HCI- oder HEI-Aktivität.
completion	Der Grad der Fertigstellung eines Arbeitsschritts (*step*). Vgl. mit *state*.
design	Eine Gestaltungsoption bez. des gerade in Entwicklung befindlichen Programmkodes. Kann sich auf einzelne Elemente oder die Struktur beziehen.
finding	Eine gerade gewonnene und verbalisierte Einsicht einer Person.[a] Indikator für eine Wissenserweiterung auf Basis eines kognitiven Prozesses. Dieser Prozess kann beispielsweise durch eine Beobachtung angestoßen worden sein. Vgl. auch mit *knowledge* und *standard of knowledge*.
gap in knowledge	Das Fehlen von bestimmtem Wissen bei *beiden* Paarmitgliedern. Vgl. mit *standard of knowledge*.
hypothesis	Eine Hypothese oder Vermutung, oft bez. Eigenschaften des in Entwicklung befindlichen Programms.
knowledge	Explizites Wissen (siehe Abschnitt 7.1.1), bei welchem es sich weder um bestimmtes Metawissen (siehe *standard of knowledge* und *gap in knowledge*) noch um eine gerade gewonnene Einsicht[b] (siehe *finding*), eine Hypothese (siehe *hypothesis*) oder in Vorschläge zum Produkt oder Prozess „verpacktes“ Wissen handelt. Wird vom Sprecher als wahr angenommen.
off topic	Alles, was nicht mit der Lösung der Aufgabe bzw. dem Lösungsprozess zu tun hat.
requirement	Eine vorgegebene oder vermutete Anforderung oder Voraussetzung bez. der zu entwickelnden Lösung.
something/sth	Nicht näher spezifiziertes oder spezifizierbares Objekt. Wird im HHI-Umfeld nur im Zusammenhang mit dem Verb *mumble* eingesetzt.
source of information	Wissen über (das Vorhandensein von) Dokumentation, Quellkode, Webseiten usw. Wurde explizit von *knowledge* separiert.
standard of knowledge	Wissensstand einer Person bez. eines bestimmten Themas. Achtung: Eine bei beiden Entwicklern gleichermaßen existierende und vom Paar als solche identifizierte Wissenslücke wird als *gap in knowledge* bezeichnet.
state	Grad, zu dem eine Strategie (*strategy*) abgearbeitet ist. Vgl. mit *completion*.
step	Ein potentieller nächster Schritt im Arbeitsprozess; vom Akteur als atomare Einheit taktischen Verhaltens betrachtet (ein sog. Arbeitsschritt). Vgl. mit *strategy*.
strategy	Ein längerfristiger Arbeitsplan zur Lösung einer (Teil-)Aufgabe, der noch nicht (vollständig) abgearbeitet ist. Strategien beinhalten in der Regel mehrere Arbeitsschritte.
todo	Eine Teilaufgabe oder ein Arbeitsschritt, der nicht aktuell, sondern in der Zukunft durchgeführt werden soll. Hierbei adressiert Zukunft sowohl die aktuelle PP-Sitzung wie auch nachfolgende Programmiersitzungen.

[a] Welche Indizien für das Erkennen einer Einsicht zulässig sein sollen, wird im Rahmen der detaillierten Erörterungen der Konzeptklasse *finding* (ab S. 250) erläutert.

[b] Ausnahmen hiervon werden später diskutiert (siehe z.B. Abbildung 7.1).

Tab. 7.1: Grobe Beschreibung der Objektbestandteile der in den Daten identifizierten HHI-Konzepte. Die Objekte adressieren im Rahmen der BS nur solche Entitäten, die von Entwicklern (allerdings nicht unbedingt bewusst) als derartige verbalisiert werden.

Verb	Beschreibung
amend	Eine vorausgegangene Äußerung (HHI), z.B. einen Vorschlag (siehe *propose*) bzw. eine unmittelbar vorausgegangene oder gerade ablaufende Aktivität (HCI/HEI) erweitern, ergänzen bzw. detaillieren. Im Grundsatz wird der ursprünglichen Äußerung bzw. Aktivität dabei aber zugestimmt.
ask	Eine (meist offene, manchmal auch geschlossene) Frage stellen.
agree	Einer vorausgegangenen Äußerung (HHI), z.B. einem einzelnen Vorschlag, bzw. einer unmittelbar vorausgegangenen oder gerade ablaufenden Aktivität (HCI/HEI) ohne weitere Ergänzungen zustimmen. Vgl. mit *decide*.
challenge	Eine vorausgegangene Äußerung (HHI) bzw. eine unmittelbar vorausgegangene oder gerade ablaufende Aktivität (HCI/HEI) ablehnen bzw. missbilligen und einen Gegenvorschlag machen. Vgl. mit *disagree*.
decide	Eine Auswahl aus einer Menge von Vorschlägen (siehe z.B. *propose*) treffen. Vgl. mit *agree*.
disagree	Eine vorausgegangene Äußerung (HHI) bzw. eine unmittelbar vorausgegangene oder gerade ablaufende Aktivität (HCI/HEI) missbilligen, ohne einen konkreten Gegenvorschlag zu machen. Vgl. mit *challenge*.
explain	Dem Partner einen Sachverhalt[a] erklären. Eine solche Erklärung kann sich, muss sich aber nicht, auf eine offene oder geschlossene Frage beziehen (siehe *ask*). Erklärungen können vielmehr aus eigenem Antrieb des Sprechers heraus formuliert werden.
mumble	Eine akustisch unverständliche oder stark fragmentarische, nicht mehr interpretierbare Äußerung machen. Wird nur im Zusammenhang mit dem Objekt *sth* verwendet.
propose	Einen Vorschlag machen, der nicht Bezug auf einen anderen Vorschlag nimmt. In der Regel handelt es sich dabei um eine neue, bisher nicht geäußerte bzw. diskutierte Absicht. Diese kann mehrere Alternativen enthalten. Auf einen solchen Vorschlag kann z.B. mit *agree* (bei Vorschlägen ohne Alternativen), *decide* (bei Vorschlägen mit Alternativen), *challenge*, *disagree* oder *amend* reagiert werden.
remember	Sich erkennbar an etwas Spezifisches erinnern.
say	Etwas sagen. Wird nur im Zusammenhang mit dem Objekt *off topic* verwendet.
stop	Vorschlagen, eine Aktivität (HCI/HEI) zu beenden bzw. abzubrechen.
think aloud	Verbalisierung der eigenen Aktivitäten (HCI/HEI) und/oder der mit ihnen zusammenhängenden Gedanken.

[a] Im Gegensatz zu dem Begriff Tatsache, soll der Begriff Sachverhalt hier Aussagen unabhängig von ihrer Überprüfung adressieren.

Tab. 7.2: Grobe Beschreibung der Verbbestandteile der in den Daten identifizierten HHI-Konzepte. Im Rahmen der BKM ist jedes dieser Verben nur für die Verwendung im Zusammenhang mit verbalen Äußerungen und nicht in Bezug auf andere Interaktionsformen oder Vorgänge definiert. Dies gilt auch für das Verb *remember*, welches nicht einen bestimmten kognitiven Prozess (Erinnern), sondern die Verbalisierung des Ergebnisses eines bestimmten kognitiven Prozesses adressiert. Viele der Verben – aber nicht alle (z.B. *mumble*) – beziehen sich in gewisser Weise auf den illokutionären Akt von Äußerungen.

(b) Bestimmte Kombinationen von Objekten und Verben führen zu Konzepten, die sich inhaltlich mit anderen überschneiden. Um dies zu vermeiden, wurde ein Reihe von Regeln definiert. Beispielsweise gibt es das Konzept *explain_step* in der BKM deshalb nicht, da im Verlauf der Konzeptgenese festgelegt wurde, dass der Transfer von Wissen immer über das Objekt (bzw. die Konzeptklasse) *knowledge* abgebildet werden soll.[3]

(c) Für manche Verben (bzw. Objekte) wurde festgelegt, dass sie ausschließlich in Zusammenhang mit bestimmten Objekten (bzw. Verben) verwendet werden dürfen. So dient das Verb *mumble* einzig dazu, undeutliche Aussagen, d.h. insbesondere Aussagen, deren konkretes Bezugsobjekt (durch den Untersuchenden) nicht wirklich identifiziert werden kann, zu markieren. Es wird dementsprechend nur zusammen mit dem unspezifischen Objekt *something* verwendet (was im Rahmen der HHI-Konzepte auch umgekehrt gilt: *something* wird nur zusammen mit *mumble* verwendet).

(d) Viele Kombinationen sind zwar theoretisch vorstellbar und würden auch nicht zu inhaltlichen Überschneidungen führen, wurden aber bisher nicht in den Daten identifiziert (z.B. *disagree_todo*).

2. Mit vielen der Verben der BKM wird in gewisser Weise das adressiert, was in der Sprechakttheorie als illokutionärer (bzw. illokutiver) Akt bezeichnet wird (siehe Exkurs 6 auf S. 146).[4] So markiert *ask* beispielsweise eine Illokution vom Typ Frage und *propose* eine vom Typ Vorschlag.[5]

   Bei der Analyse sollte der Forscher also in vielen Fällen versuchen, die Illokutionen von Äußerungen zu verstehen. Hierzu dienen ihm Indikatoren wie z.B. Satzstellung, Modalpartikel[6], performative Verben oder Intonation. Außerdem liefert natürlich der Kontext, die so genannte „kommunikative Situation“, in der eine Äußerung gemacht wird, hilfreiche Informationen.

[3] Diese Aussage ist stark vereinfachend. Denn genau genommen gibt es neben den Elementen der Klasse *knowledge* auch noch andere Konzepte, um Wissenstransfer zu erfassen. Dies hängt mit dem Wissensbegriff zusammen, der im Rahmen der BS verwendet wird. In Abschnitt 7.1.1 wird hierauf genauer eingegangen.

[4] Bei den hier geschilderten Untersuchungen spielte die Sprechakttheorie keine Rolle. Erst gegen Ende der Untersuchungen wurde die BS/BKM diesbezüglich betrachtet. Zu diesem Zeitpunkt waren aber bereits alle Konzepte vollständig ausgearbeitet. Warum die Verben der BKM illokutionäre Akte nur „in gewisser Weise“ adressieren, wird erst in Abschnitt 9.1.1 erörtert.

[5] Die Verwendung von Verben mit bestimmten Objekten kann dazu führen, dass die adressierte Illokution eingeschränkt wird. Diese Einschränkung hat nicht unbedingt etwas mit dem von der Äußerung adressierten Thema, also dem zuzuweisenden Objekt zu tun. So adressiert beispielsweise *ask_design* nur noch bestimmte Typen von Fragen, nämliche solche, die keine eigenen Ideen enthalten.

[6] In der Linguistik bezeichnet der Begriff Modalpartikel nicht flektierbare Wörter, die ein Sprecher dazu verwendet, um seine Haltung zu einer Aussage zu verdeutlichen. Z.B. wird das Wort „ja“ auch dazu verwendet, um die Erwartung auszudrücken, dass das Gegenüber vielleicht schon über die geäußerte Information verfügt (siehe z.B. Äußerung 2 in Tabelle 7.6).

In den nachfolgenden Beschreibungen der einzelnen Konzepte wird vor allem auf situationsspezifische Indizien, d.h. solche, die aus dem Kontext kommen, eingegangen. Allgemeine, wie z.B. die Verwendung von Modalverben oder Intonation, werden hingegen nur in Ausnahmefällen erörtert. Es wird davon ausgegangen, dass diese in der Regel auch so angemessen gedeutet werden können.

3. In der Regel ist der Bedeutungsumfang der einzelnen im Rahmen der BKM zum Einsatz kommenden Verben (und zum Teil auch Objekte), das heißt die Menge der von der BKM jeweils adressierten Bedeutungen, breit angelegt. Hierdurch wird es möglich, kurze Bezeichnungen für komplexere/vielfältige Phänomene zu erhalten. So adressiert beispielsweise das Verb *propose* nicht nur Vorschläge im Sinne von Anregungen oder Angeboten, sondern auch Anweisungen. Der Bedeutungsumfang wurde aber nicht vordefiniert, sondern ergab sich aus Erkenntnissen bei den Untersuchungen – z.B. hinsichtlich einer geeigneten Granularität – und wird bei der Erörterung der einzelnen Konzepte näher erläutert (siehe z.B. Tabelle 7.5). Dabei wurde darauf geachtet, dass die verwendeten Begriffe nicht kontraintuitiv eingesetzt werden.

4. Die Klasse HHI verwendet drei Typen von Verben:[7]

   (a) **Initiative Verben:** Mit einer *initiativen Äußerung* wird ein neuer Punkt bzw. ein bisher nicht diskutierter Aspekt in den Diskurs eingebracht, z.B. ein Vorschlag zur Gestaltung eines (bisher nicht betrachteten) Teils des Kodes. Es wird nicht, zumindest nicht direkt, Bezug auf den Inhalt vorhergegangener Äußerungen genommen. Es handelt sich also nicht um eine Erwiderung, wenn auch der Themenkreis, z.B. die Implementierung einer bestimmten Methode, durch die Äußerung nicht unbedingt gewechselt werden muss. Um solche Äußerungen zu konzeptualisieren, werden von der BKM drei *initiative Verben* verwendet: *propose*, *remember* und *stop*.[8]

   (b) **Reaktive Verben:** Äußerungen, die konkret Bezug auf eine oder mehrere andere Äußerungen nehmen, z.B. Zustimmungen, Widersprüche oder auch Verbesserungen bzw. Detaillierungen, werden *reaktiv* genannt. Für ihre Konzeptualisierung kommen die folgenden *reaktiven Verben* zum Einsatz:[9] *agree*, *amend*, *challenge*, *decide* und *disagree*.

   (c) **Bivalente Verben:** Es gibt Verben, die sowohl initiative wie auch reaktive Äußerungen adressieren können. Dies sind *explain*, *think aloud*

[7] Die Verben *mumble* und *say* werden hier nicht einbezogen, da sie Sonderrollen einnehmen.

[8] Bisher wurden nur sehr wenige *stop*-Phänomene beobachtet. Evtl. kann es in Zukunft notwendig werden, *stop* einer anderen Verbklasse zuzuordnen.

[9] Diese Verben können sich auch auf Äußerungen zu sonstigen Aktivitäten beziehen. Dies aber nur im Zusammenhang mit der Klasse *activity*. In Abbildung 7.3 wird hierauf genauer eingegangen.

und in gewisser Weise auch *ask*. Beispielsweise kann Wissenstransfer sowohl aufgrund einer Frage wie auch aus Eigenantrieb des Sprechers erfolgen. Beide Formen werden mit *explain* konzeptualisiert.

Konzepte, deren Namen initiative, reaktive oder bivalente Verben enthalten, werden auch **initiative, reaktive** bzw. **bivalente Konzepte** genannt. Im weiteren Verlauf wird demonstriert werden, dass die BS derart konstruiert ist, dass Eigenschaften von initiativen Konzepten eine hervorgehobene Rolle spielen. Sie (bzw. genau genommen ihr Auftreten in Phänomenen) bestimmen in vielen Fällen das bei der Kodierung von verbalen Reaktionen zu verwendende Objekt. So wird beispielsweise eine Ergänzung zu einer vom Partner initiativ geäußerten Erkenntnis in der Regel auch dann als *amend_finding* kodiert, wenn sie selbst nicht auf einer Erkenntnis beruht (nähere Erläuterungen hierzu finden sich auf S. 335).

5. Die Verben der HHI-Konzepte lassen sich darüber hinaus noch auf eine zweite Weise typisieren:[10]

   (a) **Konstruktive Verben:** Zu diesen zählen *amend*, *challenge*, *explain*, *propose* und *remember*. Sie adressieren Äußerungen, in denen in konstruktiver Weise weitere Inhalte/Aspekte in die Diskussion eingebracht werden.

   (b) **Nicht konstruktive Verben:** Dies sind *ask*, *agree*, *decide* und *disagree*. Sie adressieren Äußerungen, in denen keine weiteren Inhalte/Aspekte in konstruktiver Weise in die Diskussion eingebracht werden, sondern Inhalte/Aspekte angefordert oder bereits eingebrachte beurteilt werden.[11]

   Konzept, deren Namen konstruktive bzw. nicht konstruktive Verben enthalten, werden auch **konstruktive** bzw. **nicht konstruktive Konzepte** genannt, Äußerungen dementsprechend **konstruktive** bzw. **nicht konstruktive Äußerungen**.

Bevor nun die HHI-Konzepte im Einzelnen detailliert dargestellt werden, sollen vier, im Laufe des GTM-Prozesses herausgearbeitete Abgrenzungen zwischen Verben bzw. Objekten einführend erläutert werden:

– *knowledge* vs. *finding* vs. *standard of knowledge* vs. andere Objekte

– *propose* vs. *explain*

[10] Hier werden neben den Verben *mumble* und *say* auch *think_aloud* und *stop* nicht miteinbezogen. Sie nehmen eine Sonderrolle ein.

[11] Durch das Stellen von Fragen (*ask*) können sicherlich in gewisser Weise auch Inhalte/Aspekte in die Diskussion eingebracht werden. Dies geschieht aber in der Regel nicht auf eine Art, die man als konstruktiv bezeichnen würde.

Exkurs 6: *Sprechakttheorie/Illokution/Indirekte Sprechakte*

In der Sprechakttheorie werden nicht einzelne Symbole, Sätze oder Wörter als Grundelemente der sprachlichen Kommunikation angesehen, sondern *sprachliche Handlungen*, so genannte *Akte*, also die Produktion oder Hervorbringung von Symbolen, Wörtern oder Sätzen unter bestimmten Bedingungen [189, S. 30]. Es wird die Auffassung vertreten, dass sprachliche Äußerungen für sich genommen bereits Handlungen sind, die die Realität verändern. Hierzu wird grob zwischen einem *Äußerungsakt*, der „physischen Aktivität einer Person, bei der sie phonische oder graphische Ereignisse produziert“, und der „Interpretation dieser Aktivität relativ zu einem bestimmten Sprachsystem, einem bestimmten Handlungssystem und zur sozialen Situation, in die Äußerer und Wahrnehmer eingeschlossen sind“ unterschieden [226, S. 51]. Nach der zentralen Hypothese der Sprechakttheorie bedeutet Sprechen, „in Übereinstimmung mit Regeln Akte zu vollziehen“ [189, S. 38].

John Searle [189] beschreibt vier Teilakte[a], die er unter dem Oberbegriff *Sprechakt* (im Weiteren nach [95, S. 236] auch *Sprechhandlungen* genannt) zusammenfasst: neben dem Äußerungsakt und dem so genannten propositionalen und perlokutionären Akt (werden erst in Exkurs 13 erläutert) den auf Austin zurückgehenden *illokutionären Akt* [7, S. 117]. „Die Illokution eines Spechaktes[b] drückt aus, in welcher interaktionalen Funktion [die dargestellte Wirklichkeit] im Sprechakt thematisiert wird“ [226, S. 26]. Vereinfacht ausgedrückt [95, S. 238]: Der illokutionäre Akt bezeichnet das, was man mit einer Äußerung tut. Hierbei kann es sich um Behaupten, Fragen, Befehlen, Versprechen etc. handeln [189, S. 40]. Folgende Beispiele sollen den Begriff verdeutlichen [225, S. 336]: *a*) „Wie spät ist es?“ → Frage; *b*) „Ich kann dir versichern, dass ich es nicht gewesen bin.“ → Versicherung; *c*) „Warum hast Du das getan?“ → Vorwurf[c]

Bestimmten Verben, wie z.B. „versichern“, wird in der Sprechakttheorie eine besondere Bedeutung beigemessen, da sie die Art der Handlung explizieren können. Sie werden *performative Verben* genannt. Dementsprechend spricht man von *explizit performativen Sprechakten*, wenn ein performatives Verb verwendet wird und von *implizit performativen Sprechakten*, wenn kein solches performatives Verb vorkommt, die Äußerung sich aber in eine explizit performative paraphrasieren lässt (z.B. „Ich bin es nicht gewesen“).

Darüber hinaus unterscheidet Searle zwischen direkten und *indirekten Sprechakten* [190]. Ein Sprechakt wird genau dann als indirekt klassifiziert, wenn der wörtlich indizierte illokutionäre Akt (*sekundärer illokutionärer Akt*) vom intendierten nicht-wörtlichen illokutionären Akt (*primärer illokutionärer Akt*) abweicht [53]. So ist die Äußerung „Wollen wir einen Kaffee trinken?“ sekundär eine Frage, aber primär ein Vorschlag. Die Antwort „Ich muss jetzt zur Vorlesung“ ist sekundär eine Feststellung und primär eine Ablehnung.

---

[a] Der Begriff „Akt“ ist etwas unglücklich gewählt, denn es „handelt sich nicht um nebeneinander ausgeführte Handlungen, sondern um die verschiedenen (und zwar natürlich auf einer theoretischen Ebene differenzierten) Aspekte nur einer Handlung“ [225, S. 324].

[b] Mit Illokution wird oft die vollzogene illokutionäre Handlung bezeichnet. Im Rahmen der vorliegenden Arbeit wird Illokution und illokutionärer Akt allerdings synonym verwendet.

[c] Beispiel dafür, dass „der Typ einer Sprechhandlung [...] nicht ausschließlich durch Form und Inhalt des Äußerungsresultates bestimmt [ist], sondern in eingeschränkten Kontexten außerdem durch (a) institutionsspezifische Erwartungen und Handlungsobligationen [oder] (b) personenspezifische Einschätzungen der kommunikativen Situation“ [225, S. 336].

– *explain* vs. *think aloud*

– *disagree+propose* vs. *challenge*

Dies geschieht in den folgenden Unterabschnitten. Dem Leser soll durch diese Darstellungen ermöglicht werden, ein erstes konzeptklassenübergreifendes Verständnis der BS/BKM, im Speziellen der HHI-Konzepte zu erlangen. Ein Hauptschwerpunkt liegt hierbei auf Erörterungen des sich im Rahmen der Herleitung der BS/BKM herausgebildeten Wissensbegriffs. Darüber hinausgehende, detailliertere Darstellungen des Wissensbegriffs sind in Abschnitt 7.6 zu finden.

7.1.1 *knowledge* vs. *finding* vs. *standard of knowledge* vs. andere Objekte

Wie bereits erwähnt (siehe z.B. Abschnitt 6.3) ist Wissenstransfer (Übermittlung von Projektwissen, technischem Know-How etc.) ein häufig diskutierter Aspekt der PP. Die BKM bildet ihn unter anderem (allerdings bei weitem nicht ausschließlich) über die Klassen *knowledge*, *finding* und *standard of knowledge* ab. Hierbei gelten folgende, sich im Laufe der Analyse herausgebildete Festlegungen:[12]

1. **Wissensbegriff der BS:** Von einem groben, umgangssprachlichen und im Detail nicht näher explizierten Verständnis des Begriffs Wissen als Kenntnis und Verständnis von Fakten, Wahrheiten und Informationen ausgehend, stellte sich schon zu Beginn der Untersuchungen heraus, dass aufgrund des behavioristischen Ansatzes nur solches Wissen berücksichtigt werden kann, welches verbal, in Ausnahmefällen unter Zuhilfenahme von Skizzen und Gesten, also über Worte und evtl. Bilder kommunizierbar ist.[13] Dies bedeutet, dass die BS nur das so genannte explizite Wissen (*explicit knowledge* [169]) berücksichtigt, welches als dasjenige definiert ist, über welches der Sprecher bewusst verfügen[14] und welches er (sprachlich) übermitteln kann (das Gegenstück wird implizites Wissen genannt (siehe Exkurs 7)). Sie unterscheidet hierbei nicht zwischen wahren und nur als wahr angenommenen,

[12] Um die wissensspezifischen Konzepte bzw. ihre Objektbestandteile besser beschreiben zu können, werden in den Darstellungen auch externe Begriffsbildungen und Theorien herangezogen. Diese hatten aber keinen Einfluss auf die Herleitung der Konzepte bzw. Objekte, da sie erst nach der Fertigstellung der hier dargestellten Form der BKM eruiert wurden.

[13] Genau genommen wäre es sogar möglich an dieser Stelle von dem Wissen zu sprechen, welches in den untersuchten bzw. zu untersuchenden Sitzungen tatsächlich kommuniziert wird. Wobei die Kommunikation nicht unbedingt wirklich verbal stattfinden, sondern nur verbal (unter Zuhilfenahme von Skizzen und Gesten) möglich sein muss. So könnte ein Driver ein Tastenkürzel (*keyboard shortcut*) benutzen und somit das Wissen über dieses zum Ausdruck bringen.

[14] An dieser Stelle muss zwischen „bewusst verfügen können“ und „bewusst übermitteln“ unterschieden werden. Im Rahmen der BS wird auch dann von Wissen gesprochen, wenn die verbale Übermittlung desselben an den Partner unbewusst erfolgt, z.B. im Rahmen eines Vorschlags.

evtl. also auch falschen Äußerungen bzw. Aussagen. Der Hauptgrund hierfür besteht darin, dass es in vielen Fällen schwierig, bei PP-Sitzungen aus dem professionellen Umfeld oft sogar unmöglich ist, eine diesbezügliche Unterscheidung durchzuführen, da dem Forschenden der Kontext oder die optimale Lösung bzw. der optimale Lösungsweg bestenfalls eingeschränkt bekannt ist. Ohne solche Kenntnisse ist eine Differenzierung zwischen falscher oder auch wahrer Meinung[15] und Wissen, im Sinne von wahrer, gerechtfertigter Meinung[16] (W_{wgM}) aber in der Regel kaum praktikabel.

Das auf diese Weise grob charakterisierte Wissen einer Person (zu einem Zeitpunkt) sei im Weiteren mit W_{BS}^{+} bezeichnet (auf weitere Details wird in Abschnitt 7.6 eingegangen). Es referenziert das im Rahmen der BS-Herleitung *potentiell überhaupt adressierbare/referenzierbare Wissen* dieser Person (nicht zu verwechseln mit dem letztendlich wirklich von der BS/BKM adressierten Wissen). Aufgrund von im Laufe des Herleitungsprozesses der BS erlangten Erkenntnissen und getroffenen Entscheidungen wurde festgelegt, dass nur ein Teil der Verbalisierungen dieses Wissens tatsächlich dafür in Frage kommen soll durch einzelne Klassen bzw. Konzepte der BKM, evtl. sogar die primäre Illokution (siehe Exkurs 6) hervorhebend, adressiert zu werden. Das diesbezügliche, bis zu einem bestimmten Zeitpunkt geäußerte Wissen einer Person wird im Weiteren als W_{BS}^{-} bezeichnet (genauso wie W_{BS}^{+} darf es nicht mit dem Wissen verwechselt werden, welches letztendlich wirklich explizit adressiert wird).[17] Auf W_{BS}^{-} wird erst in Abschnitt 7.6 genauer eingegangen. Um einen ersten Eindruck zu bekommen seien hier aber schon einmal zwei Beispiele vorgestellt: Im Rahmen der Analysen wurde festgelegt, dass Wissen über elementare Methoden der Softwa-

[15] Unter einer Meinung soll hier nur grob folgendes verstanden werden: „Ansicht, persönliches Urteil über etw., jmdn., Standpunkt in Bezug auf etw., jmdn." (Digitales Wörterbuch der deutschen Sprache (DWDS): http://www.dwds.de/?kompakt=1&qu=Meinung (Abruf: 25.01.2012)).

[16] Traditionell wurde Wissen oft als wahre, gerechtfertigte Meinung definiert (siehe auch Platons Theätet [165]). Im erkenntnistheoretischen Umfeld wird diese Definition allerdings als nicht hinreichend angesehen (siehe z.B. [71]). Wie in Abschnitt 7.6 diskutiert werden wird, gibt es aber bisher keine allgemein akzeptierte Definition von dem, was als „Wissen" bezeichnet werden soll. Es existieren vielmehr unterschiedlichste Definitionen, die sich zum Teil wesentlich von der klassischen, schon von Platon selbst hinterfragten Begriffsbildung unterscheiden. So ist nach Polanyi [168, S. 113] Wahrheit Wahrhaftigkeit, „das bedeutet, im Sprechen drückt der Sprecher seine Sicht der Dinge aus, seine Realität [109, S. 64]". Diese Wahrhaftigkeit ist es, die mit den Elementen der BKM adressiert werden soll. Eine Untersuchung ihres Verhältnisses zur Realität, z.B. ob es sich in Hinsicht auf den Prozesserfolg um einen Irrtum handelt, geht hingegen über das hinaus, was mit der BS bzw. BKM alleine geleistet werden soll und kann, sondern ist eine der Aufgaben die in höheren Schichten der Analyse zur Aufschlüsselung des Paarprogrammierungsprozesses zu lösen sein wird.

[17] Dass hier nur das tatsächlich verbalisierte Wissen adressiert wird, schließt eine explizite Berücksichtigung von nur nonverbal kommuniziertem Wissen – wie es in Fußnote 13 beispielhaft erläutert wurde – aus. Der Hauptgrund hierfür besteht darin, dass bei rein nonverbaler Kommunikation von Wissen meist kaum zu ermitteln ist, ob es sich wirklich um eine Weitergabe handelt. Weiterführende Untersuchungen müssen diese Beschränkung hinterfragen.

***Exkurs 7:** Implizites Wissen*

Dem expliziten, „in Worten und Zahlen [ausdrückbaren] und problemlos mit Hilfe von Daten, wissenschaftlichen Formeln, festgelegten Verfahrensweisen oder universellen Prinzipien [mitteilbaren]“ [153, S. 18] Wissen steht das implizite, auch *tacit knowledge* (nach Michael Polanyi [169] auch *tacit knowing*) genannte, nicht verbalisierbar im Können steckende Wissen gegenüber: „we can know more than we can tell“ [169]. Es zeigt „sich besonders plastisch und einfach nachvollziehbar in zwei Situationen, die wir alle häufig erleben: Im Prozess des Wahrnehmens und Erkennens von uns bekannten Objekten einerseits und in der Anwendung künstlerischer bzw. handwerklicher Fähigkeiten und allgemeiner Fertigkeiten andererseits“ [109, S.24]. Achtung: Neuweg [152, S. 12] weist im Rahmen seiner terminologisch-semantischen Begriffsannäherung darauf hin, dass der Terminus „implizites Wissen“ ausgesprochen unscharf ist: Er wird keineswegs einheitlich gebraucht, im Rahmen verschiedener, paradigmatisch partiell oder vollständig unverträglicher Theoriekontexte verwendet und mit unterschiedlich starken Annahmen verbunden. Beispielsweise bezeichnet für Ryle [179] implizites Wissen „zunächst einen bestimmten Modus des (inneren oder äußeren) Tuns und ist Synonym für intuitives Können oder ‚knowing-how‘, wobei die Emphase darauf liegt, daß dieses Können nichtsdestotrotz den Anspruch auf Intelligenz erheben kann. Es gibt, so die These, *intelligentes Tun als Leistung sui generis*; Denken im allgemeinsprachlichen Wortsinne und das Erwägen von Wissen müssen ihm nicht notwendig vorausgegangen sein oder es begleiten“ [152, S. 12f]. Da implizites Wissen aber im Rahmen der BS/BKM in keiner Weise adressiert werden soll, reicht es hier aus, die häufigste Begriffsfassung, nämlich die, die von „nichtberichtbarem Wissen“ [63] spricht, zu verwenden. In weiterführenden Untersuchungen, insbesondere in solchen, die sich mit Wissenstransfer beschäftigen, kann es aber notwendig werden, implizites Wissen explizit zu adressieren. Schließlich wird z.B. von Myers und Davids [145] vermutet, dass sich erfolgreiche Mitarbeiter durch ein höheres Ausmaß an implizitem Wissen auszeichnen [152, S. 3]. Es wird dann zwangsläufig ein differenzierterer Umgang mit dem Begriff implizitem Wissen notwendig werden.

retechnik, z.B. über Funktionsweisen von Designpattern [70], welches zum Zwecke der Erklärung verbal vermittelt wird zu W^-_{BS} gehört (und tatsächlich auch durch ein Element der BKM nämlich *explain_knowledge* adressiert wird). Es wurde aber auch definiert, dass Wissen, welches (nur) im Rahmen eines Designvorschlages mitgeäußert wird, nicht W^-_{BS} zugerechnet werden soll. Eine solche Äußerung wird nur mit dem Konzept *propose_design* annotiert – der Wissensaspekt wird im Rahmen einer BS-Kodierung nicht explizit erfasst.[18]

Zu einem festen Zeitpunkt (innerhalb einer Paarprogrammierungssitzung) gilt also $W^-_{BS} \subset W^+_{BS}$[19], wobei noch einmal festgehalten werden soll, dass

[18] Achtung: Wenn Vorschläge begründet werden, so werden die entsprechenden Äußerungen bzw. Äußerungsteile einzeln kodiert, z.B. mit *explain_knowledge*. Hierauf wird noch im Detail eingegangen.

[19] An dieser Stelle wird vernachlässigt, dass Entwickler Informationen natürlich auch wieder vergessen können.

sich W_{BS}^{+} direkt aus der initialen Konzeption der Untersuchung sowie dem Vorverständnis der Untersuchenden[20] ergeben hat und die Konzeption von W_{BS}^{-} darüber hinaus auch Erkenntnisse und Entscheidungen widerspiegelt, die erst im Laufe des Analyseprozesses erlangt bzw. getroffen wurden. Wie schon angesprochen wird erst in Abschnitt 7.6 (Universelle Konzepte) erläutert worum es sich bei dem Wissen im Sinne von W_{BS}^{-} im Einzelnen handelt und welche Teile von W_{BS}^{-} tatsächlich explizit von Elementen der BKM adressiert werden bzw. warum nicht das gesamte W_{BS}^{-} explizit durch Elemente der BKM adressiert wird. Die folgenden vier Anmerkungen sollten aber jetzt schon beachtet werden:

- In anderen Zusammenhängen als dem hier dargestellten wird externalisiertes Wissen, also z.B. das verbalisierte, oft auch *Information* genannt (siehe z.B. [9]) bzw. wird davon gesprochen, dass die Rolle von Information darin besteht, „Wissen von einem System in ein anderes zu übertragen“ [116, S. 45]. Dieser Begriffsbildung folgend wird im Weiteren gelegentlich auch von Information statt von Wissen gesprochen. Auch wird nicht explizit zwischen Wissen und Wissenstransfer unterschieden. Dies liegt unter anderem daran, dass die BS/BKM nur verbalisiertes, also in Transfer befindliches Wissen adressiert. Die Verwendung des Begriffs Wissenstransfer muss im Weiteren allerdings weder bedeuten, dass ein solcher vom Sprecher in jedem Fall wirklich intendiert ist, noch wird durch ihn festgeschrieben, dass er (erkennbar) erfolgreich gewesen sein muss. Details hierzu werden im Rahmen der ausführlichen Beschreibungen des Wissensbegriffs der BS auf S. 247 erörtert.
- Wenn im Rahmen der Erörterungen der BS/BKM davon gesprochen wird, dass die durch eine bestimmte Objektklasse adressierten Phänomene Wissen übertragen oder (potentiell) „mit übertragen“, sind in der Regel alle Elemente dieser Klasse gemeint, nicht nur die *propose*-, *remember*- oder *explain*-Konzepte. Analoges gilt für die Fälle, in denen von in Form von Vorschlägen „mit übertragenem“ Wissen gesprochen wird. Hier sind in der Regel nicht nur die Vorschläge selbst, sondern auch die Bewertungen der Vorschläge (z.B. durch ein *amend*) gemeint. Das hierbei übermittelte Wissen kann allerdings höchst unterschiedliche Themenkreise betreffen. Beispielsweise kann durch ein *propose_design* ein technischer Fakt mit übermittelt werden (siehe z.B. Abbildung 7.2) oder durch ein *ask_knowledge* die Information, dass die sprechende Person auf einem bestimmten Sektor über kein Wissen verfügt.[21]

[20] Da die Praktik des Paarkodierens zum Einsatz kam (siehe Kapitel 10), handelt sich hier nicht nur um das Vorverständnis des Autors des vorliegenden Dokumentes.

[21] In welcher Weise die BS/BKM zwischen Wissensstandsäußerungen und Fragen unterscheidet, wird z.B. auf S. 340 erörtert.

- Bei der Darstellung des Wissensbegriffs könnte der Eindruck entstanden sein, dass bestimmtes Wissen bzw. bestimmte Formen von Wissen von der BKM entweder generell explizit adressiert oder generell nicht explizit adressiert werden. Dieser Eindruck ist aber nicht richtig. Vielmehr gilt, dass es passieren kann, dass ein und dieselbe Information (Teil des verfügbaren Gesamtwissens einer Person) bei einer Verbalisierung explizit kodiert wird und in einer anderen nicht. Ausschlaggebend ist hierbei die Art und Weise, in der die Information im Rahmen der Äußerungen verwendet wird. So wird Wissen in der Regel nur dann explizit adressiert, wenn es über einen sprachlichen Akt (evtl. unterstützt durch Skizzen und Bilder) vermittelt wird, dessen (primäre) Illokution eine Erklärung (oder Beurteilung bzw. Korrektur einer Erklärung) darstellt.[22]
- Der Verzicht auf eine Unterscheidung zwischen Meinung, wahrer Meinung und Wissen (im Sinne von wahrer, gerechtfertigter Meinung) kann sich in spezialisierten Untersuchungen hinderlich auswirken. Ggf. müssen Methoden bzw. Kriterien entwickelt werden, mit denen es – zumindest partiell – möglich wird, zu differenzieren.

2. **Separierung bestimmter Formen von Metawissen:** Die BKM unterscheidet zwischen einem bestimmten Typ von Wissen über Wissen – als *standard of knowledge* bezeichnet – und anderen Formen von explizitem Wissen (z.B. *knowledge* oder *finding*). Die Klasse *standard of knowledge* adressiert Einschätzungen, die eine Person über den Grad des Vorhandenseins von *eigenem* Wissen zu einem bestimmten Aspekt oder Thema macht, also eine gewisse Form von Metawissen. Mit *standard of knowledge* können sowohl Aussagen über das vollständige Fehlen von bestimmtem Wissen, das vollständige Vorhandensein von bestimmtem Wissen und auch Aussagen über jede Stufe dazwischen konzeptualisiert werden.

   Achtung: Die Sprecher formulieren solche Aussagen meist nicht extensional, also durch explizite Aufzählungen der vorhandenen bzw. fehlenden Wissensinhalte, sondern zusammenfassend und auf den Kontext bezogen (z.B. „Ich bin mir nicht 100%ig sicher.“ (PR1.1)). Details hierzu folgen in Abschnitt 7.6.

3. **Partielle Separierung von Bestandswissen und Erkenntnissen:** In bestimmten Fällen differenziert die BKM zwischen einem noch näher zu beschreibenden Teil dessen, was von Strömungen der Psychologie wie dem Kognitivismus als Langzeitgedächtnis (siehe z.B. [6]), genau genommen Inhalt

[22] Auf S. 316 wird erläutert, dass es Äußerungen bzw. Äußerungstypen gibt, bei denen die BKM zur Markierung eines Wissenstransfers eingesetzt wird, obwohl es sich bei der primären Illokution der Äußerungen nicht um eine Erklärung, Beurteilung oder ähnliches handelt.

des Langzeitgedächtnisses[23] bezeichnet wird – im Weiteren als *Bestandswissen* (*existing knowledge*, zum Teil mit *EK* abgekürzt) bezeichnet – und bestimmten neuen, im Laufe einer PP-Sitzung entstehenden Erkenntnissen, z.B. in Form von „Aha-Erlebnissen“[24]. Im Vorfeld gewonnene Erkenntnisse, die während einer Sitzung lediglich verbalisiert werden, müssen hierbei dem Bestandswissen zugerechnet werden. Außerdem gilt: Da es nicht sinnvoll ist, das Bestandswissen einer Person als statisch zu betrachten, werden Erkenntnisse, die innerhalb einer Paarprogrammierungssitzung erlangt werden, in der Regel zu einem späteren Zeitpunkt auch als Bestandswissen angesehen.

Verbalisiertes Bestandswissen wird zu großen Teilen durch die Klasse *knowledge* adressiert, während viele Verbalisierungen von Erkenntnissen durch die Klasse *finding* behandelt werden (Details hierzu finden sich in den Erörterungen der Konzeptklassen *knowledge* und *finding* auf S. 315 f bzw. S. 250 f).

Achtung: Die im vorliegenden Abschnitt angegebene Definition des Begriffs Bestandswissen liefert nur eine erste grobe Beschreibung, wobei die Verwendung des Terminus Langzeitgedächtnis die Tatsache verdeutlichen soll, dass gerade gewonnene Erkenntnisse, zumindest soweit diese in gewisser Weise als solche identifizierbar sind – z.B. weil es sich um die Benennung eines offensichtlich gerade beobachteten Ereignisses handelt – nicht zum Bestandswissen gezählt werden. Andere Aspekte, die sich daraus ergeben, dass der Begriff Bestandswissen im Kontext des von den Psychologen bzw. Kognitionswissenschaftlern R. C. Atkinson und R. M. Shiffrin entwickelten Drei-Komponenten-Modells des Gedächtnisses [6], bestehend aus Sensorischem Gedächtnis, Kurzzeitgedächtnis und Langzeitgedächtnis bzw. den Prozessen, die einen Übergang von „Information“ von einem zum anderen Teil bewirken (für Details siehe z.B. [38, S. 57f] oder [115, S.259f]), verankert wird, sollen erst einmal nicht interessieren.

[23] Genau genommen bezeichnet der Begriff Gedächtnis die Fähigkeit unseres Gehirns (bzw. des Nervensystems von Lebewesen), unsere „Erfahrungen aufzuzeichnen und zu speichern“ [195, S. IX] sowie wieder abzurufen und kann somit nicht unbedingt mit dem gleichgesetzt werden, was mit dem nicht einheitlich verwendeten Terminus Wissen (siehe z.B. Exkurs 10) bezeichnet wird. So geht dem Vermitteln bzw. Äußern von Wissen ein Abruf von Gedächtnisinhalten voraus, wobei dieser Abruf, das so genannte Erinnern, nach Ansicht von Frederic C. Bartlett – einem der Begründer der kognitiven Psychologie – „seinem Wesen nach [...] ein kreativer Rekonstruktionsprozeß ist“ [195, S. 7]: „Erinnern ist keine Reaktivierung unzähliger fixierter, lebloser und fragmentarischer Spuren. Es ist eine phantasievolle Rekonstruktion, oder eine Konstruktion, errichtet aus dem Spannungsfeld zwischen unserer Haltung gegenüber einer ganzen, aktiven Masse organisierter ehemaliger Reaktionen oder Erfahrungen einerseits und einem kleinen, hervorstechenden Detail andererseits, das gewöhnlich in Bild- oder Sprachform auftritt“.

[24] Ein „Aha-Erlebnis“ soll hier nur grob eine unvermittelt aufgetretene und geäußerte Einsicht, z.B. in Zusammenhänge oder die Lösung eines Problems, bezeichnen. Es wird nicht explizit auf Begriffsbildungen aus der Psychologie wie z.B. die von Bühler [35] Bezug genommen. Details der Klasse *finding*, insbesondere solche, die die Abgrenzung zur Klasse *knowledge* verdeutlichen, werden in Abschnitt 7.6 erläutert.

In Abbildung 7.1 wird ein Überblick darüber gegeben, in welcher Weise die einzelnen Konzeptklassen der BKM Wissen oder Wissenstransfer adressieren bzw. „mit adressieren“. In diesem Zusammenhang soll schon einmal auf drei Besonderheiten der BS/BKM hingewiesen werden:

- Die Klasse *hypothesis* bzw. Hypothesen und Vermutungen werden dem Bereich Wissen zugeordnet. Insbesondere kann es sich bei den durch die Elemente der Klasse adressierten Phänomene sogar um Äußerungen handeln, die Bestandswissen transferieren. Dies liegt daran, dass im Rahmen der BS/BKM auch die Kenntnis von Hypothesen als Wissen angesehen wird.
- Man könnte geneigt sein, aus Abbildung 7.1 zu schließen, dass Äußerungen, in denen keine neuen Erkenntnisse formuliert werden, generell nicht mit Konzepten der Klasse *finding* zu annotieren sind. Dies ist aber nicht richtig. Da mit der BS möglichst auch Zusammenhänge zwischen Äußerungen wie z.B. Diskussionsverläufe sichtbar gemacht werden sollen (hierauf wird in Abschnitt S. 7.2 näher eingegangen), werden z.B. auch Äußerungen, die eine Erkenntnis um Bestandswissen ergänzen, darüber hinaus aber keine Erkenntnis verbalisieren, mit einem *finding*-Konzept annotiert (z.B. einem *amend_finding*). Hierauf wird später genauer eingegangen (siehe z.B. S. 335 oder S. 338).
- Die BKM bzw. die HHI-Konzepte sind nicht dahingehend konzipiert, jegliche Formen von Bestandswissen explizit zu adressieren. Schließlich werden nur Äußerungen betrachtet. Aber auch das im Rahmen dieser verbalisierte Bestandswissen wird von der BKM nicht vollständig bzw. nicht immer explizit adressiert. Auch hierauf wird später (siehe z.B. S. 242 f) näher eingegangen.

Betrachtet man die Klassen *knowledge* und *finding* genauer, so wird schnell klar, dass es im Rahmen der hier verwendeten Analysemethoden oft schwierig werden kann, zu entscheiden, ob es sich bei einer Äußerung um die Verbalisierung von schon länger vorhandenem Wissen oder die Formulierung einer gerade gewonnenen Erkenntnis handelt. Aus diesem Grund werden in Abschnitt 7.6.1 Indikatoren erläutert, anhand derer zwischen durch *knowledge* zu adressierenden Verbalisierungen von Bestandswissen und durch *finding* zu konzeptualisierenden Äußerungen unterschieden werden soll. Genau genommen handelt es sich hierbei um Regeln, mit denen festgelegt wird, auf welche Weise Erkenntnisse als solche identifizierbar sein müssen, damit sie *finding* zugerechnet werden können.[25] In diesem Zusammenhang ist es auch von Interesse, in welcher Weise die Erkenntnis einer Person, dass es angebracht ist, bestimmtes schon länger vorhandenes Wissen in einer konkre-

[25] Wie noch erläutert werden wird, gibt es auch identifizierbare Erkenntnisse, die nicht *finding*, sondern anderen Klassen zuzurechnen sind (z.B. *completion*).

ten (Programmier-) Situation anzuwenden, von der BS/BKM berücksichtigt wird.

4. **Separierung von Vorschlägen:** Obwohl auch Vorschläge zum Programmdesign oder zur Gestaltung des weiteren Entwicklungsprozesses

   – in gewisser Weise Meinungen sind, die vor dem Hintergrund des zu erreichenden Entwicklungszieles oft als richtig oder falsch klassifiziert oder zumindest als mehr oder weniger angemessen bewertet werden können

   – bzw. hinter solchen Vorschlägen Wissen steht, welches zumindest zum Teil mit übertragen wird und somit die Vorschläge selbst als Wissenstransfer oder zumindest als Äußerungen, die Wissen „mit übermitteln“ interpretiert werden könnten,

   werden sie im Rahmen der BS/BKM nicht über *knowledge*- oder *finding*-Elemente konzeptionalisiert, sondern mittels eigener Objekte wie z.B. *design* oder *step* behandelt (diese Sonderbehandlung erfahren auch die Bewertungen solcher Vorschläge, z.B. durch *amend_design*). Ein Beispiel hierfür ist in Abbildung 7.2 dargestellt.

   Durch die Abtrennung wird der besondere Stellenwert hervorgehoben, den Vorschläge, sei es in Bezug auf das Produkt oder in Bezug auf den Prozess, für den Ablauf von PP-Sitzungen besitzen. Hierbei ist es von zentraler Bedeutung, dass zwischen einem Vorschlag und seiner Begründung unterschieden wird. Begründungen werden separat mittels *knowledge*- oder *finding*-Konzepten annotiert. Ein Beispiel hierfür ist in Tabelle 7.3 dargestellt (siehe Äußerungen 4 und 5).

   Analoges gilt für Beurteilungen des Arbeitsfortschritts. Obwohl es sich bei solchen Äußerungen in der Regel um Erkenntnisse handelt, kommen spezielle Objekte (*completion* und *state*) zum Einsatz und nicht *finding*. In Abbildung 7.1 wird dies noch einmal zusammenfassend dargestellt.

In Abschnitt 7.6 wird darauf eingegangen, in welcher Weise die hier erläuterten Separierungen dazu beitragen, dass W^{-}_{BS} eine echte Teilmenge von W^{+}_{BS} ist.

7.1.2 *propose* vs. *explain*

Im letzten Abschnitt wurde ein zentrales Abgrenzungskriterium zwischen den Klassen *design*, *requirement*, *step*, *strategy* und *todo* auf der einen und den Klassen *knowledge* und *finding* auf der anderen Seite erläutert. Vereinfacht lautet es: Vorschläge zur Gestaltung von Artefakten oder des Arbeitsprozesses bzw. Beurteilungen solcher (Ablehnungen, Zustimmungen, Erweiterungen) werden durch Elemente der Klassen *design*, *requirement*, *step*, *strategy* bzw. *todo* konzeptualisiert, „reiner“ Wissenstransfer bzw. Beurteilungen eines solchen (Ablehnungen,

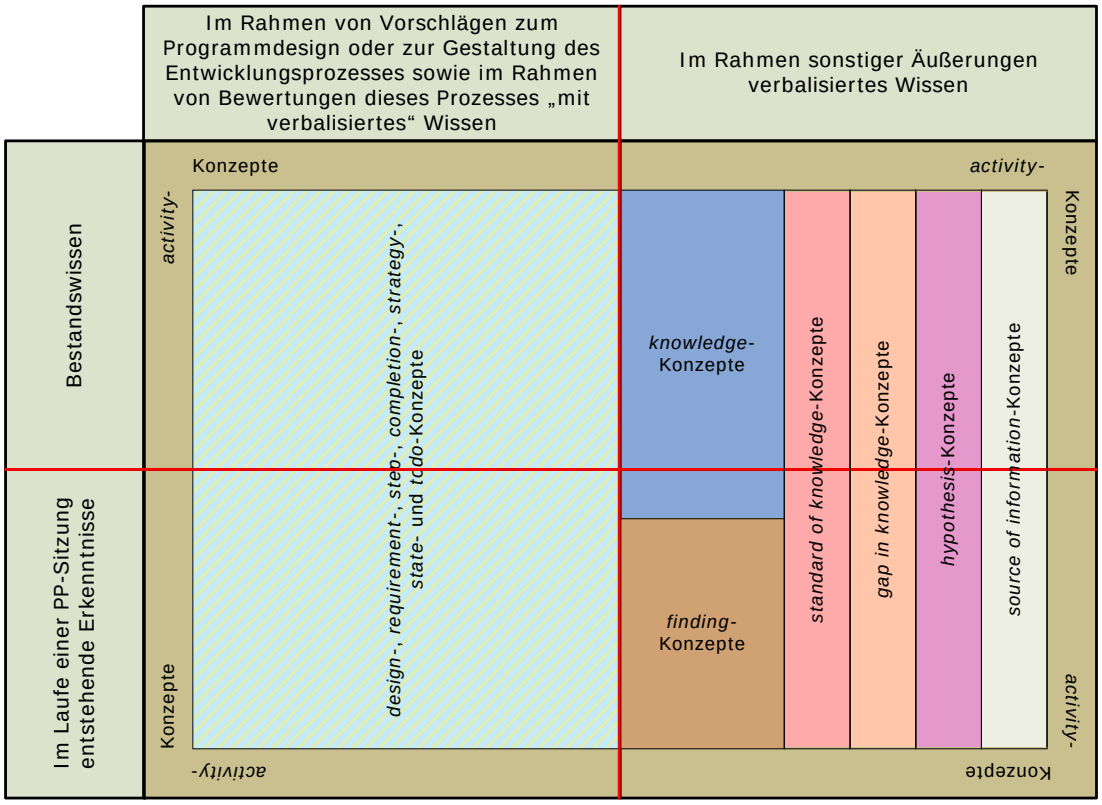


Abb. 7.1: Darstellung des Zusammenhangs zwischen einzelnen HHI-Objektklassen und bestimmten Formen von verbalisiertem Wissen (im Sinne von W_{BS}^{+}). In den Zeilen wird zwischen Bestandswissen und Erkenntnissen, in den Spalten zwischen im Rahmen von Design- und Prozessvorschlägen (oder im Rahmen von Bewertungen von Design- und Prozessvorschlägen) sowie im Rahmen von Prozessbewertungen „mit übermitteltem“ Wissen und im Rahmen sonstiger Äußerungen verbalisiertem Wissen (z.B. im Rahmen von Erklärungen und deren Bewertungen geäußertem) unterschieden. Es gilt: 1. Mit den Konzepten der Klassen *design*, *requirement*, *step*, *completion*, *strategy*, *state* und *todo* wird nicht zwischen Bestandswissen und Erkenntnissen unterschieden. 2. Auch die Konzepte der Klassen *standard of knowledge*, *gap in knowledge*, *hypothesis* und *source of information* differenzieren nicht, wobei ihre Aufgabe im Gegensatz zu den vorher genannten darin besteht, bestimmte Formen von Wissen bzw. Wissenstransfer zu kennzeichnen. 3. Konzepte der Klasse *finding* adressieren Verbalisierungen von Erkenntnissen bzw. Erkenntnisbewertungen, die nicht von den unter 1. und 2. genannten Klassen mit adressiert werden. In Unterabschnitt 7.6.1 werden Kriterien erläutert, die es möglich machen sollen, solche zu identifizieren. 4. Die Konzepte der Klasse *knowledge* adressieren alle Äußerungen die (allem Anschein nach) nicht von anderen Elementen der BKM (außer solchen der Klasse *activity*) behandelt werden können. Bei derartigen Äußerungen handelt es sich oft um solche, mit denen Bestandswissen zum Zweck des Erklärens transferiert wird (siehe Abschnitt 7.6.5). Da nicht garantiert werden kann, dass Erkenntnisäußerungen, immer als Verbalisierungen von Erkenntnissen zu identifizieren sind, kann es (unabsichtlich) dazu kommen, dass auch Erkenntnisse mit *knowledge*-Konzepten annotiert werden. Im Bild wird also die Real- und nicht die Idealsituation dargestellt. Das von der Klasse *knowledge* eigentlich zu adressierende Bestandswissen sei im Weiteren mit W_k bezeichnet. 5. Die Konzepte der Klasse activity bilden eine Ausnahme, da sie aufgrund ihrer Fassadeneigenschaft alle hier angesprochenen Wissensformen adressieren bzw. „mit adressieren“ können (siehe Abschnitt 7.7).

#	HHI-Kode	Äußerung	Kontext/Kommentar
1	*P2.ask_knowledge*	„Was passiert, wenn der nicht gesetzt ist?“	Mit dem Artikel ‚der‘ referenziert *P2* einen Übergabeparameter.
2	*P1.explain_knowledge*	„Er macht auf jeden Fall das Skript so lange lauffähig, wie wir (.) ähm (.) noch nicht an allen Stellen den ‚Change‘ machen - also den `Memcache` schreiben. (...)“	Der Driver antwortet sofort auf die Frage des Observers. Es ist wohl so, dass er bereits vorhandenes Wissen verbalisiert (keine Indizien für eine neue Erkenntnis). Nach seiner Erklärung hält er ca. drei Sekunden inne.
3	*P1.explain_finding*	„(~Na) obwohl, wenn wir es nicht in den `Memcache` schreiben, wird er es auch irgendwann zurückgeben.“	Das Innehalten (s.o.) im Zusammenspiel mit der Verwenden des Subjunktion ‚obwohl‘ spricht dafür, dass es sich nicht um Bestandswissen handelt. *P1* ergänzt also seine eben abgegebene Erklärung durch eine Einsicht.
4	*P1.propose_design*	„Wir können das auch da oben lassen, erst mal. (..)“	Der Driver macht auf Basis seiner Einsicht einen Designvorschlag. Dieser ist im folgenden Sinne zu verstehen: Wir ändern die Logik in der kurz zuvor mit `TODO` markierten Bedingung erst einmal nicht.
5	*P1.ask_standard of knowledge*[a]+ *P1.explain_finding*	„Also‿weißt‿Du‿was‿ich‿meine?‿Wenn wir jetzt einfach nur dieses Skript ändern, (.) dann würde halt einfach nur (.) das hier nie erfüllt werden. Deswegen würden wir nie `exit('NOCHANGE')` zurückgeben, sondern einfach hier weitermachen – und das wäre ja die gleiche Funktionalität.“	Nach einer kurzen Pause begründet *P1* seinen Vorschlag, indem er seine Einsicht erneut, aber mit anderen Worten erläutert. Dabei zeigt er auf bestimmte Stellen im Kode.

[a] Kann auch als rhetorische Frage interpretiert werden.

Tab. 7.3: Beispiel für einen Designvorschlag, der mit einer Begründung auf Basis einer Einsicht einhergeht: In der Episode beschäftigt sich das Paar mit Kodezeilen, in denen vom gerade in Bearbeitung befindlichen PHP-Skript Übergabeparameter überprüft werden. Diese Überprüfung hat die Form einer `if`-Anweisung. Sie bricht das Skript bei bestimmten disjunktiv verknüpften Eigenschaften der Übergabeparameter ab. Im Laufe einer vorher geführten Diskussion hatte der Driver einen Teil der Bedingungen mit einem `TODO` versehen und somit markiert, dass diese noch in einer gewissen Weise geändert werden müssen. Die Episode stammt aus der Sitzung PR1.1 (14:08:27–14:09:00 – siehe Abbildung C.2) und startet mit einer Frage des Observers zu einem der Übergabeparameter.

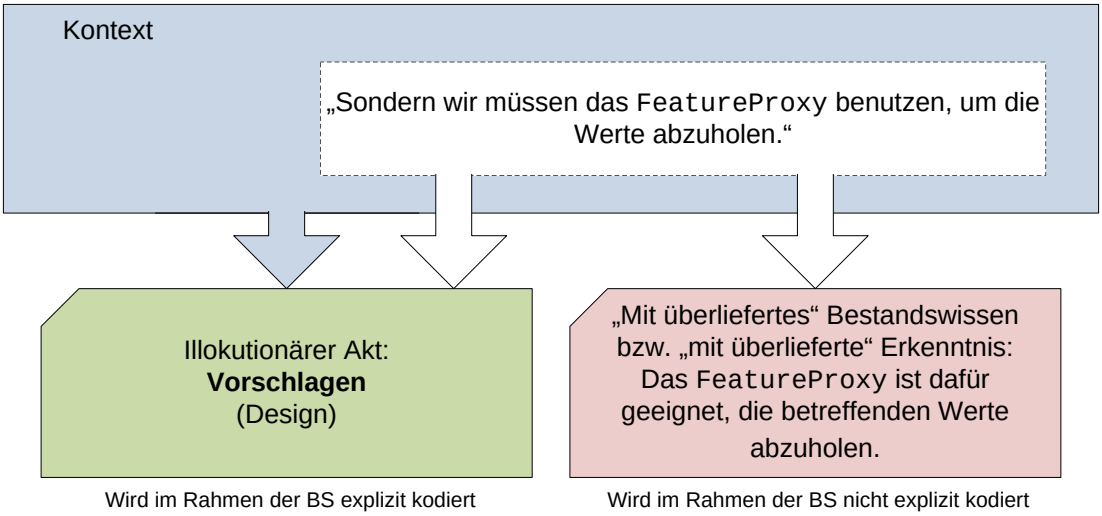


Abb. 7.2: Beispiel für Wissen, welches bei der Äußerung eines Vorschlags „mit übermittelt“ wird: Interpretiert man die angegebene Äußerung (aus Sitzung PR2.1) im Kontext ihres Auftretens, so ist sie als Designvorschlag identifizierbar (Details hierzu folgen ab S. 167). Schon die Äußerung an und für sich zeigt aber, das durch sie Wissen übermittelt wird, z.B. dasjenige, dass das `FeatureProxy` überhaupt dazu geeignet ist, die betreffenden Werte „abzuholen“. Wenn Wissen auf diese Weise „mit übermittelt“ wird, ist es, zumindest im Rahmen der BS, nicht zusätzlich als solches zu kodieren. Die Konzeptualisierung erfolgt vielmehr nur mittels *propose*-Konzepten. Analoges gilt auch für alle Äußerungen, in denen auf das Produkt oder den Prozess bezogene Vorschläge erweitert, geändert, abgelehnt oder angenommen werden. Insbesondere wird die Erkenntnis, dass bestimmtes Wissen zur Anwendung gebracht werden sollte, nicht explizit kodiert.

Zustimmungen, Erweiterungen) im Wesentlichen durch *knowledge*- und *finding*-Konzepte.[26]

Diese Trennung spiegelt sich auch in der Verwendung der Verben *propose* und *explain* wider.[27] So gibt es beispielsweise das Konzept *propose_design*, um einen Designvorschlag zu markieren, aber nicht das Konzept *explain_design*, um Erklärungen bez. eines solchen kenntlich zu machen. Begründungen von Vorschlägen werden vielmehr wie „reiner“ Wissenstransfer behandelt und mit *knowledge*- oder *finding*-Konzepten wie *explain_knowledge* oder *explain_finding* kodiert.

In diesem Zusammenhang sollte beachtet werden: Obwohl es auf den ersten Blick einfach aussieht, zwischen einem Vorschlag und seiner Begründung bzw. zwischen Vorschlägen und reinem Wissenstransfer zu unterscheiden, ist dem oft nicht so. Denn betrachtet man konkrete PP-Sitzungen, so sieht man schnell, dass die Sprecher ihre Vorschläge nur selten explizit als solche markieren, d.h. ein

[26] Zum Zwecke der Vereinfachung wird an dieser Stelle ignoriert, dass es weitere Klassen wie z.B. *completion* oder *state* gibt, die ebenfalls gewisse Formen von Wissenstransfer adressieren (siehe Abbildung 7.3).

[27] Eine Ausnahme bildet das Konzept *propose_hypothesis* (siehe Abschnitt 7.6.2).

performatives Verb verwenden. Z.B. formulieren sie nur in Ausnahmefällen wie folgt:

> „Ich schlage vor, dass wir ...“

Oft sehen die Äußerungen eher wie Wissenstransfers aus, z.B.

> „Sondern wir müssen das `FeatureProxy` benutzen, um die Werte abzuholen.“ (Details sind in Tabelle 7.4 (Abschnitt 7.4.1) und Abbildung 7.2 zu finden)

Es sollte also immer genau überprüft werden, welchen Zweck der Sprecher mit seiner Äußerung verfolgt.[28] Insbesondere muss dabei zwischen der grammatischen Satzform und der jeweils aktuellen illokutiven Funktion (siehe Exkurs 6 auf S. 146) eines Satzes unterschieden werden (in Abschnitt 7.4 in Tabelle 7.5 werden unterschiedliche Intentionen diskutiert, die durch *propose*-Konzepte adressiert werden).

Für das obige Beispiel zeigt die Analyse des Kontextes, dass es dem Entwickler darum geht, festzulegen, wie auf bestimmte Werte zugegriffen werden sollte. Es handelt sich also um ein *propose_design*. Außerdem gilt: Da der Sprecher in keiner Weise erklärt, warum der Zugriff wie vorgeschlagen gestaltet sein sollte, handelt es sich nicht – auch nicht in Teilen – um ein *explain_knowledge*.

7.1.3 *explain* vs. *think aloud*

Wie in den letzten Abschnitten beschrieben adressiert das Verb *explain* alle Äußerungen, mit denen versucht wird, einen Sachverhalt, z.B. handlungsrelevantes Wissen oder eine gerade gewonnene Erkenntnis, zu vermitteln, wobei Vorschläge zur Gestaltung von Artefakten, z.B. zum Aufbau des Programmkodes sowie zur Strukturierung des Arbeitsprozesses, nicht zu dieser Klasse von Äußerungen gehören. Nicht erläutert wurde bisher die Abgrenzung zu *think aloud*. Um dies tun zu können, muss zunächst einmal *think aloud* näher beschrieben werden.

Im Rahmen der BKM wird das Verb *think aloud* verwendet, um Äußerungen bzw. Abfolgen von Äußerungen zu konzeptualisieren, die im Zusammenhang mit aktuell ablaufenden HCI- oder HEI-Aktivitäten der sprechenden Person stehen. Es wird also z.B. dann eingesetzt, wenn ein Driver Editierschritte, die er gerade durchführt, mitspricht.[29] Es wurde festgelegt, dass direkt aufeinander folgende[30]

[28] In Abschnitt 9.3.1 wird darauf eingegangen, dass es natürlich nicht immer möglich ist, den Zweck bzw. die Intention einer Äußerung zweifelsfrei zu ermitteln.

[29] In gewisser Weise sieht es bei *think aloud*-Phänomenen so aus, als wenn der Sprechende (bewusst oder unbewusst) daran interessiert ist, dass sein Partner „auf der Höhe“ der gerade ablaufenden HCI/HEI-Aktivitäten (und der mit ihnen zusammenhängenden Gedanken bzw. Überlegungen) bleiben kann. Hierbei handelt es sich aber lediglich um eine Vermutung. Schließlich ist eine solche Intention des Sprechers in der Regel nicht – zumindest nicht ohne weiteres – sicher festzumachen.

[30] Was genau unter „direkt aufeinander folgende Äußerungen“ verstanden werden soll, wird später genauer erläutert. Vorerst soll ein grobes umgangssprachliches Verständnis ausreichen.

und auf dieselbe HCI/HEI-Aktivität bezogene Äußerungen zu einem *think aloud* zusammengefasst werden sollen. In den Abschnitten 7.7 und 9.3 wird hierauf noch einmal genauer eingegangen.

An dieser Stelle könnte man annehmen, dass es hilfreich wäre, *think aloud*- und *explain*-Phänomene auch anhand der Kommunikationsrichtung auseinander zu halten, d.h. festzulegen, dass es sich bei Phänomenen des Typs *think aloud* immer um ungerichtete Äußerungen, also um das, was man umgangssprachlich als „laut Denken“ bezeichnen würde, und bei solchen des Typs *explain* um gerichtete Äußerungen, also bewusst und direkt an den Partner adressierte, handeln soll. In der Praxis zeigt sich aber schnell, dass eine solche Differenzierung nicht praktikabel ist. Denn potentiell können derartige „erläuternde Kommentare“ zu bestimmten, wohl umrissenen HCI- oder HEI-Aktivitäten beide Charakteristika aufweisen, wenn auch in unterschiedlichen Teilen. Außerdem – und das ist noch viel entscheidender – kann oft kaum zweifelsfrei herausgefunden werden, um was für einen Typ von (Teil-)Äußerung (gerichtet oder ungerichtet) es sich handelt, da die Sprecher ihre diesbezügliche Intention, wenn sie denn überhaupt eine solche haben bzw. sich selbst einer solchen bewusst sind, nicht immer verbalisieren.[31]

Es gilt aber das Folgende: Durch die Einführung der Klasse *think aloud* bzw. des Konzeptes *think aloud_activity* wurde eine partielle Hierarchie auf der Menge der Konzepte definiert. Denn Phänomene bzw. Äußerungen vom Typ *think aloud_activity* können meist weiter aufgebrochen werden, z.B. in Teilsequenzen vom Typ *explain_knowledge*, *explain_finding* und/oder *propose_step*. Es kann also passieren, dass eine Äußerung bzw. ein Äußerungsteil sowohl mit *think aloud* als auch mit *explain_knowledge* annotiert werden muss. Details hierzu werden in Abschnitt 7.7 erläutert.

7.1.4 *disagree+propose* vs. *challenge*

In gewisser Weise handelt es sich bei einem *challenge* um ein *disagree* mit in der Regel nachfolgendem *propose*. Es stellt sich also die Frage, ob nicht, zumindest im Rahmen der BKM, auf *challenge*-Konzepte verzichtet werden sollte. Folgende zusammenhängende Gründe sprechen gegen einen solchen Verzicht:

1. Mit dem Annotieren eines *disagree*- und eines *propose*-Konzeptes wird weniger ausgedrückt, als mit dem Annotieren eines *challenge*-Konzeptes, da bei ersterem der explizite Bezug zwischen Ablehnung und neuem Vorschlag, also ein wesentlicher Teil der Intention des Sprechers, verloren geht.

[31] Bei der Herleitung der BKM wurde ursprünglich tatsächlich versucht, zwischen gerichteten und ungerichteten Äußerungen zu unterscheiden. Trotz Schwierigkeiten wurde diese Unterscheidung über einen längeren Zeitraum aufrecht erhalten. Letztendlich schlug dieser Ansatz aber fehl. Dies hatte zur Folge, dass eine Reihe von *think aloud*-Kodes (*think aloud_completion*, *think aloud_finding* und *think aloud_state*) wieder abgeschafft bzw. durch *explain*-Kodes ersetzt wurden. Übrig blieb nur noch *think aloud_activity*. In [180] bzw. [181] ist noch die ursprüngliche Version der BKM zu finden.

2. Das Verb *propose* ist im Rahmen der BKM als initiativ klassifiziert worden (siehe S. 144). Diese Festlegung führt dazu, dass mit jeder Vergabe eines *propose*-Konzeptes der Aspekt „Initiative“ kodiert wird, ohne dass mit zusätzlichen Eigenschaften gearbeitet werden muss. Ein Verzicht auf *challenge*, um stattdessen *disagree* und *propose* zu verwenden, wäre also nur dann möglich, wenn auf die „automatische“ Erfassung dieser Eigenschaft verzichtet würde, d.h. Konzepte der Klasse *propose* nicht automatisch initiatives Vorgehen adressieren würden.

Ähnliche Überlegungen gelten bez. der Abgrenzung von *agree* und *propose* zu *amend*.

7.2 Zentrale Prinzipien im Umgang mit HHI-Konzepten

Im Rahmen des Analyseprozesses haben sich zwei zentrale, miteinander interagierende Kodierprinzipien herausgebildet. Auf ihnen bauen die HHI-Konzepte auf und ihnen sollte auch bei der Verwendung der HHI-Konzepte, soweit es möglich und im Rahmen des Analyseziels sinnvoll ist, gefolgt werden. Um das Verständnis der nachfolgenden Erörterungen zu erleichtern, sollen sie hier schon einmal vorweggenommen werden:

1. **Primäre Intentionen kodieren**: Mittels der BKM-Konzeptualisierung einer Äußerung sollte möglichst die primäre Intention des Handelnden bzw. des Sprechenden, erfasst werden. In diesem Zusammenhang stellen sich allerdings zwei Fragen:

   (a) Existiert in jedem Fall genau eine zu ermittelnde primäre Intention? Bzw.: Was soll die Eigenschaft „primär“ in diesem Zusammenhang genau bedeuten?

   (b) Die Intentionen, mit denen Äußerungen gemacht werden können, decken ein großes Spektrum ab – z.B. Fragen, Widersprechen, Versprechen, Bitten, Drohen, Empfehlen, Behaupten, Ersuchen, Anerkennen, Einwenden, Tadeln, Argumentieren.[32] Dies gilt – zumindest theoretisch – auch für solche Äußerungen, die im Rahmen von Paarprogrammierungssitzungen gemacht werden. Wie kann also das angestrebte Ziel einer überschaubaren Menge von Basiskonzepten erreicht werden?

   Im Rahmen der Erläuterungen der einzelnen Konzepte bzw. Konzeptklassen wird deutlich werden, welche Antworten die BS/BKM in Hinsicht auf diese Fragen gibt. In den Abschnitten 9.1.1, 9.3.1 und 9.3.5 werden diese zusammengefasst.

[32] Laut Searle [189, S. 40] behauptet Austin, „[dass] es im Englischen über tausend solcher Ausdrücke gebe“.

2. **Sprachliche Bezüge kodieren**: Mittels BKM-Konzeptualisierungen von Äußerungen bzw. Episoden sollten sprachliche Bezüge, beispielsweise Frage-Antwort-Paare oder Entscheidungsfindungsprozesse (siehe beispielsweise Abbildung 7.4) sichtbar gemacht werden. Dies geschieht über die Verbbestandteile der Konzepte. Darüber hinaus wird das in einer Episode verhandelte, oft durch genau ein Objekt wie z.B. *strategy* konzeptualisierbare Thema (Äußerungsgegenstand) nicht als statische, sondern als sich im Verlauf einer Diskussion verändernde Sache betrachtet. Diese Sichtweise ist allerdings nicht explizit aus einzelnen Kodierungen ablesbar. Hierzu betrachte man folgendes Beispiel: Wird zu Beginn eines Dialoges eine Strategie vorgeschlagen, und entspinnt sich aus diesem Vorschlag eine Diskussion, so referenzieren die einzelnen auf die Strategie selbst bezogenen Äußerungen wie z.B. Zustimmungen in der Regel nicht deren initiale Form, sondern die zuletzt z.B. durch ein *challenge_strategy* modifizierte. Mit einer BS/BKM-Kodierung wird dieser Bezug zur letzten Objektversion festgehalten – und zwar wie erwähnt über den Verbbestandteil des Konzeptes. Das Objekt wird hierbei (wie in allen Stadien) unverändert benannt. Veränderungen am Objekt, die in den Schritten davor gemacht wurden, sind somit aus einer einzelnen Kodierung nicht mehr abzulesen. Im vorliegenden Beispiel würde also ein *agree_finding* annotiert werden und kein *agree_(challenge_finding)* oder gar *agree_finding'* (für ein konkretes Beispiel siehe Abbildung 7.6). Auf die Kodierung von sprachlichen Bezügen wird noch einmal in Abschnitt 9.3.11 eingegangen.

   Achtung:

   – Dem Anliegen, sprachliche Bezüge festzuhalten, kann, zumindest wenn der Kodieraufwand bewältigbar bleiben soll, nicht immer vollständig nachgekommen werden. Schließlich ist es möglich, dass sich eine Äußerung zugleich auf mehrere vorhergehende bezieht, oder dass das verhandelte Thema sich derart ändert, dass es unumgänglich erscheint, ein anderes Objekt zu verwenden. Im weiteren Verlauf der Erörterung wird hierauf genauer eingegangen (siehe z.B. Abschnitt 7.9).

   – Im Rahmen der BS wird in weiten Teilen, d.h. abgesehen von der Klasse *activity*, außer Acht gelassen, dass sich Äußerungen auch auf HCI/HEI-Aktivitäten oder Ergebnisse bzw. Folgen von HCI/HEI-Aktivitäten beziehen können. Man betrachte hierzu folgendes Beispiel aus PR1.1. Hier äußert der Observer

     „Nee,‿das‿war‿schon‿korrekt“,

     nachdem der Driver durch das Markieren von Kode angedeutet hat, dass er bestimmte Teile löschen will (Details zum Kontext werden im Rahmen der Klasse *activity* in der Tabelle 7.55 auf S. 360 erläutert). Im Rahmen der BS wird eine solche Äußerung mit *propose_design* annotiert, obwohl es sich bei genauer Betrachtung um einen Wider-

spruch mit Gegenvorschlag handelt – allerdings um einen, bei dem der ursprüngliche Vorschlag nicht verbalisiert, sondern implizit durch Textmanipulation kundgetan wird. Im Rahmen der BS wird derartig vorgegangen, da es, zumindest mit vertretbarem Aufwand, kaum möglich ist, alle impliziten Vorschläge durch Annotationen festzuhalten. Es ist aber davon auszugehen, dass weiterführende Untersuchungen andere Vorgehensweisen etablieren müssen, damit für die Analysen unverzichtbare Informationen nicht verloren gehen. So ist es beispielsweise denkbar, implizite Vorschläge genau dann zu konzeptualisieren, wenn dies aufgrund von (nachfolgenden) Äußerungen notwendig erscheint. In Folge eines solchen Vorgehens müsste dann allerdings auch die Anmerkung selbst anders kodiert werden (im Beispiel z.B. durch ein *challenge_design*).

In diesem Zusammenhang muss noch einmal betont werden, dass es sich bei dem hier beschriebenen Verfahren der Konzeptualisierung um einen (zumindest ab einem gewissen, schwerlich genau festzumachenden Punkt) subjektiven Prozess handelt, der aber der intersubjektiven Nachvollziehbarkeit (siehe S. 124) höchste Priorität einräumt. Es muss also darauf geachtet werden, dass dieser Nachvollziehbarkeit sowohl bei der Bestimmung von Intentionen wie auch bei der von Bezügen genügt wird.

7.3 Übersicht über die HHI-Konzepte entlang von Hauptklassen

Im Laufe der GTM-Analyse bildeten sich fünf Hauptklassen von HHI-Konzepten heraus:

1. **Produktorientierte Konzepte** werden im Zusammenhang mit Vorschlägen verwendet, die die Gestalt des zu entwickelnden Programms betreffen, also die Inhalte, die Struktur und die Anordnung der Programmartefakte.

2. **Prozessorientierte Konzepte** referenzieren Vorschläge und Entgegnungen auf solche, die sich auf die Gestaltung des Arbeitsprozesses beziehen. Die Menge der produktorientierten und prozessorientierten Konzepte wird im Weiteren auch $P\&P$ genannt. Ein Element aus $P\&P$ heißt dementsprechend auch $P\&P$-Konzept, ggf. auch $P\&P$-Vorschlag. Die in den Namen der $P\&P$-Konzepte vorkommenden Objekte (z.B. *design* und *step*) werden zusammenfassend auch als $P\&P$-Objekte oder $O_{P\&P}$ bezeichnet.

3. **Universelle Konzepte** (UK) adressieren Äußerungen, mit denen Wissen angefordert, transferiert und beurteilt wird sowie Wissenslücken offenbart oder Vermutungen bzw. Hypothesen aufgestellt und bewertet werden. Sie heißen universell, da sie sowohl im Produkt- wie auch im Prozesskontext

verwendet werden können. Mit ihnen ist es also möglich, sowohl Äußerungen zu konzeptualisieren, die sich auf das zu entwickelnde Programm beziehen, wie auch solche, in denen es um den Arbeitsprozess geht.

Achtung: Im Rahmen der BS/BKM werden einzelne Äußerungen in der Regel nicht gleichzeitig mit einem $P\&P$-Konzept und einem universellen Konzept annotiert. Hierbei gilt die Regel, dass eine Konzeptualisierung mit einem $P\&P$-Konzept immer der mit einem universellen Konzept vorzuziehen ist. Hierdurch wird vermieden, dass letztendlich alle Phänomene nur als unterschiedliche Formen von inhaltlich nicht weiter ausdifferenziertem Wissenstransfer betrachtet werden.

4. **Fassadenkonzepte** bilden eine Teilmenge der universellen Konzepte und liefern einen vereinfachten, auf einen bestimmten Aspekt fokussierten Blick auf in der Regel komplexere Phänomene, z.B. auf Äußerungen oder Sequenzen von Äußerungen bez. momentan ablaufender HCI- oder HEI-Aktivitäten. Hierdurch stellen sie unter anderem einen Kontext zur Verfügung, in dem diese komplexeren Phänomene aufgesplittet und durch spezifischere Konzepte beschrieben werden können.

5. **Sonstige Konzepte** sind Kennzeichnungen für Äußerungen, die unverständlich sind oder nicht mit der Aufgabenbearbeitung zusammenhängen.

In Abbildung 7.3 sind die Elemente (HHI-Konzepte) dieser Klassen sortiert nach ihren sich aus den Objektbestandteilen ergebenden Unterklassen (z.B. *strategy*) aufgelistet. Es ist zu erkennen, dass die einzelnen Elemente einer Unterklasse jeweils genau einen potentiellen Diskussionsschritt zu einem konkreten, durch den Objektbestandteil konzeptualisierten Thema oder Aspekt adressieren. Abbildung 7.4 erläutert mögliche Sequenzen dieser durch Verben repräsentierten Diskussionsschritte für produktorientierte sowie prozessorientierte Konzepte.

Die Abbildungen 7.3 und 7.4 wie auch die Tabellen 7.1 und 7.2 abstrahieren von einer großen Zahl von Details. Diese werden erst in den nachfolgenden Unterabschnitten entlang der durch die identifizierten Objekte definierten Unterklassen (also z.B. *design* oder *step*) in in der Regel jeweils vier Teilschritten erläutert:

- Zuerst wird der Fokus der Klasse näher spezifiziert, d.h. die Beschreibung des Objektbegriffs detailliert. Hierbei wird auf solche primär charakterisierenden Eigenschaften eingegangen, die für die gesamte Klasse von Bedeutung sind. Falls es für das Verständnis der Klasse angebracht erscheint, wird auch näher auf die Bedeutung einzelner initiativer Konzepte eingegangen.

- Dann werden spezielle, über die Darstellungen in Abbildung 7.3 hinausgehende Eigenschaften einzelner Konzepte erläutert. Im Wesentlichen geht es dabei um konzeptspezifische, primär charakterisierende Eigenschaften sowie um besonders relevante Eigenarten verbaler Kommunikation. Erst diese Beschreibungen machen eine konsistente Anwendung der BKM möglich.

– Als drittes folgt eine Diskussion der wichtigsten Abgrenzungen zu Konzepten anderer Klassen. Hierbei werden in der Regel Paare von Konzepten gegenübergestellt. Das Ziel dieser Gegenüberstellungen besteht allerdings nicht darin, einen vollständigen Vergleich der Konzepte bzw. ihrer Einsatzszenarien zur Verfügung zu stellen. Vielmehr geht es darum, die im Laufe der Untersuchungen identifizierten Abgrenzungsprobleme zu erörtern. Diese Probleme betreffen in der Regel spezielle Situationen. Diese werden dann detailliert erörtert. Darüber hinaus gilt das Gleiche wie beim vorhergehenden Punkt.

– Danach werden, falls erforderlich, Anmerkungen gemacht, die über die vorher gemachten Erläuterungen hinausgehen. Z.B. wird die Konzeptklasse, falls es möglich oder nötig ist, mit ähnlichen softwaretechnischen Begriffsbildungen verglichen.

Im Zusammenhang mit diesen Darstellungen sollte das Folgende beachtet werden:

– Die ersten drei Punkte beschreiben die jeweiligen Konzepte in der Regel so, wie sie aus den Daten heraus entwickelt wurden, während es erst im vierten Punkt zu expliziten Vergleichen mit verwandten Begrifflichkeiten kommt. Von dieser Form der Erörterung wird dann Abstand genommen, wenn davon ausgegangen werden muss, dass der Leser ein den Erörterungen zum Teil zuwiderlaufendes Vorverständnis besitzt oder „entliehene“ Termini die Darstellung erheblich erleichtern. In diesen Fällen werden Vergleiche mit verwandten Begrifflichkeiten auch in den ersten drei Punkten vorgenommen.

– Im Rahmen der Punkte zwei und drei werden auch Regeln wie die, ob in einem bestimmten Fall eine Doppelkodierung angebracht ist, festgelegt. Diese Regeln helfen dabei, potentielle Schwierigkeiten bei der Anwendung der BKM zu verstehen und zu behandeln, und erleichtern somit ihren Einsatz. Wie bereits angesprochen ist es aber keinesfalls so, dass sie als verbindlich betrachtet werden sollten. Denn so wie vorgesehen ist, dass die BKM bei spezialisierten Untersuchungen erweitert oder beschnitten werden muss, kann und wird es auch notwendig werden, die vorgeschlagenen Regeln gezielt zu ändern. In Abschnitt 9.3 wird hierauf genauer eingegangen.

7.4 Produktorientierte Konzepte

Die Menge der produktorientierten Konzepte ist in zwei Unterklassen aufgeteilt, *design* und *requirement*. Details zu diesen Klassen bzw. zu ihren Elementen werden in den folgenden beiden Abschnitten erläutert.

Abb. 7.3: Die HHI-Konzepte geordnet nach Haupt- und Unterklassen. Einen Sonderfall bildet die Klasse *activity*. Hier beziehen sich die reaktiven Verben nicht auf Äußerungen zu Äußerungen, sondern auf Äußerungen zu HCI/HEI-Aktivitäten. Hinweis: Etwas ablehnen bedeutet hier in der Regel, es als nicht richtig/angebracht zu bewerten.

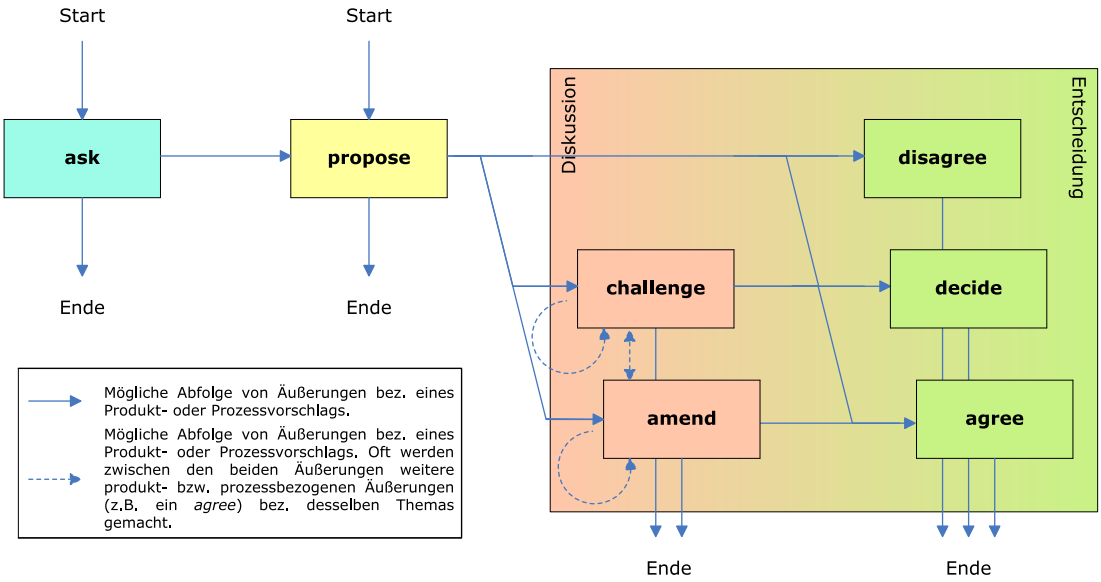


Abb. 7.4: Vereinfachte Darstellung bestimmter prozess- oder produktspezifischer Diskussionsverläufe/Entscheidungsprozesse: Der Startpunkt ist im Allgemeinen ein Vorschlag (*propose*) zu einem bestimmten Produkt- oder Prozessaspekt, z.B. ein konkreter Vorschlag zur Gestaltung des Programmkodes. Diesem kann eine Diskussion oder direkt eine Entscheidung folgen. Bei einer Diskussion wird der Vorschlag entweder erweitert bzw. detailliert (*amend*) oder durch einen Gegenvorschlag ersetzt (*challenge*). Dabei kann die Diskussion über mehrere Iterationsschritte verlaufen, bis eine Entscheidung getroffen wird (siehe z.B. Tabelle 7.7). Bei der Entscheidung wird der Vorschlag dann entweder angenommen (*decide*) oder abgelehnt (*disagree*). Falls der Vorschlag Alternativen enthielt, wird evtl. eine Auswahl getroffen (*decide*).
Eine Diskussion muss allerdings nicht zu einer expliziten Entscheidung führen. Sie kann vielmehr an jeder Stelle enden (auch direkt nach einem *propose*). Außerdem beginnt eine Episode nicht immer mit einem konkreten Vorschlag, sondern in Einzelfällen auch mit einer offenen Frage, die keinen Vorschlag beinhaltet (*ask*). Neben den abgebildeten Abläufen wurden auch eine Reihe von Variationen beobachtet. Beispielsweise muss eine Diskussion nicht bei einem *disagree* oder *agree* enden. Der Diskussionsverlauf kann auch nach einem solchen weiter laufen. Achtung: Das Modell abstrahiert von Personen (*P1* bzw. *P2*). Hiermit wird der Tatsache Rechnung getragen, dass aufeinander folgende Diskussionsschritte nicht unbedingt von verschiedenen Personen geäußert werden müssen, dass es z.B. vorkommt, dass ein Sprecher seinen eigenen Vorschlag korrigiert. Darüber hinaus vereinfacht die Abbildung Diskussionsverläufe auch noch bez. einer Reihe anderer Aspekte: (1) Obwohl es regelmäßig vorkommt, dass innerhalb von Diskussionen auch ganz andere Äußerungen, z.B. solche des Typs *knowledge* gemacht werden, sind diese in der Abbildung nicht berücksichtigt. (2) Von der Tatsache, dass nicht für alle Objektklassen dieselben Verben aus den Daten extrahiert worden sind, wird abstrahiert. Beispielsweise wurde für die Klasse *strategy* kein *disagree* beobachtet (siehe Abbildung 7.3). (3) Es ist keineswegs immer so, dass sich eine Äußerung nur auf *eine* andere, z.B. die unmittelbar vorhergehende bezieht. Oft sind die Bezüge komplexer. Ein diesbezügliches Beispiel ist in Abbildung 7.6 zu finden.

7.4.1 *design*-Konzepte

Fokus der *design*-Konzepte

Wie Tabelle 7.1 und Bild 7.3 zu entnehmen ist beziehen sich *design*-Konzepte auf Äußerungen, die aktuell zu entscheidende Gestaltungsmöglichkeiten von Programmartefakten adressieren. Hierbei wird der Designbegriff in einer sehr allgemeinen Weise verwendet: Jeder Gestaltungsvorschlag in Bezug auf Programmartefakte oder die Anordnung von Programmartefakten ist ein Designvorschlag, unabhängig davon, ob er sich auf eine konkrete Programmzeile (bzw. einen Teil dieser), die übergeordnete Architektur der in Entwicklung befindlichen Software oder irgendeinen Aspekt zwischen diesen beiden Ebenen bezieht.

So werden u.a. folgende Typen von Äußerungen der Klasse *design* zugerechnet:

- Vorschläge zur Gestaltung des Programmkodes, z.B. bez.
  - der Benennungen von Variablen oder Methoden,
  - der Beschaffenheit von Signaturen,
  - der Wahl von Kontrollstrukturen,
  - der Anordnung von Kodeblöcken,
  - der Nutzung von Methoden oder
  - der Festlegung der Paketstruktur.
- Vorschläge bez. zu verwendender Algorithmen.
- Vorschläge zum Entwurf von SQL-Statements.[33]
- Vorschläge bez. des Inhaltes oder der Struktur von Konfigurationsdateien.

Äußerungen bez. festgeschriebener oder auch festzuschreibender Anforderungen an das Programm wie auch solche bez. Entwurfsentscheidungen, die vor der PP-Sitzung gefällt wurden, werden hingegen nicht dem Design zugerechnet. Sie gehören zur Klasse *requirement*.[34] Tabelle 7.4 listet eine Reihe von Beispielen für *design*-Äußerungen auf.

Identifizierte *design*-Konzepte und ihre speziellen Eigenschaften

In Bezug auf Designaspekte wurden die HHI-Aktivitäten *propose_design*, *agree_design*, *decide_design*, *disagree_design*, *amend_design*, *challenge_design* und *ask_design* beobachtet. Sie sind in Abbildung 7.3 grob dargestellt. Weitergehende Charakterisierungen werden im Folgenden beschrieben.

[33] SQL ist die Abkürzung für *Structured Query Language*. Hierbei handelt es sich um eine Sprache zur Definition, Abfrage und Manipulation von Daten in relationalen Datenbanken.

[34] Im Rahmen der BS/BKM ist für eine entsprechende Kodierung nur notwendig, dass die Entwickler eine solche Anforderung diskutieren. Eine Umsetzung muss nicht erfolgen. Auch eine (schriftliche) Ausformulierung ist nicht notwendig.

#	Session:Kode	Äußerung	Kontext/Kommentar
1	ST1.1: *P1.propose_design*	„'ne `TopicConnection` brauchen wir.“	*P1* editiert eine gerade komplett kopierte Methode. Die ursprüngliche Methode versendete ein Objekt mittels einer JMS[a]-Queue. Die Kopie soll nun so angepasst werden, dass ein Objekt an ein JMS-Topic übergeben wird. In diesem Zusammenhang schlägt *P1* vor, die Deklaration einer `QueueConnection` in die einer `TopicConnection` zu ändern.
2	PR1.1: *P1.propose_design*	„Und‿wie‿nennen‿wir‿das‿Ding?‿`$id`?“	*P1* ist Driver. Er schlägt die Benennung einer Variablen vor. Die einleitende Frage wird nicht separat als *ask_design* kodiert, da *P1* diese unmittelbar durch eine mögliche Antwort ergänzt und nachfolgend sofort mit dem Tippen beginnt.
3	PR1.1: *P2.agree_design*	„[Â]hmm.“	Zustimmung zum Vorschlag, eine Variable mit `$id` zu bezeichnen. Die Zustimmung bezieht sich auf die unter Beispiel 2 aufgeführte Äußerung.
4	PR1.1: *P2.propose_design*	„Was woll'n wir‿zurückgeben? Wollen‿w[i]r (!...!) (.) `getFriendsLastChange`.‿Wir wollen doch `timestamp` zurückkriegen, ne? Oder wollen wir True oder False? (∼∼)“	Der Observer *P2* macht Alternativvorschläge für Rückgabewerte. Er formuliert die Auswahl als Frage. Dabei ist die einleitende unkonkrete Frage rhetorisch. Ihr folgt direkt eine Antwort. Sie wird deshalb nicht einzeln als *ask_design* kodiert.
5	PR2.1: *P2.propose_design*	„Genau, das muss ganz raus.“	*P2* ist Driver. Er bezieht sich auf einen im Editor blau hervorgehobenen Methodenaufruf, den er entfernen möchte.[b]
6	PR2.1: *P1.propose_design*	„Sondern wir müssen das `FeatureProxy` benutzen, um die Werte abzuholen.“	*P1* ist Observer. Er schlägt vor, auf welche Weise auf bestimme Werte zugegriffen werden soll.

[a] Siehe Fußnote 21 auf S. 90.

[b] Mit dem einleitenden „genau“ bezieht sich *P2* nicht auf eine vorhergehende Äußerung, sondern betont den Erkenntnischarakter seines Vorschlags.

Tab. 7.4: Beispiele aus den Sessions ST1.1, PR1.1 und PR2.1 für Äußerungen, die mit *design*-Konzepten annotiert wurden. Es wird deutlich, dass die Äußerungen oft nur in ihrem Kontext richtig verstanden bzw. interpretiert werden können (siehe S. 136). Dies betrifft besonders solche, die sich auf andere, im Gesprächsverlauf vorhergegangene beziehen (siehe Beispiel 3). Aber auch Vorschläge, die als Reaktion auf Reize aus der Umgebung (z.B. Ausgaben in der Entwicklungsumgebung) gemacht werden, können oft nur im Kontext richtig zugeordnet werden (siehe Beispiel 5).

1. **Vorschlagstypen/Ziele von Vorschlägen:** Im Allgemeinen beginnen Episoden, in denen die Entwickler Designaspekte diskutieren, mit einem *propose_design* (siehe Abbildung 7.4), also einem konkreten Vorschlag zur Gestaltung der in Arbeit befindlichen Programmartefakte.

   Hierbei ist es, wie bereits in den Beispielen in Tabelle 7.4 gesehen, keineswegs immer so, dass die Sprecher ihre Vorschläge explizit als solche markieren. Stattdessen wird in der Regel, wie in Abschnitt 7.1.2 erläutert, auf das performative Verb verzichtet und nur der Vorschlag selbst in Form einer Feststellung bzw. Behauptung (als Deklarativsatz) artikuliert (siehe Beispiel 1 in Tabelle 7.4) bzw. eine Formulierung in Frageform gewählt (siehe Beispiel 2 in Tabelle 7.4). Hierbei kann die Frage auch nachgeschoben sein, z.B. durch ein „O.K.?“.

   Darüber hinaus ist oft nicht ohne weiteres, d.h. bestenfalls unter Berücksichtigung des Kontextes, zu erkennen, ob ein Sprecher seine Idee überhaupt als zu diskutierenden Vorschlag betrachtet oder es ihm eigentlich nur darum geht, den Partner über bereits getroffene Entscheidungen zu informieren bzw. diesbezügliche Anweisungen zu geben, ohne dass er eine (verbale) Reaktion erwartet bzw. wünscht.[35] So kommt es beispielsweise vor, dass die Driver unmittelbar nach, manchmal sogar schon während des Formulierens mit der Umsetzung ihres Vorschlags beginnen (siehe Beispiel 2 in Tabelle 7.4).

   Die BKM abstrahiert von diesen (illokutiven) Variationen, d.h. sie erfasst nicht, welche Art der Formulierung ein Sprecher wählt bzw. welche Ziele er in Hinsicht auf eine nachfolgende Diskussion oder Bestätigung seines Designvorschlags verfolgt bzw. zu verfolgen scheint. Alle derartigen Phänomene werden unter dem Konzept *propose_design* (bzw. ggf. *amend_design* etc.) subsumiert.[36]

   Dies bedeutet, dass das Konzept eine große Bandbreite von Äußerungen adressiert. Um diese im Auge zu behalten, sollten die beiden folgenden Punkte beachtet werden:

   (a) **Notwendige und optionale Bestandteile einer Äußerung vom Typ *propose_design*:** Potentiell kann ein *propose_design* drei verschiedene Arten von Informationen transportieren:

[35] Auch wenn der Sprecher durch die Formulierung seines Vorschlags suggeriert, dass er diesen ohne weitere Diskussion umsetzen will bzw. umgesetzt haben möchte, bietet er allein dadurch, dass er den Vorschlag überhaupt verbalisiert, eine Gelegenheit zum Widerspruch. In vielen Situationen nutzt der Partner diese dann auch.

[36] Hier spiegelt sich unter anderem die im Laufe des Herleitungsprozesses getroffene Entscheidung wieder, nach der das Hervorheben von Designvorschlägen Priorität hat. Spezielle Aspekte solcher Vorschläge können in nachfolgenden Untersuchungen leicht hinzugefügt werden.

i. Einen (neuen) Designaspekt, also etwas, was zur weiteren Gestaltung der Programmartefakte beiträgt.[37]
ii. Eine zumeist positive Bewertung dieses Aspektes durch den Sprecher.[38]
iii. Den Wunsch, dass der Partner den geäußerten Designaspekt bewerten soll.

Im Rahmen der BKM ist nur das Vorhandensein des Designaspektes notwendig dafür, dass eine Äußerung mit dem Konzept *propose_design* annotiert werden kann. Das Vorhandensein des Designaspektes ist sogar hinreichend, wobei es vorkommen kann, dass der Designvorschlag selbst nur referenziert und nicht ausformuliert wird (Details hierzu auf S. 172). Die Informationen zwei und drei sind in Hinsicht auf eine solche Konzeptualisierung optional. Eine Äußerung vom Typ *propose_design* kann sie enthalten, muss dies aber nicht.

Somit ergeben sich drei Typen von *propose_design*-Äußerungen:

- *Typ 1 (FO): Weitere Meinung einholen* (Information i, ii und iii werden übermittelt).
- *Typ 2 (PI): Information liefern oder Anweisung geben* (Nur Information i und ii werden übermittelt).
- *Typ 3 (SO): Orientierung suchen* (Nur Information i und iii werden übermittelt).

Sie werden in Tabelle 7.5 detailliert erläutert.

Rein rechnerisch ergeben sich sogar vier Typen von *propose_design*-Äußerungen. Allerdings scheint es wenig wahrscheinlich, dass ein Entwickler eine *propose_design*-Äußerung formuliert, die er selbst in keiner Weise beurteilt und bez. derer er sich auch keine Beurteilung durch seinen Partner wünscht. Ein solches Phänomen konnte zumindest bisher nicht beobachtet werden.

(b) ***propose_design* und indirekte Sprechakte[39]:** Die gerade dargestellten Typen von *propose_design*-Äußerungen stellen unter *propose* zu subsumierende Intentionen[40] der Sprecher dar. Sie können nicht in jedem Fall direkt aus der Art der Formulierung selbst, also z.B. aus

[37] Dieser kann auch negativ formuliert sein, also z.B. derart, dass geäußert wird, dass etwas eine bestimmte Eigenschaft *nicht* besitzen soll (siehe z.B. Zeile 2 in Tabelle 7.7).

[38] Vgl. auch mit dem Begriff der *propositionalen Einstellung* – z.B. bei Lenzen [117].

[39] Die in diesem Abschnitt referenzierte Sprechakttheorie beeinflusste die Herleitung der BS nicht. Im Sinne der GTM wurde sie erst gegen Ende der hier dargestellten Analysen berücksichtigt. Hierbei wurde ein Abgleich bestimmter Ideen der Sprechakttheorie mit den Daten und dem Bedeutungsumfang einzelner Konzepte vorgenommen. In diesem Zusammenhang kam es zu keinen relevanten Änderungen der BS/BKM.

[40] Der Begriff Intention adressiert hier den Zweck, den der Sprecher mit seiner Äußerung verfolgt. Dieser ist abzugrenzen von den tatsächlichen Folgen seiner Sprechhandlung (siehe hierzu auch [226, S. 37]).

dem Satzbau (Syntax) oder den verwendeten Begrifflichkeiten, abgelesen werden. Beispielsweise besteht die Intention eines Fragesatzes nicht immer, zumindest nicht immer ausschließlich, darin, eine Antwort auf die gestellte Frage zu bekommen (siehe Beispiel im Exkurs 6). Eine Formulierung in Frageform schließt also weder aus, dass es sich um einen Designvorschlag handelt, noch dass dieser vom Typ 2 ist.

Searle (siehe Exkurs 6) spricht an dieser Stelle von *indirekten Sprechakten*. Diese bezeichnen – grob gesprochen – solche Äußerungen, bei denen das Gesagte auf zwei unterschiedliche Weisen verwendet wird, z.B. nicht wörtlich (*primär*) als Vorschlag und wörtlich (*sekundär*) als Frage (siehe Beispiel 2 in Tabelle 7.4).

Die BKM adressiert in der Regel[41] nur den primären illokutionären Akt.[42] Dies bedeutet, dass es im Rahmen der BKM bei einer Äußerung, die einen neuen Designaspekt ins Spiel bringt, nicht darauf ankommt, ob sie als Frage oder Aufforderung formuliert ist. In beiden Fällen wird sie mit *propose_design* annotiert. Auf eine darüber hinausgehende Klassifizierung wird verzichtet.

Dies hat vor allem folgenden Grund: Im Rahmen der GTM geht es nicht um eine vollständige Beschreibung aller irgendwie beobachtbaren Phänomene. Die BKM ist vielmehr als ein grobes Schema konzipiert, mit dessen Hilfe der Blick auf ein bestimmtes Interessenfeld gelenkt werden kann. Spezielle Aspekte, also solche, die potentiell nur für wenige, auf bestimmte Teilbereiche fokussierte Untersuchungen interessant sind, werden nicht adressiert.

Es sollte aber beachtet werden: Da die BS/BKM nur einen Ausgangspunkt darstellt, kann es bei spezialisierten Untersuchungen wichtig werden, auch den sekundären illokutionären Akt zu berücksichtigen und in die Analyse einzubringen. Hierbei kann es sinnvoll sein, mehrere Basiskonzepte zur Konzeptualisierung eines Phänomens einzusetzen.

Da Äußerungen von Typ *challenge_design* und *amend_design* in Teilen eine ähnliche Natur wie Äußerungen vom Typ *propose_design* besitzen – sie enthalten immer einen Vorschlag bzw. eine Vorschlagspräzisierung – sind die gerade dargestellten Überlegungen auch bez. der Identifikation dieser

[41] Es gibt Situationen, in denen auch im Rahmen der BKM eine Kodierung sowohl des primären wie auch des sekundären illokutionären Aktes angezeigt ist. Dies gilt vor allem dann, wenn nicht wirklich eindeutig zwischen beiden differenziert werden kann (siehe z.B. die Diskussion *disagree_step* vs. *explain_knowledge*/*explain_finding* auf S. 205). Für bestimmte Äußerungen ist es im Rahmen der BS/BKM sogar in keiner Weise möglich, den primären illokutionären Akt zu adressieren. Hierauf wird aber erst später eingegangen (siehe z.B. S. 316).

[42] In diesem Zusammenhang stellt sich die Frage, wie oder mit welchen Einschränkungen der primäre illokutionäre Akt überhaupt als solcher erkannt werden kann (vom Partner wie auch vom Forscher). Eine erschöpfende Erörterung dieses Themengebietes würde den Rahmen der vorliegenden Arbeit aber bei weitem sprengen. Ansatzweise wird hierauf in Abschnitt 9.3 eingegangen. Details werden z.B. von Searle in [190] diskutiert.

Phänomene gültig. Die Überlegungen lassen sich außerdem auf eine Reihe anderer Klassen übertragen, in denen Vorschläge artikuliert werden (z.B. *step* oder *strategy*).

2. **Auf laufende oder abgelaufene Editierschritte verweisen:** Es sind zwei Fälle zu unterscheiden:

   (a) **Informieren, ohne den eigentlichen Vorschlag zu verbalisieren (Untertyp von *PI*)**: In Tabelle 7.5 ist erläutert, dass Vorschläge nicht immer mit dem Ziel gemacht werden, sie zu diskutieren oder vom Partner bewerten zu lassen. Manchmal werden sie nur deshalb geäußert, um das Gegenüber darüber zu informieren, was man für richtig hält und dementsprechend umsetzen wird oder gerade umsetzt. Ein Extremfall dieses Vorschlagtyps tritt auf, wenn der Sprecher den Vorschlag gar nicht mehr ausformuliert, sondern beim Editieren nur noch darauf hinweist, dass es genauso zu machen ist, wie er es gerade tut. So äußert in Sitzung PR2.1 *P2*, während er eine Methode modifiziert: „Eigentlich müsste das *so* implementiert sein“ (Details zum Kontext dieser Äußerung sind im Abschnitt 7.7 (*activity*-Konzepte) auf S. 345 zu finden).

   (b) **Meinung einholen, ohne den eigentlichen Vorschlag zu verbalisieren (Untertyp von *FO*)**: Nicht alle Änderungen, die am Kode vorgenommen werden, sind im Vorfeld im Paar diskutiert und beschlossen worden. Vielmehr ist es so, dass oft (große) Teile des Kodes vom Driver geschrieben werden, ohne dass der Partner verbal einbezogen wird. Allerdings kommt es in diesen Fällen vor, dass der Observer unmittelbar danach angesprochen bzw. befragt wird. So ruft *P1* in Sitzung PR2.1 einen Wizard auf, mit dem ein neues Java-Paket (*Java Package*) angelegt werden kann. Nachdem er den Paketnamen eingegeben, aber bevor er die *Finish*-Schaltfläche bedient hat, fragt er „Richtig?“ (Details zum näheren Kontext werden im Rahmen von Erörterungen zum Konzept *think aloud_activity* in Tabelle 7.53 erläutert.). Äußerungen dieser Art können als Vorschläge interpretiert werden, in denen der eigentliche Vorschlagsinhalt nicht verbal, sondern durch das Schreiben von Kode oder ähnlichem vermittelt wird, und sollten deshalb als *propose_design* annotiert werden.[43] [44]

[43] An dieser Stelle stellt sich die Frage, wie die Äußerung zu werten wäre, wenn *P1* seine Frage erst nach dem Betätigen der *Finish*-Schaltfläche gestellt hätte. Derartige Phänomene wurden im Rahmen der hier berücksichtigten Analysen nicht beobachtet. Die Beantwortung der Frage muss also weiterführenden Studien überlassen werden.

[44] Gegen Ende der vorliegenden Arbeit (S. 429) wird darauf eingegangen, warum Wissenstransfer, der ohne Verbalisierungen stattfindet im Rahmen der BS/BKM nicht erfasst wird.

#	Typ	Eigenschaften (E_i) des Typs/Kommentar	Beispiel	Kommentar zum Bsp.
1	Weitere Meinung einholen (*further opinion* (*FO*))	**E_1: Sprecher bewertet seinen Vorschlag.** **E_2: Sprecher wünscht Bewertung durch den Partner.** Die Bewertung durch den Sprecher kann positiv, nur eingeschränkt positiv oder auch negativ sein. Sie ist nicht immer explizit formuliert. In vielen Fällen wird aber, spätestens bei Berücksichtigung des Kontextes, auch ohne eine solche Explikation erkenntlich, dass der Sprecher seinen Vorschlag zumindest als eine „in gewissem Maße geeignete" Möglichkeit betrachtet (z.B. durch Phrasen wie „ich würde" oder „man müsste"). Auch der Wunsch nach Bewertung durch den Partner muss nicht explizit, z.B. durch Verwendung einer Frageform, artikuliert sein.	PR2.1: „Also ich würd', im Endeffekt würd' ich das so umstellen, dass das `FeatureProxySet` (.) diese `TableModel`-Eigenschaften (.) verliert. Dass du aus dem `FeatureProxySet` (.) nur noch das `Feature-Proxy` abholst."	Der Driver macht einen Vorschlag zum „Refactoring". Er betont, dass er seinen Vorschlag wie formuliert umsetzen würde. Gegen Ende seiner Äußerung wendet er sich seinem Partner zu und sieht diesen fragend an.
2	Information liefern/ Anweisung geben (*provide information* (*PI*))	**E_1: Sprecher bewertet seinen Vorschlag.** **E_2: Sprecher scheint keine Bewertung zu wünschen.** Der Sprecher ist entweder Driver und somit in der Lage seinen Vorschlag ungeachtet der Meinung des Partners direkt umzusetzen (*Information*), oder er ist Observer, überzeugt von seiner Idee und geht davon aus, dass der Partner es nicht besser weiß bzw. wissen kann (*Anweisung*, z.B. in Form eines Aufforderungssatzes). Der zweite Fall ist nicht immer eindeutig vom Typ 1 zu unterscheiden. Vor allem dann nicht, wenn der Partner widerspricht.	PR2.1: „Du musst die überschreiben."	Der Observer fordert den Driver *P*1 auf, eine geerbte Methode zu überschreiben. Obwohl es sich um eine Anweisung handelt, führt *P*1 diese nicht durch. Stattdessen schlägt er vor, sich zuerst die geerbte Klasse anzusehen.
3	Orientierung suchen (*search for orientation* (*SO*))	**E_1: Sprecher bewertet seinen Vorschlag nicht.** **E_2: Sprecher wünscht Bewertung durch den Partner.** Der Sprecher kann oder will nicht einschätzen, ob oder inwieweit seine Idee bez. der Gestaltung des Designs brauchbar ist und wünscht eine Bewertung durch den Partner (z.B. in Form eines Fragesatzes).	PR1.1: „Die Frage ist, wollen wir dann überhaupt etwas returnen?"	Details hierzu werden in Tabelle 7.6 erörtert.

Tab. 7.5: Typisierung von *propose_design*-Äußerungen nach Subintentionen, also in gewisser Weise spezifischer formulierten (primären) Illokutionen. Hierbei muss beachtet werden, dass die (Sub-)Intention des Sprechers nicht immer direkt anhand der Formulierung des Vorschlags identifiziert werden kann. In der Regel ist es notwendig, zusätzlich den Kontext heranzuziehen. Die erläuterten Typen lassen sich, zumindest im Kontext von Prozess- oder Produktdiskussionen, auf andere Objektklassen wie z.B. *step* (siehe S. 194) oder Verbklassen wie z.B. *decide* (siehe Beispiel in Tabelle 7.22) übertragen.

3. **Begründete Vorschläge:** Phänomene vom Typ *propose_design* treten häufig im Zusammenhang mit einer Begründung auf. Diese sollte separat als *explain_knowledge* bzw. ggf. als *explain_finding* kodiert werden (siehe Abschnitt 7.6). Gleiches gilt auch für Phänomene vom Typ *challenge_design*, *amend_design* und *disagree_design*.[45]

   Beispielsweise widerspricht der Observer in PR1.1 einem Designvorschlag mit den Worten:

   > „Aber◡das◡ist◡doch◡unschön. Dann haben wir zwei Schleifen, das ist doch Mist."

   Der Ablehnung (*disagree_design*) folgt eine, wenn auch nicht sehr präzise, Begründung.[46]

   Auf S. 180 sowie bei der Darstellung der Klasse *knowledge* (siehe S.331) wird noch einmal auf die Differenzierung zwischen *explain_knowledge* und *propose_design* eingegangen.

4. **Zustimmung vs. Auswahl:** Das Auftreten eines mit dem Konzept *decide_design* annotierbaren Phänomens setzt voraus, dass im Vorfeld zumindest zwei unterschiedliche Designoptionen vorgeschlagen wurden. Bei einer *decide_design*-Äußerung handelt es sich dann um eine explizite Auswahl zwischen den Alternativen und nicht nur um eine Zustimmung zum z.B. zuletzt genannten Vorschlag.

   Diese Differenzierung zwischen *decide_design* und *agree_design* ist in der Regel nur dann möglich, wenn die betroffenen Gestaltungsoptionen zusammen in einer Äußerung formuliert wurden. Außerdem kann nur dann von einem *decide*-Phänomen gesprochen werden, wenn zum ersten mal eine Auswahl aus einer Menge vorgeschlagener Alternativen getroffen wird. Wird eine solche Auswahl hingegen nachfolgend vom Partner abgelehnt, indem dieser eine andere Option auswählt, sollte nicht *decide*, sondern *challenge* annotiert werden. Ein diesbezügliches Beispiel, allerdings die Klasse *strategy* betreffend, ist in Tabelle 7.22 zu finden.

5. **Präzisierungen erkennen:** Das Konzept *amend_design* wird vergeben, wenn ein bereits geäußerter Vorschlag konkretisiert, d.h. Teilaspekte präzisiert oder für die Umsetzung nötig erscheinende Ergänzungen hinzugefügt werden. Für den Forschenden kann an dieser Stelle ein Problem entstehen:

[45] Natürlich kann es auch vorkommen, dass Phänomene vom Typ *agree_design* und *decide_design* begründet werden. Dies gilt für alle *agree*-, *decide*- oder *disagree*-Äußerungen.

[46] Zumindest der erste Teil der Begründung hat die Form eines *explain_finding*s (siehe S. 250f). Ob es sich beim Nachsatz auch um ein solches oder eher um die Formulierung von allgemeinem Programmierwissen handelt, dieser also separat als *explain_knowledge* zu kodieren wäre, ist hier kaum zu entscheiden. Derartige Situationen werden auf S. 331 erörtert.

Wenn er, wie bei der Analyse von Sitzungen aus einem professionellen Umfeld nicht unwahrscheinlich, den ursprünglichen Vorschlag bzw. die Ergänzung oder deren Bedeutung für den Entwicklungskontext nicht vollständig versteht, kann es für ihn schwierig werden, zwischen *amend_design* und *propose_design*, also zwischen der Erweiterung eines Designvorschlags und einem komplett neuen Vorschlag, zu unterscheiden. Denn die beobachteten Entwickler selbst weisen nur selten explizit darauf hin, dass sich ihre Äußerung auf einen vorher gemachten Designvorschlag bezieht (siehe Beispiel in Tabelle 7.7). Vielmehr gehen sie in der Regel davon aus, dass ihr Partner den Zusammenhang ohne weitere Erklärung erkennen kann (siehe S. 136).

6. **Eigene Vorschläge ergänzen oder widerrufen:** Genau wie Äußerungen vom Typ *amend_design* (siehe Tabelle 7.7) können sich auch die vom Typ *challenge_design* nicht nur auf Vorschläge des Partners, sondern auch auf die eigenen beziehen. Der Sprecher korrigiert sich in diesen Fällen also selbst.

   Dies gilt potentiell für alle *challenge*- sowie *amend*-Konzepte der BKM. In Abschnitt 9.3.9 wird hierauf noch einmal eingegangen.

7. **Zustimmung vs. Aufmerksamkeit anzeigen:** Viele im Kontext einer Diskussion über Design gemachte Äußerungen, die potentiell affirmativen Charakter haben, also mit *agree_design* konzeptualisiert werden könnten, sind sehr kurz. Sie beschränken sich auf unterschiedliche Formen von „O.K.“, „Genau“, „Ja“ oder Partikel[47] wie „[A]hm“. Dies hat zur Folge, dass nicht immer eindeutig entschieden werden kann, ob es sich wirklich um eine explizite, bewusste Zustimmung handelt oder um das, was in den Sprachwissenschaften den Namen *Back-channel* trägt, ein von Yngve [230] eingeführter Terminus „zur Bezeichnung von sprachlichen Ausdrücken und entsprechenden nicht-sprachlichen Signalen (z.B. hm, ja oder Kopfnicken), mit denen der Hörer dem Sprecher seine Aufmerksamkeit und seine Zuhörerbereitschaft anzeigt. [Wobei] die jeweilige ‚lokale' Bedeutung ebenso wie der Status dieser Ausdrücke und Signale als Turn[48] [umstritten sind]“ [37].

   Die BKM differenziert an dieser Stelle nicht. Sobald eine Äußerung potentiell Zustimmung zu einem Designvorschlag signalisiert, wird sie mit *agree_design* annotiert. Dies gilt analog für die *agree*-Phänomene in allen anderen Bereichen, z.B. für *agree_step* oder *agree_knowledge*.

   Aus dieser Entscheidung folgt, dass im Rahmen der BKM potentiell jedes in den Redefluss des Partners eingestreute (und evtl. leicht zu überhörende) „[Â]hm“ eine zu annotierende Affirmation darstellen kann, ohne dass der Grad der Zustimmung hierbei in irgendeiner Weise berücksichtigt wird. Eine

[47] Unflektierbare Wortart.

[48] Unter einem *Turn* versteht man in der Gesprächsforschung einen Gesprächsschritt. Er bildet nach Brinker und Sager die Grundeinheit eines Dialogs [31, S. 58] und wird von Goffman [80, S. 201] „als das, was ein Individuum tut und sagt, während es an der Reihe ist“ definiert.

Differenzierung wird nachfolgenden Untersuchungen überlassen. Hier sollte versucht werden, mittels zum Blickwinkel passender Kriterien den diesbezüglichen Annotationsaufwand von vornherein zu kanalisieren und damit zu beschränken.[49]

Achtung: Auf detailliertere Charakterisierungen von *agree_design* bzw. *agree_design*-Phänomenen, ähnlich denjenigen, die im weiteren Verlauf bez. anderer *agree*-Konzepte wie z.B. *agree_finding* (siehe S. 277f) gemacht werden, wurde im Rahmen der hier vorgestellten Untersuchungen verzichtet. Diese Entscheidung wurde auf Basis der Erfahrungen bei der Kodierung getroffen. Hier zeigte sich, dass davon auszugehen ist, dass derartige Zustimmungen auch ohne weitere Beschreibungen hinreichend gut identifizierbar sind. Dies bedeutet aber keineswegs, dass alle weiterführenden Untersuchungen auf solche verzichten können. Dies gilt für alle $P\&P$-Konzepte vom Typ *agree*.

8. **Kurze Negationen:** Bei manchen sehr kurzen (eingestreuten) Äußerungen wie z.B. dem Partikel „Hm“ scheint es sich, wenn man die Betonung/Intonation (sowie den Kontext) berücksichtigt, um Ablehnungen zu handeln. Solche Äußerungen müssen im Rahmen der BS/BKM mit *disagree_**[50] annotiert werden. Hierbei gelten analoge Überlegungen zu den oben bez. *agree_** gemachten. Ob und wenn ja wie hierbei hinreichend gut zwischen Zustimmungen und Ablehnungen bzw. Back-channels unterschieden werden kann (und in der Folge dann auch soll), wird weiterführenden Studien überlassen (Beispiele hierzu werden in Tabelle 7.47 ab S. 327 diskutiert).

9. **„Vorschlagsfreie“ Fragen:** Die zentrale Eigenschaft von mit dem Konzept *ask_design* annotierten Äußerungen besteht darin, dass sich diese zwar auf einen bestimmten Designaspekt beziehen, der Sprecher über dessen Ausgestaltung aber keine konkrete Vorstellung hat bzw. äußert. Es geht ihm vielmehr darum, den Partner nach diesbezüglichen Vorschlägen zu befragen. In Tabelle 7.6 ist ein *ask_design*-Phänomen erläutert.

[49] Eine Beschränkung kann z.B. dadurch erfolgen, dass man nur solche Höreräußerungen berücksichtigt, die über das reine Rückmeldeverhalten hinausgehen, z.B. solche, „die einen kurzen Kommentar, eine sog. Einstellungsbekundung – etwa in Form eines Zwischenrufs oder dergleichen – ausdrücken (z.B. *das ist ja interessant*, *das glaube ich nicht*, *ach Gott* usw.)“ [31, S. 59].

[50] Der Stern (*) dient hier als Platzhalter für ein beliebiges im Kontext der BKM passendes Objekt. Die Verwendung des Sterns bedeutet allerdings nicht, dass für alle resultierenden Konzepte bereits Phänomene beobachtet werden konnte, sondern nur, dass solche potentiell denkbar sind. Aus Platzgründen wird in der Regel darauf verzichtet, aufzuzählen, in welchen Fällen wirklich derartige Beobachtungen gemacht werden konnten. In der Regel gilt aber: Wenn der Stern im Zusammenhang mit einem $P\&P$-Objekt verwendet wird, bezieht er sich auf $P\&P$-Objekte, bzw. wenn er im Zusammenhang mit einem Element aus der Menge der UK verwendet wird, auf die UK.

10. **Eingeschränkte Ablehnungen:** Ablehnungen von Designvorschlägen (oder anderen prozess- oder produktorientierten Vorschlägen) sind nicht immer derart formuliert, dass zu erkennen ist, dass der Sprecher hundertprozentig hinter seinem Einwand steht. Vielmehr kann es vorkommen, dass ein Sprecher seine Ablehnung explizit als nicht zwangsläufig richtig oder angemessen beschreibt. Beispielsweise reagiert *P2* in Sitzung PR2.1 auf den Vorschlag

    „(~Dies) könnte man da raus ziehen“

    mit der Äußerung

    „(Hm, ja). (..) Ich‿bin‿mir‿nicht‿sicher,‿ob‿ich's (.) da raus ziehen würde.“

    Das Beispiel macht darüber hinaus deutlich, dass es Situationen gibt, in denen es nicht einfach ist, zwischen Ablehnung und Zustimmung zu differenzieren. Schließlich leitet der Sprecher seine Bemerkung mit einem „(Hm, ja)“ ein. Dieser Äußerungsteil könnte für eine Klassifizierung als Zustimmung sprechen. Die Affirmation wird vom Sprecher allerdings nur sehr leise artikuliert, während er sein „Bedenken“ in normaler Lautstärke äußert. Aus diesem Grund wurde an dieser Stelle ein *disagree_design* annotiert.[51] In Abschnitt 9.3.8 wird näher auf derartige Phänomene eingegangen. Hier kann aber schon festgehalten werden, dass weiterführende Untersuchungen zusätzliche Regeln und/oder Konzepte einführen müssen, um zu einer die jeweilige Analyse unterstützenden Kodierung zu kommen.

    Abschließend sei noch bemerkt, dass der weitaus überwiegende Teil der im Rahmen der hier analysierten Sitzungen beobachteten Ablehnungen und Zustimmungen derart formuliert war, dass kein Zweifel des Sprechers ausgemacht werden konnte.

In Tabelle 7.7 wird ein kompletter *design*-Dialog wiedergegeben.

Abgrenzungen von *design*-Konzepten zu anderen Konzepten

Von besonderem Interesse sind die bereits angesprochenen Grenzziehungen zu Konzepten der Klassen *knowledge* und *finding*, vor allem zu *ask_knowledge* und *explain_knowledge* bzw. *explain_finding*:

- ***propose_design* vs. *ask_knowledge*:** Obwohl, wie gesehen, Designvorschläge als Frage formuliert sein können, ist eine Differenzierung zwischen

[51] Inwieweit hier die Verwendung des Konzeptes *standard of knowledge* angebracht ist, wird auf S. 311 diskutiert.

HHI-Kode	Äußerung	Kontext/Kommentar
P1.ask_design	„Was wollen wir denen eigentlich wirklich übergeben?“	Die Frage bezieht sich auf einen Rückgabewert des sich gerade in Überarbeitung befindlichen PHP-Skripts. Mittels dieses Wertes soll der Aufrufende über Veränderungen bestimmter Datenbankinhalte informiert werden. Die Frage von *P1* adressiert allerdings nur den Rückgabewert (bzw. Rückgabetyp) in einen bestimmten, durch eine `if`-Anweisung spezifizierten Fall.
P1.explain_finding	„`NOCHANGE` ist ja auch unnötige Daten“	Der Driver erkennt, dass das Skript im betrachteten Fall die Zeichenkette `NOCHANGE` zurück gibt. Da es häufig aufgerufen werden wird, verweist er darauf, dass diese Implementierung evtl. nicht optimal ist. *P1* begründet damit den Zweck seiner Frage. Dies geschieht unmittelbar nach dem geäußerten *ask_design* und ohne dass er von *P2* diesbezüglich angesprochen wird.
P1.propose_design	„Die Frage ist, wollen wir dann überhaupt etwas returnen?“	Nach drei Sekunden, in denen nichts weiter passiert, macht *P1* den Vorschlag, evtl. gar keinen Wert zurückzugeben. Er transformiert also seine ursprüngliche Frage in einen Vorschlag vom Typ 3.

Tab. 7.6: Beispiel (PR1.1: 14:10:51–16:10:59) für einen Gesprächsverlauf, der mit *ask_design* beginnt. Die Frage leitet eine ca. einminütige Episode ein, von der hier nur die ersten neun Sekunden dargestellt sind. In diesen ersten Sekunden spricht nur der Driver *P1*. Über die gesamte Episode hinweg (also auch in den hier nicht dargestellten Teilen) wird über den Rückgabewert eines PHP-Skriptes diskutiert. Dieses Skript soll in Zukunft von einem externen Dienstleister aufgerufen werden und diesem Informationen aus dem hier in Entwicklung befindlichen System liefern. Die Episode bildet einen Grenzfall zwischen einer Design- und einer Anforderungsdiskussion (siehe hierzu Abschnitt 7.4.2). Da aus der Episode zu erkennen ist (hier nicht im Einzelnen wiedergegeben), dass dem externen Dienstleister (zumindest vorerst) kein Mitspracherecht bez. der Gestaltung der Schnittstelle eingeräumt wird, wurde entschieden, die fraglichen Äußerungen der Klasse Design zuzuordnen. Achtung: Im dargestellten Teil der Episode erreicht *P1* mit seiner offenen Frage nicht das von ihm anvisierte Ziel: *P2* unterbreitet keinen Vorschlag. Somit macht *P1* selbst einen solchen. Er formuliert diesen allerdings vorsichtig als von beiden zu entscheidende Frage.

HHI-Kode	Äußerung	Kontext/Kommentar
P1.propose_step	„Ich würd' (.), lass uns mal das `TableModel` angucken, wie das arbeitet, wie das die Werte abholt."	Bevor *P1* seinen Satz zu Ende gesprochen hat, hat *P2* die angesprochene Klasse (die eigentlich `DisplayNameFeatureProxyTableModel` heißt) im Editor geöffnet.
P1.propose_design	„Also (.) ich (.) bin (.) der Meinung, dass das `TableModel` gar nicht wissen darf oder sollte, dass das (.) gegen virtuelle Spalten oder nicht virtuelle Spalten arbeitet."	*P1* macht den Vorschlag nach einer Pause von ca. 5 Sekunden. In der Pause wird von *P2* nicht gescrollt. Es ist im Wesentlichen der Konstruktor der Klasse zu sehen. Erst während der Formulierung des Vorschlags scrollt *P2* weiter nach unten im Kode.
P2.agree_design	„Ōkayyyy"	*P2* stimmt zu.
P2.explain_standard of knowledge	„Das ist die Abstraktion,‿an die ich vorher gedacht hatte."	*P2* erklärt, dass dies sowieso seine Idee gewesen wäre. Aus dem Kontext geht hervor, dass er wohl schon vor der Sitzung darüber nachgedacht hatte. Währenddessen scrollt er zurück zum Konstruktor.
P2.mumble_sth	„Haste im Prinzip, haste im Prinzip nicht (!...!)."	Satzfragment mit unklarer Bedeutung (evtl. abgebrochene Zustimmung).
P1.amend_design	„Der sollte (...), der sollte aus der Konfiguration noch einen `ValueProvider` abholen"	*P1* konkretisiert seinen Designvorschlag, indem er festlegt, wie die Abstraktion implementiert werden soll.
P2.agree_design	„Haste im Prinzip nicht Unrecht."	*P2* stimmt zu.
P2.explain_standard of knowledge	„Ja, das ist die Abstraktion, an die ich vorher gedacht hatte."	Siehe oben.
P2.mumble_sth	„Das heißt (...) O.K. (!...!)"	Satzfragment mit unklarer Bedeutung.
P1.amend_design	„, dass du,‿dass‿wir die Attributkonfiguration noch um so'n `ValueProvider` erweitern‿und der gibt dann immer das `Feature` (.), der gibt dann immer den‿den‿den‿dis‿dis‿dis `FeatureProxy` rein, um die Werte abzuholen, (.) und dahinter passiert dann der Rest."	*P1* führt nach zweisekündiger Pause den Satz von *P2* weiter, indem er damit fortsetzt, seinen Designvorschlag zu konkretisieren.

Tab. 7.7: Beispiel für eine Episode (PR2.1: 12:55:47–12:56:49), in der das Design von Kode diskutiert wird. Der Observer *P1* schlägt vor, die bereits existierende Implementierung (genau genommen den Konstruktor einer Klasse) zu modifizieren. Der Vorschlag zielt nicht auf die Implementierung neuer Funktionalitäten, sondern auf ein *Refactoring* [65]. Hinweis: Mit dem Begriff Konfiguration bzw. Attributkonfiguration bezeichnet *P1* den Parameter `attributeConfiguration`, der dem Konstruktor des `DisplayNameFeatureProxyTableModel` übergeben wird.

propose_design- und *ask_knowledge*-Phänomenen in der Regel ohne weiteres möglich. Schließlich wurde durch die Einführung der Klasse *design* explizit eine bestimmte Gruppe von Phänomenen aus der Menge der mittels *knowledge*-Konzepten zu annotierenden entfernt (siehe Abschnitt 7.1.1). Anders ausgedrückt: Die BKM wurde so konzipiert, dass eine Trennlinie zwischen „Designwissen“ (bzw. „Designideen“) und anderem explizitem Wissen gezogen werden kann. Fragen, die explizite Designvorschläge zur Diskussion stellen, müssen also mit *propose_design* und nicht mit *ask_knowledge* kodiert werden.[52]

Folgendes sollte in diesem Zusammenhang aber beachtet werden: Fragen vom Typ *ask_knowledge* können, ohne dass dies tatsächlich intendiert ist, darauf hindeuten, dass der Sprecher etwas bestimmtes vor hat. Derartige Äußerungen konnten bisher allerdings nur in Hinsicht auf die Klasse *step* beobachtet werden (siehe S. 204). Sie sind aber auch im Zusammenhang mit der Klasse *design* vorstellbar. Im Abschnitt 9.3.4 über implizierte Handlungsankündigungen wird auf solche Phänomene detaillierter eingegangen.

– ****_design* vs. *explain_knowledge*/*explain_finding*[53]:** Auch in diesem Fall ist die Unterscheidung aufgrund der Konstruktion der BKM auf inhaltlicher Ebene klar definiert: So werden zwar durch Designvorschläge in gewisser Weise fachliches Wissen oder/und neue Ideen übermittelt, diese müssen aber mit *propose_design* und nicht mit *explain_knowledge* oder *explain_finding* annotiert werden. Nur etwaige zugehörige Begründungen sind über *explain_knowledge* oder *explain_finding* zu konzeptualisieren.

  Gleiches gilt im Prinzip auch für alle anderen *design*-Konzepte – insbesondere wenn man das Prinzip der Sichtbarmachung von Bezügen berücksichtigt (siehe S. 161). Folgende Phänomene sollten aber beachtet werden:

  – **Verpackte Designvorschläge**: Designvorschläge können in einer Wissensäußerung „verpackt“ sein, also derart formuliert sein, dass wörtlich genommen der Wissenstransfer im Vordergrund zu stehen scheint. In derartigen Fällen sollte eine Doppelkodierung vorgenommen werden. Im Rahmen der Erläuterungen zur Klasse *activity* wird in Tabelle 7.55 (S. 360) ein Beispiel gegeben.
  – **Verpackte Zustimmungen zu oder Ablehnungen von Designvorschlägen**: Zustimmungen und Ablehnungen können in einer Wissensäußerung „verpackt“ sein. Ein Beispiel, allerdings für die Klasse *step*, ist auf S. 205 (*disagree_step* vs. *explain_knowledge*/*explain_finding*) bzw. in Zeile 2 in Tabelle 7.44 zu finden.

[52] Natürlich sind auch Fälle denkbar, bei denen trotz der hier erörterten Grenzziehung Probleme bei der Kodierung auftreten können. Einen Eindruck von solchen Situationen vermittelt die Diskussion der Abgrenzung zwischen *propose_step* und *ask_knowledge* ab S. 202.

[53] Der Stern (*) dient als Platzhalter für ein beliebiges, im Kontext der BKM passendes Verb. Außerdem: Die Abgrenzung zu *explain_finding* verläuft analog.

Diese Phänomene können, zumindest theoretisch, auch im Zusammenhang mit den Klassen *requirement*, *step*, *strategy* und *todo* auftreten.

Weitere diesbezügliche Anmerkungen folgen im Zusammenhang mit Erläuterungen zur Klasse *knowledge* auf S. 331f.

– ***ask_design* vs. *ask_knowledge*:** Die Abgrenzung verläuft analog zu den anderen Abgrenzungen bez. der Klasse *knowledge*: Jede offene Frage zu einem Designaspekt, die selbst keine konkreten Vorschläge enthält (siehe Beispiel in Tabelle 7.6), wird mit *ask_design* und nicht mit *ask_knowledge* konzeptualisiert.

Die Abgrenzungen von Konzepten der Klasse *design* zu denen der Klasse *knowledge* bzw. *finding* stellt also in der Regel konstruktionsbedingt kaum ein Problem dar. Bei Differenzierungen bez. anderer Konzeptklassen ist die Lage aber nicht ganz so einfach. Dies gilt insbesondere für die Abgrenzungen zwischen

– *propose_design* und *propose_step* sowie

– *propose_design* und *propose_todo*.

Um diese im Detail verstehen zu können, sind vertiefte Kenntnisse der Konzeptklassen *step* bzw. *todo* vonnöten. Sie werden deshalb erst im Zusammenhang mit den Beschreibungen dieser Klassen erörtert (Abschnitt 7.5.1 bzw. Abschnitt 7.5.3).

Weitere Anmerkungen zu *design*-Konzepten

Grob gesprochen wird im Rahmen der „klassischen“ softwaretechnischen Begriffsbildung zwischen der Definitionsphase (Anforderungserhebung und -beschreibung), der Strukturierung des zu entwickelnden Programms (Design) und seiner Implementierung unterschieden (siehe z.B. Balzert [11, S. 686] oder Pfleeger und Atlee [161, S. 223f][54]). Der Begriff Design oder auch Softwaredesign bezeichnet in diesem Zusammenhang nicht nur den Teil des Softwareentwicklungsprozesses, in welchem die erhobenen Anforderungen analysiert und in eine Beschreibung der Struktur des zu entwickelnden Programms umgesetzt werden, sondern auch das aus dem Prozess resultierende Ergebnis, also z.B. den Aufbau der Software aus Komponenten inkl. der Schnittstellen zwischen diesen. Hierbei wird oft zwischen

– Architekturdesign (*software architectural design*), der Identifikation und Inbeziehungsetzung von Komponenten auf höchster Ebene und

[54] Genau genommen unterscheiden Pfleeger und Atlee [161, S. 223f] zwischen zwei Designdokumenten, dem Systemdesign (*system design*) und technischen Design (*technical design*). Ersteres beschreibt dem Auftraggeber, was das zu entwickelnde System leisten wird, und entspricht somit im Wesentlichen dem, was im deutschen Sprachraum als Lastenheft [11, S. 62f] bezeichnet und der Definitionsphase zugerechnet wird. Zweiteres hilft den Entwicklern zu verstehen, welche Hard- und Software benötigt wird, um das Problem des Kunden zu lösen (z.B. Hierarchie und Funktion der Softwarekomponenten).

– detailliertem Design (*software detailed design*), einer im Hinblick auf die Implementierung ausreichenden Beschreibung der einzelnen Komponenten unterschieden (siehe z.B. [1, S. 3-1]).

Die hiermit festgelegte Unterscheidung in separate Teile ist aber nicht wirklich im Sinne agiler Softwareentwicklung, im Speziellen nicht im Sinne von XP bzw. Beck und somit oft auch nicht der PP, da hier im Rahmen des iterativen Vorgehens ein nicht unerheblicher Teil des Designprozesses mit der Implementierung verwoben werden soll. Kent Beck hebt sogar hervor, dass in XP „Design beim Programmieren zum Alltagsgeschäft aller Programmierer“ gehört [13, S. 49].

Hier hakt die Begriffsbildung der BKM ein, da sie die direkt bei der Implementierung stattfindenden Gestaltungsentscheidungen unabhängig von ihrer Reichweite adressiert[55]. Sie schafft somit unter anderem eine Voraussetzung dafür, dass die von XP aufgestellte Behauptung – denn um viel mehr handelt es sich erst einmal nicht – untersucht werden kann, dass es also möglich wird, zu beobachten und zu beschreiben, in welcher Form sich Designaspekte unter verschiedenen Bedingungen im Paarprogrammierungsprozess wiederfinden und welche Auswirkungen dies auf das Endergebnis hat.

7.4.2 *requirement*-Konzepte

Fokus der *requirement*-Konzepte

Die Klasse *requirement* adressiert alle Äußerungen, die sich mit Anforderungen an das zu entwickelnde Programm beschäftigen. Dies können Anforderungen sein,

– die vor dem Start der Programmiersitzung erhoben bzw. festgeschrieben wurden,

– die erst im Laufe der Entwicklung entdeckt bzw. vermisst werden bzw.

– auch solche, die im Laufe der Programmiersitzung temporär als gültig angenommen werden, um z.B. bestimmte Spezialfälle besser verstehen oder testen zu können.

[55] Es muss festgehalten werden, dass der Designbegriff der BKM nicht unter besonderer Berücksichtigung von Erkenntnissen aus bzw. über XP entwickelt wurde. Dies wäre auch gar nicht möglich gewesen, da XP zwar auf Basis von praktischen Erfahrungen der Autoren entstanden ist, bisher aber keine Untersuchungen existieren, die den Designprozess bei der PP genauer untersuchen bzw. explizit charakterisieren. Der *design*-Begriff der BKM ist vielmehr, wie alle anderen Begriffe bzw. Begriffsbeschreibungen der BKM auch, direkt aus den untersuchten Daten heraus entwickelt worden. Deutlich wird dies auch durch die Tatsache, dass ursprünglich nicht der Terminus *design* verwendet wurde, sondern *degree of freedom* (genau genommen *choice about degree of freedom*). Dieser machte klar, dass es sich bei den durch diese Konzepte annotierbaren Phänomenen um jede Form von Gestaltungsmöglichkeiten handelt, die im Rahmen von PP-Sitzungen vorhanden sind. Der Begriff wurde im Laufe des Untersuchungsprozesses nur deshalb durch *design* ersetzt, weil er nicht verdeutlichte, dass es sich um die Gestaltung der Artefakte bzw. Artefaktstruktur und nicht etwa um die Gestaltung des Entwicklungsprozesses handelt.

Requirement-Äußerungen grenzen sich also zu Design-Äußerungen dadurch ab, dass sie sich auf von außen vorgegebene bzw. vorzugebende Eigenschaften des zu entwickelnden Programms beziehen. Diese Eigenschaften können z.B. von einem externen Kunden kommen und sowohl funktional wie auch nicht-funktional sein, also beispielsweise konkret zur Verfügung zu stellende Funktionalitäten (z.B. Dienste) oder das Laufzeit- oder Speicherverhalten der Applikation adressieren. Auch Vorgaben bez. bestimmter Schnittstellen werden im Rahmen der BKM als funktionale Anforderung betrachtet.

Darüber hinaus werden auch solche Äußerungen mit *requirement*-Konzepten annotiert, die sich auf explizite oder implizite Voraussetzungen/Bedingungen beziehen, die durch die zukünftige Laufzeitumgebung des Programms bestimmt werden. Dies bedeutet, dass der Begriff *requirement* im Rahmen der BKM sowohl im Sinne von Anforderung wie auch im Sinne von Voraussetzung oder Bedingung verwendet wird. Hauptgrund hierfür ist, dass sowohl Voraussetzungen wie auch Bedingungen im Allgemeinen automatisch Anforderungen an das zu entwickelnde Programm implizieren. Äußerungen zu Voraussetzungen und Bedingungen sind also zumindest indirekt anforderungsbezogen. Im weiteren Verlauf dieser Arbeit wird im Zusammenhang mit der Klasse *requirement* der Begriff Anforderung synonym für das Begriffstripel Anforderung, Bedingung und Voraussetzung verwendet.

Es muss noch einmal festgehalten werden: Um entscheiden zu können, ob es sich bei einer Äußerung um eine Anforderungs- oder um eine Designaussage handelt, muss ermittelt werden, ob sie sich auf von außen kommende bzw. von außen erwartete Vorgaben oder Bedingungen bezieht oder auf in der Paarprogrammierungssitzung entscheidbare Aspekte. Ist beispielsweise im Vorfeld der Sitzung verbindlich festgelegt worden, dass bei der Entwicklung einem bestimmten Designpattern [70] gefolgt werden soll, so werden Äußerungen, die diese Festlegung rekapitulieren, mit Konzepten der *requirement*-Klasse kodiert. Denn das Paar hat an dieser Stelle keinen Entscheidungsspielraum. Existiert eine solche Festlegung hingegen nicht, werden dieselben Äußerungen mit *design*-Konzepten annotiert.

Die Zuverlässigkeit der Kodierung in Hinsicht auf die Differenzierung zwischen diesen beiden Klassen kann dadurch erhöht werden, dass für die betrachteten Sitzungen die jeweiligen Kontexte, u.a. die im Vorfeld abgelaufenen Diskussionen, möglichst lückenlos ermittelt werden. Dies kann speziell bei Untersuchungen im professionellen Umfeld in der Regel nur dann erreicht werden, wenn der Forscher es schafft, den Entwicklungsprozess über einen längeren Zeitraum, bei XP-Projekten z.B. über eine Iteration hinweg, weitgehend vollständig zu beobachten und zu protokollieren. Der hierdurch entstehende Aufwand ist allerdings erheblich. Es sollte also für jede Untersuchung im Einzelnen abgewogen werden, in welchem Maß diesbezügliche Daten erhoben werden müssen bzw. inwieweit es akzeptabel ist, dass Anforderungen möglicherweise nicht immer als solche erkannt werden können.

In Tabelle 7.8 sind Beispiele für *requirement*-Äußerungen aufgeführt.

#	Session und HHI-Kode	Äußerung	Kontext/Kommentar
1	PR1.1: *P1.propose_requirement*	„O.K. gehen wir mal davon aus, die haben eine Stunde Differenz.“	Der Rückgabewert des in Entwicklung befindlichen Skriptes ist abhängig von der aktuellen Uhrzeit. Es ist aber nicht bekannt, ob das System des Aufrufenden und das System, unter dem das Skript laufen wird, bez. der Uhrzeit synchronisiert sein werden. *P2* macht deshalb eine temporäre Vorgabe, die dem Paar helfen soll, Probleme zu verstehen, die in diesem Zusammenhang entstehen können.
2	PR2.1: *P1.remember_requirement*	„Und die andere Sache, ich hatte es halt so verstanden, dass wir halt versuchen wollen, (.) das [a] über die Fassade nach außen für alle anderen Geschichten unsichtbar zu machen.“	Driver *P1* verweist auf eine Absprache, die allem Anschein nach vor der Sitzung gemacht worden ist. Diese definiert eine Strukturanforderung (Fassade [70]) an das in der Sitzung zu programmierende Modul. Im Rahmen der Entwicklung des Gesamtsystems handelt es sich hierbei um einen Designaspekt. Dieser fungiert allerdings im Rahmen der gerade ablaufenden Sitzung als Anforderung.
3	PR2.1: *P1.remember_requirement*	„Das Einzigste ist, dass <**Entwicklername**> und ich gestern darüber philosophiert haben, dass wir das `FeatureProxySet` (.) ändern. Da soll es 'n `getTableModel` geben.“	*P1* weist auf eine Absprache hin, nach der eine sich im gerade in Bearbeitung befindlichen Kodeteil verwendete Klasse geändert werden soll. Er intendiert, dieses Vorhaben bei den gerade anvisierten Editierschritten zu berücksichtigen.

Tab. 7.8: Beispiele aus den Sessions PR1.1 und PR2.1 für Äußerungen, die mit *requirement*-Konzepten annotiert wurden.

Identifizierte *requirement*-Konzepte und ihre speziellen Eigenschaften

Wie bereits dargestellt wurden im Zusammenhang mit der *requirement*-Klasse die Konzepte *remember_requirement*, *propose_requirement*, *agree_requirement* und *challenge_requirement* beobachtet. Neben den allgemeinen Beschreibungen dieser Konzepte (siehe Abbildung 7.3) sollten die folgenden speziellen Eigenschaften beachtet werden:

1. **Erinnern von Anforderungen:** Das Konzept *remember_requirement* bezieht sich auf Äußerungen, in denen der Sprecher festgeschriebene Anforderungen rekapituliert bzw. dem Paar in Erinnerung ruft. In der Regel handelt es sich dabei um solche Anforderungen, die dem Sprecher selbst entfallen waren und an die er sich nun, z.B. als Folge eines Ereignisses, erinnert. Aber auch solche Äußerungen, in denen der Sprecher ihm durchgehend präsente Anforderungen verbalisiert, sind mit *remember_requirement* zu annotieren.[56]

   Das Konzept *remember_requirement* adressiert also Verbalisierungen bestimmter Formen von explizitem Wissen (siehe Abschnitt 7.1.1). Es bezieht sich somit auf einen Bereich, der eigentlich schon durch *explain_knowledge* abgedeckt ist. Da aber im Laufe der Entwicklung der BKM die Kenntnis von Anforderungen als zentral für den Prozess der PP eingestuft wurde, kam es zu der Entscheidung, entsprechende Phänomene nicht mit dem Konzept *explain_knowledge*, sondern mit einem neu zu schaffenden zu markieren. In diesem Zusammenhang wurde dann nicht nur ein neues Objekt (*requirement*), sondern sogar ein neues Verb (*remember*) eingeführt.

   Das Konzept *remember_requirement* blieb allerdings lange Zeit das einzige Element der Klasse *requirement*. Erst bei der Beobachtung einer Äußerung, in der ein Vorschlag bez. einer allem Anschein nach fehlenden Voraussetzung gemacht wurde (*propose_requirement*), wurde ersichtlich, dass eine Abspaltung der Anforderungsaspekte aus der Klasse *knowledge* auch in anderer Hinsicht zweckmäßig ist.

2. **Entdecken von fehlenden Anforderungen:** Mit *propose_requirement* werden Äußerungen annotiert, in denen Vorschläge zu offensichtlich oder anscheinend fehlenden bzw. unbekannten Anforderungen gemacht werden. Wie oben beschrieben kann es sich hierbei auch um Vorschläge bez. temporär anzunehmender Anforderungen bzw. Bedingungen handeln (siehe Beispiel 1 in Tabelle 7.8).[57]

[56] Im zweiten Fall sollte man also eher von einem *remind of_requirement* sprechen. Da die beiden Fälle aber oft kaum von einander zu unterscheiden sind, wird im Rahmen der BS/BKM nur *remember_requirement* verwendet.

[57] Im Rahmen der durchgeführten Analysen konnten nicht genug verschiedenartige *propose_requirement*-Äußerungen beobachtet werden, um eine Aussage darüber zu treffen, ob auch hier eine Typisierung analog zu der bei *propose_design* vorgestellten (siehe Tabelle 7.5) vorgenommen werden kann.

3. **Zustimmungen und Ablehnungen:** Die Konzepte *agree_requirement* und *challenge_requirement* können sowohl im Kontext von Phänomenen des Typs *remember_requirement* wie auch im Kontext von Phänomenen des Typs *propose_requirement* verwendet werden.

In den in der vorliegenden Arbeit berücksichtigten Sitzungen wurde kaum über Anforderungen gesprochen, so dass bisher nur wenige Aussagen zu dieser Klasse gemacht werden können. Ein Beispiel für einen Dialog, der sich mit Anforderungen beschäftigt, ist in Tabelle 7.9 wiedergegeben.

Abgrenzungen von *requirement*-Konzepten zu anderen Konzepten

Die wichtigsten Abgrenzungen von Elementen der *requirement*-Klasse zu solchen anderer Klassen wurden bereits erläutert. Es handelt sich um

- *propose_requirement*/remember_requirement vs. *propose_design* (Vorgabe vs. Entscheidungsspielraum) sowie
- *remember_requirement* vs. *explain_knowledge* (Spezialisierung).

Eine weitere zu beachtende Abgrenzung ist die zwischen *propose_requirement* und *explain_finding*. Auch hier handelt es sich in gewisser Weise um eine Spezialisierung.

Weitere Anmerkungen zu *requirement*-Konzepten

Der Anforderungsbegriff der BKM ist dem im Software Engineering allgemein verwendeten sehr ähnlich.[58] Er umfasst sowohl funktionale und nicht-funktionale[59] Anforderungen wie auch Benutzeranforderungen (*user requirements*), Systemanforderungen[60] (*system requirements*) und die Spezifikation von Schnittstellen zu (bestehenden) Fremdsystemen.

Trotz dieser Ähnlichkeit gibt es allerdings auch dezidierte Unterschiede. Der Wichtigste betrifft diejenigen Anforderungen, die unter dem Begriff „nicht-funktional“ zusammengefasst sind. Denn während dieser Teilmenge im Software Engineering oft auch Anforderungen an den Entwicklungsprozess zugerechnet werden

[58] Obwohl es keine allgemein verbindliche Definition bez. dessen gibt, was im Software Engineering unter einem „Requirement“ verstanden werden soll, unterscheiden sich die Beschreibungen in unterschiedlichen Publikationen (z.B. [1; 161; 194]) nur in Details. Im *Guide to the Software Engineering Body of Knowledge* [1] wird zum Beispiel gesagt: „Hence, a software requirement is a property which must be exhibited by software developed or adapted to solve a particular problem. [...] By extension, therefore, the requirements on particular software are typically a complex combination of requirements from different people at different levels of an organization and from the environment in which the software will operate“.

[59] Nicht-funktionale Anforderungen werden in einigen Veröffentlichungen (z.B. Pfleeger und Atlee [161]) eindeutiger als Qualitätsanforderungen (*quality requirements*) bezeichnet.

[60] Im Software Engineering sind Systemanforderungen solche, die Benutzeranforderungen dahingehend detaillieren, dass sie von Softwareentwicklern als Startpunkt für das Systemdesign benutzt werden können [194].

[161; 194] wie z.B. die Vorgabe, bestimmte Prozessstandards umzusetzen, versteht die BKM unter nicht-funktionalen Anforderungen nur solche, die sich direkt auf das Produkt beziehen.

Der Grund hierfür ist, dass in den bisher beobachteten Sitzungen keine Äußerungen zu Anforderungen an den Prozess beobachtet werden konnten. Sollte sich dies in Zukunft im Rahmen weiterer Untersuchungen ändern, muss die BKM diesbezüglich angepasst werden. Die einfachste Möglichkeit würde dann darin bestehen, die Konzeptklasse *requirement* sowohl den produktorientierten als auch den prozessorientierten Konzepten zuzuordnen und somit eine Differenzierung zwischen Produkt- und Prozessanforderungen auf spezialisiertere Untersuchungen zu verlagern. Eine andere Lösung wäre, die Klasse *requirement* in die Subklassen *product requirement* und *process requirement* zu zerteilen.

7.5 Prozessorientierte Konzepte

Wie in Abschnitt 7.3 ausgeführt adressieren die prozessorientierten Konzepte Äußerungen, die sich auf die Gestaltung des Arbeitsprozesses des Paares beziehen. Sie sind auf fünf Unterklassen verteilt und decken drei Bereiche ab:

1. Äußerungen, in denen unmittelbar umzusetzende, in Umsetzung befindliche oder gerade umgesetzte kurzfristig[61] ausgerichtete Handlungsoptionen bzw. Handlungen thematisiert werden, also solche, die sich in gewisser Weise mit dem taktischen[62] Verhalten des Paares auseinander setzen, werden durch Elemente der Klassen *step* und *completion* konzeptionalisiert.

2. Äußerungen bez. des (längerfristigen) strategischen Vorgehens werden hingegen durch Elemente der Klassen *strategy* und *state* behandelt, und

3. Äußerungen, die sich mit dem Zurückstellen bzw. Verschieben von einzelnen Arbeitsschritten beschäftigen, werden durch die Konzepte der Klasse *todo* referenziert.

[61] Bei den Begrifflichkeiten kurz-, mittel- oder langfristig handelt es sich nicht um präzise definierte Intervalle. Vor allem die Unterscheidung zwischen mittel- und langfristig ist im Rahmen der hier betrachteten Prozessdaten nicht ohne weiteres möglich. Sie ist aber in der Regel auch nicht notwendig (im Zusammenhang mit der Klasse *strategy* wird hierauf eingegangen). Einzig der Begriff kurzfristig soll hier etwas konkreter gefasst werden: Damit etwas als kurzfristig durchführbar gelten soll, muss es möglich sein, damit mehr oder weniger unmittelbar beginnen zu können. Außerdem muss es sich aller Voraussicht nach in überschaubarer Zeit (in der Regel wenige Minuten), dass heißt auch in wenigen Arbeitsschritten, bewältigen lassen. Details hierzu folgen in den nächsten Abschnitten.

[62] Inwieweit in diesem Zusammenhang wirklich von taktischem und nicht besser von „ad hoc“-Verhalten gesprochen werden sollte, wird erst dann erläutert, wenn der Strategiebegriff der BKM geklärt ist (siehe S. 216 ff).

HHI-Kode	Äußerung	Kontext/Kommentar
P1.propose_requirement	„Im Idealfall könnten sie einfach noch einen Timestamp von (~jetzt) mitschicken. (...)“	Um das aktuelle Implementierungsproblem zu lösen, schlägt der Driver *P1* eine Voraussetzung vor, die durch den Aufrufenden erfüllt werden sollte.
P1.challenge_requirement	„Aber die können ja nicht jedes mal 'nen Timestamp mitschicken, damit wir gucken, ähh, geht eure Uhr jetzt anders. Die könnten uns jede Minute oder jede zehn (!!...!!)[a]“	Nach drei Sekunden Pause (s.o.), in denen nichts weiter passiert, ändert *P1* seinen eigenen Vorschlag. Hierbei bricht er mitten im Satz ab, da *P2* beginnt, einen Gegenvorschlag zu unterbreiten.
P2.challenge_requirement	„Die könnten ein (~Keepalive) (~~).“	*P2* macht einen Gegenvorschlag. Dieser ist in der Aufzeichnung akustisch nur mit Einschränkungen zu verstehen. Der Partner scheint diesbezüglich aber keine Probleme zu haben. Er reagiert unmittelbar, das heisst ohne zeitliche Verzögerung (siehe nächster Schritt).
P1.agree_requirement	„Das wär' 'ne Idee.“	*P1* stimmt dem Vorschlag von *P2* zu.

[a] Nicht jeder Einwurf, der von einem Partner gemacht wird, führt dazu, dass ein Sprecher seine Äußerung (vorerst) abbricht. Solche Einwürfe werden im Weiteren als „nicht unterbrechend wirksam“ bezeichnet (Beispiele sind in Tabelle 7.47 zu finden).

Tab. 7.9: Beispiel für eine Episode, in der eine fehlende Anforderung bzw. Bedingung diskutiert wird. Ausgangspunkt ist die Erkenntnis des Paares, dass das gerade in Entwicklung befindliche PHP-Skript dem Aufrufenden, einem externen Dienstleister, nur dann die gewünschten Werte liefern kann, wenn die Uhrzeit auf dem System des Aufrufenden mit der auf dem System, auf dem das Skript zum Einsatz kommen wird, synchronisiert ist oder wird. Die Episode stammt aus der Sitzung PR1.1 (16:25:18–16:25:34).

7.5.1 *step*-Konzepte

Fokus der *step*-Konzepte

Schon in den ersten Analysesitzungen konnte beobachtet werden, dass sich die Programmierer in vielen Äußerungen mit der Auswahl bzw. Festlegung des unmittelbar als nächstes durchzuführenden Arbeitsschritts beschäftigen. Dies führte dazu, dass die Klasse *step* eingeführt wurde. Ihre Elemente adressieren Äußerungen, in denen sich die Sprecher mit dem, was als nächstes passieren soll bzw. zu tun ist, auf einem taktischen Niveau auseinander setzen. Konkrete Gestaltungsvorschläge bez. des zu entwickelnden Programms sind hier nicht mit eingeschlos-

sen. Diese werden wie bereits beschrieben über die Elemente der Klassen *design-* bzw. *requirement* konzeptualisiert.[63]

Darüber hinaus grenzen sich die Konzepte der Klasse *step* auch deutlich von denen der Klassen *knowledge* und *finding* ab: *step*-Konzepte adressieren Handlungsvorschläge an und für sich, zugehörige Erklärungen bzw. Begründungen müssen mit Elementen der Klassen *knowledge* oder *finding* annotiert werden.

Charakteristisch für Äußerungen der Klasse *step* ist, dass der Sprechende den Arbeitsschritt, ungeachtet davon, ob dies in der Realität wirklich so ist bzw. sein wird, als eine unteilbare Handlungseinheit darstellt. Hierbei propagiert der Entwickler ein konkretes Mittel für einen bestimmten Zweck. Es muss allerdings keineswegs so sein, dass er den Zweck erwähnt.[64]

Im Weiteren werden in der Regel nur noch solche – auf einem taktischen Niveau in Planung befindlichen – Handlungseinheiten, die als unteilbar dargestellt werden, als Arbeitsschritt bezeichnet.[65] Hierbei ist noch einmal hervorzuheben: So wie alle anderen Objekte, die im Zusammenhang mit den HHI-Konzepten der BKM betrachtet werden, referenzieren auch Arbeitsschritte Elemente der Diskurswelt und nicht etwa solche der Tätigkeitswelt (siehe S. 133). Der Begriff Arbeitsschritt bezeichnet also eine vom Sprecher formulierte Einheit und nicht etwa eine „natürliche“, von Darstellungen durch Personen sowie Kontexten unbeeinflusste. Insbesondere handelt es sich bei einem Arbeitsschritt um eine Maßnahme, mit der ein kurzfristig erreichbares Zwischenziel erreicht werden soll, die dabei aber nicht explizit in den Kontext anderer Maßnahmen gestellt wird. Für diese Maßnahme gilt darüber hinaus: Es muss keineswegs so sein, dass sie genau einem nicht weiter zerlegbaren Element der Tätigkeitswelt, einer einzigen so genannten *atomaren Tätigkeit* (siehe Exkurs 8) entspricht. Vielmehr ist es auch möglich, dass mehrere atomare Tätigkeiten notwendig sind, um die Maßnahme umzusetzen. Demzufolge kann von *atomaren* bzw. *nicht-atomaren Arbeitsschritten* gesprochen werden: Ein Arbeitsschritt heißt atomar, wenn er sich auf genau eine atomare Tätigkeit abbilden lässt, er heißt nicht-atomar, wenn mehrere z.B. atomare Tätigkeiten notwendig sind, um ihn umzusetzen. Die Sprecher unterscheiden hier aber in der Regel nicht explizit.

Zusammengefasst gilt für Arbeitsschritte:

- Sie adressieren Tätigkeiten, die als nächstes durchgeführt werden sollen.

- Sie werden in der Regel als unteilbare Einheiten formuliert.

[63] Oft gehen Gestaltungsvorschläge mit Arbeitsschrittvorschlägen einher. Trotzdem werden solche Äußerungen im Rahmen der BKM in der Regel nur mit *propose_design* annotiert. Auf S. 201 wird dies näher erläutert.

[64] Sollte er den Zweck erwähnen, so wäre dieser in der Regel einzeln mit einem Konzept wie z.B. *explain_knowledge* zu kodieren (siehe S. 192).

[65] Manchmal ist es nötig, taktische Arbeitsschritte zu referenzieren, die sich bereits in Umsetzung befinden bzw. umgesetzt sind. Auch hier wird in Ermangelung eines weiteren geeigneten Begriffs von Arbeitsschritten gesprochen. Dies geschieht selbst dann, wenn er im Vorfeld nicht als ein solcher verbalisiert wurde (siehe z.B. Abschnitt 7.5.2).

Exkurs 8: *Atomare Tätigkeiten*
Es ergibt sich keineswegs kanonisch, wann eine Tätigkeit einer Person (im Rahmen des Paarprogrammierungsprozesses) als atomar, also als nicht in weitere Teilschritte zerlegbar angesehen werden soll. Denn auch auf den ersten Blick atomar scheinende Tätigkeiten wie beispielsweise das auf S. 192 als atomar bezeichnete Schließen eines Programmfensters lassen sich in der Regel noch weiter zerlegen, z.B. in das Bewegen der Maus zur Schaltfläche und das Betätigen der Schaltfläche. Es scheint also erst einmal kaum sinnvoll zu sein, unter atomaren Tätigkeiten einfach solche zu verstehen, die sich in keiner Weise in weitere Schritte zerlegen lassen.
Im Rahmen der BKM spielt dieses Problem zwar nur eine untergeordnete Rolle, zur Verbesserung des Verständnisses der Natur von Arbeitsschritten ist es aber trotzdem hilfreich, eine Vorstellung davon zu bekommen, wie der Begriff „atomare Tätigkeit“ hilfreich zum Einsatz gebracht werden kann: Eine konkret aufgetretene oder in einer bestimmten Situation potentiell mögliche Tätigkeit soll als atomar angesehen werden, wenn sie sich nur noch in solche Schritte zerlegen lässt, die von den beteiligten Akteuren im aktuellen Kontext nicht mehr explizit als eigenständig wahrgenommen bzw. artikuliert werden würden.
Bei dieser Betrachtungsweise fallen zwei Dinge auf. Zum einen verlässt sie die Tätigkeitswelt, indem sie auf die Diskurswelt bzw. die Wahrnehmung einzelner Personen zurückgreift, zum anderen klassifiziert sie eine Tätigkeit nicht isoliert für sich allein, sondern unter Einbeziehung des Kontextes, in dem sie auftritt. Ersteres hängt damit zusammen, dass beim Sprechen (bzw. Nachdenken) über Tätigkeiten automatisch die Tätigkeitswelt verlassen wird. Es werden nicht-materielle Dinge, eben Tätigkeiten, als Einheiten betrachtet. Der zweite Punkt zollt einer mit dem ersten Punkt zusammenhängenden Eigenschaft Rechenschaft, nämlich der, dass Menschen bestimmte Tätigkeiten in unterschiedlichen Situationen unterschiedlich wahrnehmen. So wird beispielsweise das Ausführen eines *SVN-Updates* auf eine Datei im Allgemeinen als atomar erlebt werden. Wenn es allerdings darum geht, dass einer Personen beigebracht werden soll, wie man eine solches *Update* durchführt, wird der Schritt sinnvollerweise nicht mehr als atomar dargestellt werden.
Im Rahmen der vorliegenden Arbeit wird also davon ausgegangen, dass eine sinnvolle Definition des Begriffs „atomare Tätigkeit“ den Kontext konkreter Situationen einbeziehen muss. Dieser Kontext umfasst dann auch die (subjektive) Wahrnehmung/Deutung durch einen Beobachter. Dies kann im Rahmen der hier geschilderten Untersuchungen auch der Forscher sein, der die jeweilige Situation und deren (wahrscheinliche) Wahrnehmung durch die Entwickler einbezieht. Dies kann allerdings nur dann zuverlässig erfolgen, wenn der Forscher mit Programmierprozessen vertraut ist.

– Sie müssen aber nicht unteilbar sein. Vielmehr führt ihre Umsetzung in der Regel zu einer Reihe von (atomaren) Tätigkeiten.

– Sie werden in der Regel nicht verbal in Bezug zu anderen Arbeitsschritten gesetzt.

Beispiele für in den meisten Kontexten als nicht-atomar wahrgenommene Arbeitsschritte sind solche, die auf

– das Durchführen von Testläufen oder Durchsichten (z.B. „Wir könnten so'n ersten Test mal machen." (PR1.1)),

– das Bearbeiten eines konkreten weiteren Teiles, z.B. einer Methode (siehe z.B. S. 201),

– das Reorganisieren der Oberfläche der Entwicklungsumgebung, z.B. das Verschieben und/oder Schließen von mehreren *Views* in der IDE Eclipse,

– das Durchsehen oder Überprüfen von existierendem Kode (z.B. „Ich würd', (..) lass uns mal das `TableModel` angucken, wie das arbeitet, wie das die Werte abholt." (PR2.1)) oder

– das Diskutieren eines bestimmten, für die Realisierung des Programms wichtigen Aspektes

zielen.

Die Konzepte der Klasse *step* adressieren also sowohl komplexere Abläufe als auch in den meisten Kontexten als atomar wahrgenommene Schritte wie beispielsweise

– das Ausführen eines bestimmten Kommandos, z.B. das Speichern einer Datei (z.B. „Mach mal ‚Speichern'." (PR2.1)),

– das Auswählen eines bestimmten Menüpunkts oder Anzeigen eines bestimmten Kontextmenüs (z.B. „Geh⌣mal, geh' mal drüber." (PR2.1)),

– das Setzen eines „Breakpoints" (z.B. „Wir können 'n Breakpoint setzen." (PR2.1)),

– das Durchführen eines *Updates*[66] (z.B. „Erst mal vielleicht 'n *Update* machen?" (PR2.1)),

– das Ändern von Bildschirmeinstellungen (z.B. „Wir machen erst mal die Schrift ein bisschen kleiner." (PR1.1))[67],

[66] Der Begriff *Update* – wie auch der Begriff *Commit* (siehe Fußnote 36 auf S. 101) – bezieht sich hier auf die entsprechende Funktionalität einer Versionsverwaltung wie z.B. dem Subversion. Bei einem *Update* wird die lokale Kopie einer Datei aktualisiert.

[67] Ob das Ändern der Schriftgröße als atomare Tätigkeit bewertet werden kann, hängt unter anderem von dem Programm ab, in welchem die Schriftgröße geändert werden soll.

– das Schließen einer bestimmten Ansicht bzw. eines bestimmten Fensters (z.B. „Das können wir zumachen, oder?“ (PR2.1)) oder

– das Schreiben eines kurzen „ToDo“-Kommentars (z.B. „(~Da schreibe ich mal) 'n <**Entwicklername**> dran.“ (PR2.1)).

Weitere konkrete Beispiele für *step*-Äußerungen sind in Tabelle 7.10 zu finden.

Achtung: Äußerungen, die sich mit Arbeitsschritten beschäftigen, die zumindest nach Meinung einer der Entwickler erst zu einem späteren Zeitpunkt durchgeführt werden sollen, werden nicht mit Konzepten der Klasse *step*, sondern mit Konzepten der Klasse *todo* annotiert (siehe S. 209). Außerdem gibt es für die Konzeptualisierung komplexerer Arbeitsplanungen noch die Elemente der Klasse *strategy*. Sie werden ab S. 216 erläutert. Hier wird auch noch einmal auf die Zusammenhänge zwischen Arbeitsschritten und Tätigkeiten eingegangen (siehe z.B. Abbildung 7.8).

Identifizierte *step*-Konzepte und ihre speziellen Eigenschaften

Für die Klasse *step* wurden die Konzepte *propose_step*, *agree_step*, *decide_step*, *disagree_step*, *challenge_step*, *amend_step* und *ask_step* beobachtet (siehe Abbildung 7.3). Folgende spezielle Eigenschaften dieser Konzepte sollten berücksichtigt werden:

1. **Begründete Vorschläge:** Ähnlich wie Phänomene vom Typ *propose_design* treten auch Phänomene vom Typ *propose_step* oft im Zusammenhang mit einer Begründung auf. Diese sollte separat als *explain_knowledge* (siehe auch S. 331) oder ggf. *explain_finding* kodiert werden.

   Potentiell gilt dies auch für die Phänomene vom Typ *challenge_step*, *amend_step* und *disagree_step*. Für diese wurde der angesprochene Zusammenhang allerdings nur selten beobachtet.[68]

   Als Beispiel sei auf die *amend_step*-Äußerung aus Tabelle 7.10 verwiesen, welche vom Sprecher mit folgenden Worten ergänzt wurde:

   > „Ich dachte gerade, ich hätte eh nur das <*Working Set*> ausgecheckt, aber (!...!).“

   Mit dieser Ergänzung begründet der Driver seine Vorschlagsänderung. Er verbalisiert die gerade gewonnene Erkenntnis, dass seine ursprüngliche Annahme, der *Workspace* sei gleich dem *Working Set*, falsch ist (*explain_finding*).[69]

[68] Gleiches gilt für Phänomene vom Typ *agree_step* und *decide_step*. Auch sie können, zumindest theoretisch, zusammen mit Begründungen auftreten.

[69] Seine Erkenntnis fußt auf der Beobachtung einer Rückmeldung der direkt nach dem ursprünglichen Vorschlag gestarteten IDE-Suchfunktionalität. Dieser war zu entnehmen, dass 2000 Dateien in die Suche einbezogen werden.

#	Session und HHI-Kode	Äußerung	Kontext/Kommentar
1	PR1.1: *P1.explain_finding*[a] + *P1.amend_step*	„Ähhh, (.) ja, ist natürlich auch totaler Unsinn – wir können in unserem *Working Set* suchen.“	Das Paar ist dabei, nach Stellen im Kode zu suchen, an denen eine bestimmte Funktion aufgerufen wird. *P1* hatte als Suchbereich den gesamten *Workspace* vorgeschlagen. Nachdem er eine entsprechende IDE-Suchoperation gestartet hat und sieht, dass die Abarbeitung viel Zeit benötigen wird, verbessert er seinen Vorschlag dahingehend, dass nur noch im aktuellen *Working Set* gesucht werden soll. Nachfolgend startet er die jetzt eingeschränkte Suche neu.[b]
2	PR2.1: *P1.propose_step*	„Dann (.), (~is' es so), zeig' ich dir erst mal, was ich gemacht habe.“	Das Paar muss im Rahmen der Sitzung an Kodeabschnitten arbeiten, die im Vorfeld von *P1* editiert bzw. neu angelegt wurden. *P1* schlägt vor, seinen Partner bez. dieser Abschnitte auf einen aktuellen Stand zu bringen.[c]
3	PR2.1: *P1.propose_step*	„(O.K. *Refactor*).“	Nachdem *P1* vorgeschlagen hat, eine Klasse innerhalb der vorhandenen Paketstruktur zu verschieben, scrollt er durch den *Package Explorer* der IDE Eclipse[d]. Nach kurzer Zeit findet er die zu verschiebende Klasse in der Hierarchie und ruft auf dem entdeckten Eintrag ein Kontextmenü auf. Hierbei äußert er sich wie angegeben und gibt damit zum Ausdruck, dass er als nächstes die *Refactoring*-Funktionalität der IDE verwenden möchte. Eine knappe Sekunde später hat er diese aus dem Kontextmenü ausgewählt.
4	PR2.1: *P1.propose_step*	„Eigenschaften.“	Das Paar ist gerade dabei, den von ihnen bearbeiteten Teil der Applikation zu testen. Als der Driver einen Moment zögert, schlägt der Observer vor, als nächstes eine bestimmte Funktionalität – das Darstellen von „Eigenschaften“ – aufzurufen.
5	PR2.1: *P1.propose_step*	„Mach' mal aus den TODOs oben noch NOWs.“	*P2* ist gerade dabei, eine noch leere Methode mit einem „TODO_NOW“ zu kommentieren, als der Observer vorschlägt, auch die anderen, bisher noch nicht mit Funktionalität gefüllten und deshalb mit „TODO“ kommentierten Methoden der Klasse, mit „TODO_NOW“ zu kennzeichnen.

[a] Näheres zu der Kodierung mit *explain_finding* wird auf S. 281 erläutert.
[b] Auf dieses Beispiel wird noch einmal in Tabelle 8.3 eingegangen.
[c] Es handelt sich um einen Arbeitsschritt mit strategischem Charakter (siehe S. 230).
[d] Im *Package Explorer* wird die Hierarchie der Java-Elemente der Projekte angezeigt (http://help.eclipse.org/indigo/topic/org.eclipse.jdt.doc.user/reference/views/ref-view-package-explorer.htm (Abruf: 08.02.2012))

Tab. 7.10: Beispiele für Äußerungen, die mit *step*-Konzepten annotiert wurden. Es sind sowohl Äußerungen aufgeführt, die sich auf atomare bzw. kleinteilige Arbeitsschritte beziehen (siehe Beispiel 4), wie auch solche, die Schritte adressieren, die letztendlich nicht atomar sein werden (siehe Beispiel 2). Beispiel 3 (und in gewisser Weise auch Beispiel 1) zeigen, dass das Konzept *propose_step* (bzw. *amend_step*) auch dann angewendet wird, wenn der Sprechende allem Anschein nach gar keinen Kommentar bzw. keine Zustimmung erwartet, die Äußerung vielmehr nur Informationscharakter besitzt.

Bei der Darstellung der Klasse *knowledge* wird auf die Differenzierung zwischen *explain_knowledge* und *propose_step* noch einmal eingegangen (siehe S. 331).

2. **Ziele von Vorschlägen:** Auch in Hinsicht auf die Ziele, die ein Sprecher mit *propose_step*-Äußerungen verfolgt bzw. zu verfolgen scheint, zeigen sich Analogien zu den mit *propose_design* annotierten. Denn bei weitem nicht jeder dieser Vorschläge scheint mit der Intention gemacht zu werden, diesen zu diskutieren oder vom Partner bewerten zu lassen. Oft, speziell wenn der Vorschlag beim Ausüben einer HCI-Tätigkeit geäußert wird, dient er allem Anschein nach nur der Information des Gegenübers (siehe Beispiel 3 in Tabelle 7.10).

   So waren beispielsweise in Sitzung PR2.1 weniger als 12% der mit *propose_step* annotierten Äußerungen in der Art als Frage formuliert, dass als Intention des Sprechenden eindeutig der Wunsch nach einer Absprache mit dem Partner zu erkennen war, wie es z.B. bei folgenden Äußerungen der Fall ist:

   – „Magst Du nicht einfach bei Dir kurz nachschauen?“
   – „Das können wir zumachen, oder?“

   Beim überwiegenden Teil der *propose_step*-Äußerungen in Sitzung PR2.1 handelt es sich vielmehr eher um Handlungsaufforderungen oder, falls der Sprecher in der Rolle des Drivers war, schlicht um Informationen darüber, was er als nächstes vorhat.

   Wie schon bei den Erläuterungen der Klasse *design* angedeutet wurde, benutzt die BKM das Verb „vorschlagen“ (*propose*) also auch im Zusammenhang mit der Klasse *step* vielfältig: Beim Einholen von weiteren Meinungen, beim Übermitteln von Informationen bzw. Geben von Anweisungen sowie beim Suchen nach Orientierung. Hierbei gelten dieselben Charakterisierungen, wie in Tabelle 7.5 für die Klasse *design* dargestellt. Ein Beispiel für eine *step*-Äußerung vom Typ 3 (SO) ist in Tabelle 7.11 zu finden.

3. **Weitere Intentionen von Vorschlägen:** Mit Vorschlagsäußerungen können zusätzliche Intentionen verfolgt werden. So kann mit einem Vorschlag einer Erkenntnis des Partners widersprochen werden. Im Rahmen der Erläuterungen zur Klasse *finding* wird hierauf genauer eingegangen (siehe Tabelle 7.33).

4. **Vorschläge in Frageform:** Bei in Frageform formulierten Vorschlägen zum nächsten Arbeitsschritt wird nur der primäre illokutionäre Akt kodiert (siehe Argumentation auf S. 170).

HHI-Kode	Äußerung	Kontext/Kommentar
P2.propose_step	„Die Frage ist, ob ich mich da jetzt rantrauen will, ob wir uns da jetzt rantrauen wollen?“	Der Observer stellt zur Diskussion, ob die gerade erörterten Änderungen als nächstes durchgeführt werden sollten. Es handelt sich um einen Vorschlag zum nächsten Arbeitsschritt[a], bei dem der Sprecher allerdings keine eigene Meinung, sondern vielmehr Unsicherheit artikuliert. Diese Unsicherheit formuliert er in Frageform, was dafür spricht, dass er sich eine Beurteilung durch den Partner wünscht.
P1.agree_step	„Ich würd’ mich da schon gerne rantrauen.“	Der Driver interpretiert die Frage des Partners als Vorschlag und stimmt diesem zu.[b] Obwohl er nicht noch einmal expliziert, den Schritt als nächstes durchführen zu wollen, ist davon auszugehen, dass er genau dies beabsichtigt. Begründen lässt sich diese Annahme damit, dass das Paar zu diesem Zeitpunkt keinen anderen Schritt ins Auge gefasst hat und der Driver auch nachfolgend keinen solchen vorschlägt.
P2.agree_step	„O.K.“	Nun stimmt auch der Observer zu, d.h. er liefert eine Bewertung des eigenen Vorschlags nach.[c]

[a] Genau genommen handelt es sich um einen Grenzfall zwischen Taktik und Strategie (siehe auch Abschnitt 7.5.4). Für eine Interpretation als taktische Äußerung spricht die Tatsache, dass, obwohl es sich um eine komplexere Restrukturierung des Kodes handelt, im Grunde nur ein konkreter Arbeitsschritt adressiert wird, für eine Interpretation als Strategie (vom Typ DPR (siehe S. 217)) die Vermutung, dass es darum gehen könnte, das weitere Vorgehen durch ein vorheriges Refaktorisieren zu erleichtern. Da das Zweite im Rahmen der Analysen nicht mit Sicherheit nachgewiesen werden konnte, wurde die Äußerung der Klasse *step* zugeschrieben.

[b] Hier kommt nicht das Konzept *decide_step* zum Einsatz, da der Driver keine Auswahl zwischen unterschiedlichen Arbeitsschritten trifft, sondern nur entscheidet, dass er den angesprochenen Schritt durchführen will.

[c] An dieser Stelle könnte eingewendet werden, dass es sich nicht um eine Bewertung des eigenen Vorschlags, sondern um eine Zustimmung zur „Entscheidung“ des Drivers handelt. Die BKM differenziert an dieser Stelle aber nicht. Zustimmungen zu Zustimmungen werden wie Zustimmungen zur ursprünglichen Äußerung behandelt. Spezialisierte Untersuchungen müssen in diesem Zusammenhang evtl. Differenzierungsmöglichkeiten entwickeln.

Tab. 7.11: Beispiel für eine *propose_step*-Äußerung vom Typ 3 (SO): „Orientierung suchen“ (siehe Tabelle 7.5). Der Äußerung war ein Designvorschlag des Drivers, genau genommen ein Vorschlag zur Refaktorisierung (siehe Beispiel 1 in Tabelle 7.5) und nachfolgend ein kurzer diesbezüglicher Wissenstransfer vorausgegangen. Zum Zeitpunkt der Äußerung des *propose_step*s bestand dann allem Anschein nach kein Zweifel mehr daran, dass der Designvorschlag prinzipiell eine gute Idee ist. Nicht geklärt war hingegen, was als nächstes getan werden soll. Die Episode stammt aus der Sitzung PR2.1 (12:58:08–12:58:16).

5. **Reservieren von Zeit:** Das Konzept *propose_step* wird auch in solchen Fällen verwendet, in denen der Arbeitsschritt selbst nicht explizit verbalisiert, sondern nur Zeit für dessen Durchführung gefordert wird. Ein Beispiel hierfür ist die Äußerung

   „Warte mal kurz."

   aus Session PR2.1. Betrachtet man den Kontext dieser Äußerung, so ist zu erkennen, dass der Driver Zeit fordert, um sich einen bestimmten Teil im Programmkode näher ansehen zu können.

   Das Beispiel macht auch deutlich, dass sich die vorgeschlagenen Arbeitsschritte nicht immer auf die Arbeit im Paar beziehen müssen.

6. **Unpräzise Vorschläge:** Aus dem letzten Punkt folgert, dass Äußerungen vom Typ *propose_step* und somit potentiell auch solche vom Typ *amend_step* oder *challenge_step* nicht derart formuliert sein müssen, dass der Arbeitsschritt vollständig spezifiziert ist. Dies gilt selbst für Vorschläge, die den Charakter einer Aufforderung haben. Ein Beispiel hierfür ist in Tabelle 7.12 zu finden. Die Handlungsaufforderung ist in diesem Fall so unspezifisch formuliert, dass sie dem Driver eine Vielzahl von Möglichkeiten lässt.

7. **Zustimmung vs. Auswahl:** Für die Differenzierung zwischen *decide_step* und *agree_step* gilt dasselbe wie für die zwischen *decide_design* und *agree_design* (siehe S. 174). Das Konzept *decide_step* wurde allerdings bisher nur ein einziges Mal beobachtet.

8. **Eigene Vorschläge ergänzen oder widerrufen:** In Abbildung 7.3 wurde erläutert, dass sich Äußerungen vom Typ *amend_step* auf in der Regel kurz zuvor gemachte Vorschläge beziehen. Der ursprüngliche Vorschlag wird dabei nicht in Frage gestellt oder abgelehnt, sondern nur detailliert. Die hier beschriebenen Untersuchungen haben darüber hinaus gezeigt, dass es, insbesondere wenn der Sprechende in der Rolle des Observers ist, vorkommen kann, dass Vorschlag sowie Ergänzung von derselben Person hervorgebracht werden. Dies kann sogar dann passieren, wenn der Partner zwischenzeitlich keine Fragen oder Gegenvorschläge äußert. Dem Sprechenden scheint es in diesen Fällen darum zu gehen sicherzustellen, dass der Partner den Vorschlag wirklich in seinem Sinne ausführt. Auch solche Phänomene sind mit *amend_step* zu annotieren.

   Ein Beispiel ist in Tabelle 7.12 erläutert. Es verdeutlicht darüber hinaus, dass eine Entscheidung darüber, ob eine Äußerung wirklich mit *amend_step* kodiert werden kann, zumindest im Rahmen des offenen Kodierens nicht in jedem Fall eindeutig getroffen werden kann.

   Darüber hinaus kann es vorkommen, dass ein Entwickler einen von ihm selbst hervorgebrachten Vorschlag wieder zurück nimmt bzw. letztendlich

doch ablehnt. Derartige Phänomene sind mit *disagree_step* zu annotieren. In Tabelle 7.13 ist ein diesbezügliches Beispiel dargestellt.

Achtung: Wie schon im Rahmen der Erläuterungen zum Konzept *design* ausgeführt (siehe S. 175), ist es plausibel, dass „Selbstkorrekturen“ auch im Zusammenhang mit allen anderen Objektklassen auftreten können (siehe auch Abschnitt 9.3.9). Hier sollte analog verfahren werden, auch wenn solche Phänomene im Rahmen der in der vorliegenden Arbeit beschriebenen Untersuchungen nicht für jeden Einzelfall beobachtet werden konnten.

9. **Zustimmungen vs. Aufmerksamkeit anzeigen:** Wie schon im Zusammenhang mit den Erläuterungen zu *agree_design* (S. 175) ausgeführt unterscheidet die BKM auch bei Äußerungen, die mit *agree_step* konzeptionalisiert werden müssen, nicht zwischen bewussten Zustimmungen und dem, was eher als „Back-channel“ zu bezeichnen wäre. Jede potentielle Zustimmung, z.B. „Mhm“, „Ja“ oder auch „Gut, dann probieren wir die GUI aus“ (PR2.1) wird im Rahmen der BKM mit *agree_step* annotiert. Differenzierungen werden nachfolgenden Untersuchungen überlassen.

10. **Kontextbezug von „vorschlagsfreien“ Fragen:** Fragen, die darauf abzielen herauszubekommen, welcher Arbeitsschritt als nächster ausgeführt werden sollte, deren Formulierung aber selbst keinen konkreten Vorschlag enthält, werden mit *ask_step* annotiert.

    Solche Fragen können einerseits sehr allgemein formuliert sein, sich andererseits aber auch auf einen bestimmten Kontext, z.B. die aktuelle Situation, beziehen. Dementsprechend wird auch eine Frage wie „Wo wollten wir rein?“ (ST1.1), bei der es um die als nächstes zu öffnende bzw. zu analysierende Datei geht, als *ask_step* annotiert.

    Das Beispiel verdeutlicht außerdem, dass es keineswegs so sein muss, dass Vorschläge, die diese Frage beantworten, nicht schon im Vorfeld diskutiert und evtl. sogar entschieden wurden. Die BKM verzichtet also im Zusammenhang mit *ask_step*-Konzepten auf eine Differenzierung zwischen dem Wunsch nach Rekapitulation und dem Stellen einer neuen Anfrage.[70]

[70] Dementsprechend wurde im Rahmen der BKM auch auf das Konzept *remember_step* verzichtet bzw. festgelegt, dass das Verb *remember* nur in Bezug auf Informationen bzw. Wissen verwendet werden soll, welches bereits vor bzw. zu Beginn der Sitzung zur Verfügung stand. Als diesbezüglich eine Rolle spielendes Wissen wurde im Laufe der Analysen dasjenige über Anforderungen (*requirement*) und dasjenige über Informationsquellen (*source of information*) identifiziert. Anderweitige Äußerungen, die ein „in Erinnerung rufen“ zum Ziel haben, werden von der BS/BKM nicht explizit als solche adressiert. In weiterführenden Untersuchungen muss aber darüber nachgedacht werden, ob und in welcher Weise diese Lücke geschlossen werden sollte.

HHI-Kode	Äußerung	Kontext/Kommentar
P2.propose_step	„Geh‿mal, geh' mal drüber."	Der Observer fordert den Driver auf, mit der Maus über einen der im Editor angezeigten Defekte zu fahren, ohne näher zu spezifizieren, über welchen. Mit der Aktion soll, wie sich aus den nachfolgenden Schritten klar ablesen lässt, erreicht werden, dass die IDE ein so genanntes *Tool Tip* öffnet, in dem eine nähere Beschreibung des Defektes angezeigt wird.
-	-	Der Driver nimmt den Vorschlag des Observers direkt und ohne Kommentar an. Er fährt mit der Maus über den letzten im Editor angezeigten Defekt. Ein *Tool Tip* öffnet sich.
P2.amend_step	„Ne, hier. Über das vielleicht besser."	Nachdem sich Driver und Observer die dargestellte Information eine knappe Sekunde angesehen haben, präzisiert der Observer seinen Vorschlag. Hierbei zeigt er mit dem Finger auf diejenige Stelle im Kode, über die der Driver eigentlich mit der Maus hätte fahren sollen.

Tab. 7.12: Episode aus PR2.1 (12:18:25–12:18:31), in welcher der Observer *P2* einen Vorschlag in dem Moment präzisiert, in welchem er erkennt, dass sein Partner nicht wie von ihm intendiert handelt. Der Kontext der Episode ist der folgende: Im *Problems-View* war von der IDE Eclipse eine Reihe von Defekten (*Errors*) gelistet worden. Der Driver hatte auf den ersten Eintrag in dieser Liste geklickt und war mit dieser Aktion in einen Eclipse-Editor „gesprungen". In diesem waren mehrere fehlerhafte Stellen im Kode markiert. Nach einem kurzen Augenblick des Innehaltens wurde vom Observer der oben aufgeführte Vorschlag unterbreitet. Mit der Episode wird verdeutlicht, dass es nicht immer möglich ist, zu entscheiden, ob es sich bei einem Phänomen wirklich um ein *amend_step* handelt. Im vorliegenden Fall kann nicht ermittelt werden, auf welche Stelle im Kode sich der Observer ursprünglich bezog. Die Phrase „vielleicht besser" in der zweiten Äußerung könnte als Eingeständnis des Observers gewertet werden, ursprünglich eine andere Idee gehabt zu haben. Dann wäre seine Aussage als *challenge_step* zu annotieren. Sie könnte aber auch strategischer Natur sein und nur das Ziel verfolgen, den Partner nicht zu blamieren bzw. vor den Kopf zu stoßen. Im Rahmen der Kodierung von PR2.1 wurde zugunsten der zweiten Variante entschieden, allerdings nicht, ohne mittels eines Memos (siehe S. 54 bzw. S. 76) auf das Problem hinzuweisen.

HHI-Kode	Äußerung	Kontext/Kommentar
P2.propose_step	„Jetzt machen‿wir‿nochmal 'n Refresh.“	Der Observer schlägt vor, die gerade editierte PHP-Seite ein zweites Mal über den Browser aufzurufen, sprich zu aktualisieren.
P2.disagree_step + *P2.explain_finding*	„Ach‿ne, wir hab'n noch kein (~) `Set` gemacht“	Der Observer nimmt seinen Vorschlag fast unmittelbar – es vergeht nur eine knappe Sekunde – wieder zurück und begründet diese Meinungsäußerung. Da ihm allem Anschein nach nicht von Anfang an klar gewesen ist, dass ein erneutes Aufrufen der Seite nicht zu dem von ihm gewünschten, allerdings nicht explizit artikulierten Ergebnis führen würde, wurde die Begründung mit *explain_finding* und nicht etwa mit *explain_knowledge* kodiert.

Tab. 7.13: Episode aus PR1.1 (14:38:56–14:38:59), in welcher einer der Entwickler einen eigenen Vorschlag zum nächsten Arbeitsschritt wieder zurück nimmt. Ursprünglich hatte er gefordert, eine im Vorfeld vorgenommene Änderung an einem PHP-Skript nicht nur durch einen einzelnen, sondern zusätzlich durch einen zweiten Aufruf des Skriptes zu verifizieren. Unmittelbar nach seiner Äußerung erkennt er aber, dass sich die Ausgabe des Skriptes, bei der es sich im Wesentlichen um zu bestimmten Datenbankeinträgen gehörige Zeitangaben handelt, bei einem zweiten Aufruf nicht wie von ihm erwartet ändern würde. Diese Erkenntnis formuliert er explizit. Die Episode ist ein Beispiel dafür, dass ein Erkenntnisgewinn nicht ausschließlich an Reaktionen auf externe Phänomene festgemacht werden kann. Auch eine Meinungsänderung kann, vor allem wenn sie begründet wird, als Indiz für eine Erkenntnis gewertet werden (siehe hierzu auch Abschnitt 7.6.1).

Abgrenzungen von *step*-Konzepten zu anderen Konzepten

In Abbildung 7.5 sind die wichtigsten Kriterien für die Abgrenzung zwischen *step*-Konzepten und solchen der Klassen *knowledge* und *design* dargestellt. Einzelheiten hierzu werden nachfolgend erläutert.

Achtung: Auf Abgrenzungen zu Konzepten der Klassen *todo* und *strategy* wird erst dann eingegangen, wenn diese Klassen im Detail eingeführt sind (siehe S. 212 bzw. S. 228).

- ***_step* vs. *explain_knowledge*/*explain_finding*:** Wie bereits erläutert ist es auch denkbar, Vorschläge zum nächsten Arbeitsschritt als Wissen zu interpretieren. Die BS/BKM geht aber einen anderen Weg. Sie legt fest, dass Vorschläge von besonderem Interesse und deshalb mit *propose_step*

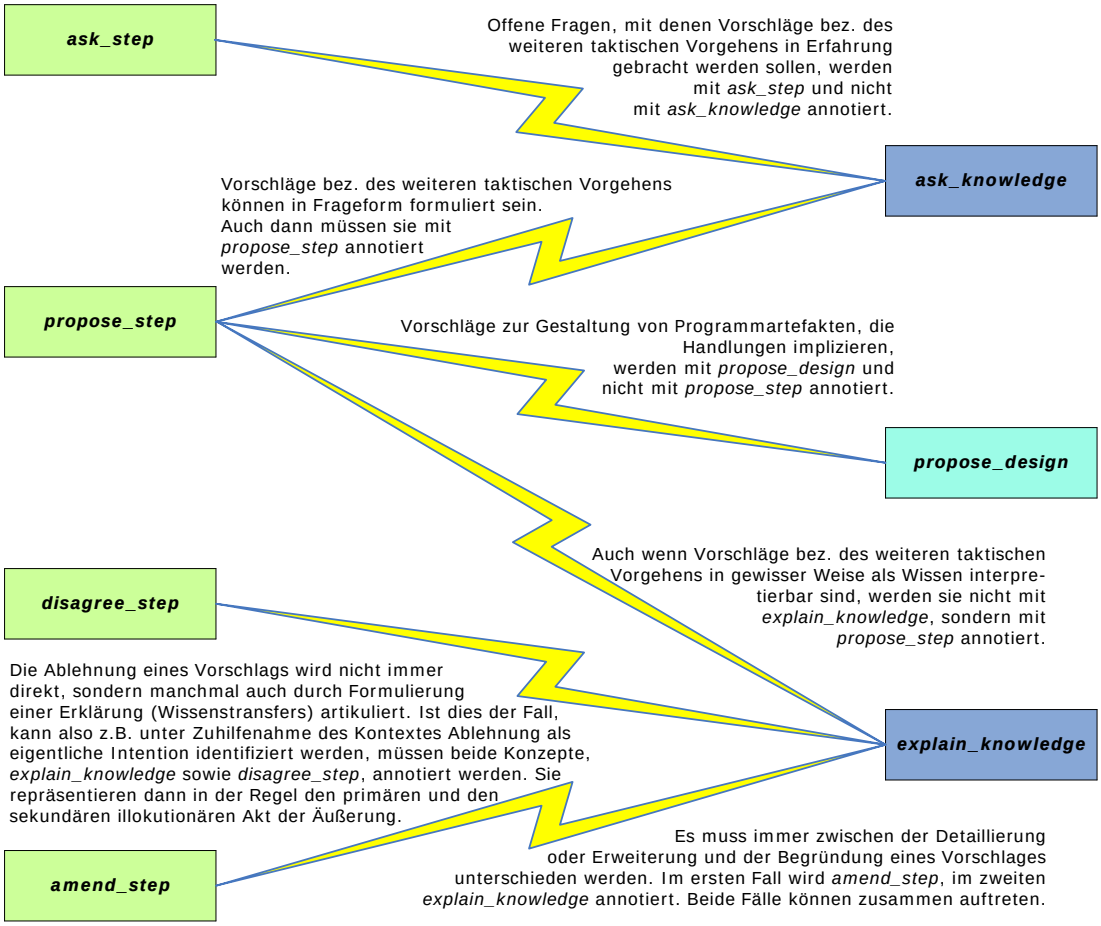


Abb. 7.5: Übersicht über die wichtigsten Kriterien zur Differenzierung zwischen Elementen der Klasse *step* und solchen der Klassen *knowledge* und *design*.

zu annotieren sind. Nur etwaige zugehörige Begründungen sind mit *explain_knowledge* oder *explain_finding* zu konzeptualisieren.

Gleiches gilt für Phänomene vom Typ *agree_step*, *challenge_step*, *amend_step*, *disagree_step* und *decide_step*[71], wobei auf spezielle, diese Abgrenzung betreffende Eigenschaften einzelner *disagree_step*- und *amend_step*-Phänomene im weiteren Verlauf dieses Abschnittes noch einmal eingegangen wird (siehe z.B. S. 205).

– ***propose_step* vs. *propose_design*:** In Abschnitt 7.4.1 wurde erläutert, dass das Konzept *propose_design* genau dann verwendet wird, wenn ein Sprecher einen Vorschlag zur Gestaltung von Programmartefakten unterbreitet. Oft ist es aber nicht alleine anhand der gerade betrachteten Äußerung, sondern nur unter Einbeziehung des näheren oder auch weite-

[71] Inwieweit Phänomene vom Typ *agree_step* und *disagree_step* wirklich als Transfer von Wissen interpretiert werden können bzw. sollten, spielt im Rahmen der Verwendung der BKM keine entscheidende Rolle und soll hier nicht weiter diskutiert werden.

ren Kontextes möglich, eine diesbezügliche Intention zu identifizieren (siehe hierzu z.B. auch Abschnitt 9.3.1 und Abschnitt 9.3.3).

Als Beispiel sei hier auf die Äußerung

> „O.K. und `setFriendship` dann“

verwiesen (PR1.1). Auf den ersten Blick könnte man schließen, dass es sich, da der Sprecher allem Anschein nach für das Anlegen einer Methode `setFriendship` plädiert, um einen Vorschlag bez. der Gestaltung des Kodes handelt. Erst wenn man den weiteren Kontext betrachtet, in diesem Fall den vorliegenden Programmkode, wird klar, dass diese Einschätzung nicht richtig ist. Denn die angesprochene Methode existierte zum Zeitpunkt der Äußerung bereits. Bei dem Vorschlag handelte es sich also keineswegs um ein *propose_design*, sondern um ein *propose_step* (Überprüfen der Methode in Hinsicht auf die Notwendigkeit sie zu modifizieren).[72]

Das Beispiel zeigt, dass es grundsätzlich nicht ausreichend ist, den Kontext einer Äußerung nur in Zweifelsfällen explizit in die Analyse einer Äußerung einzubeziehen. Um eine treffende Kodierung zur erhalten, müssen vielmehr auch die Ziele vermeintlich eindeutiger Äußerungen über den Kontext plausibilisiert werden.

Doch selbst wenn der Kontext ermittelt und verstanden wurde, kann es schwierig werden, darüber zu entscheiden, ob ein bestimmtes Phänomen mit *propose_step* oder *propose_design* zu annotieren ist. Dies liegt daran, dass Designvorschläge oft mit Handlungsvorschlägen einher gehen. Man betrachte beispielsweise folgende Äußerung aus PR1.1:

> „Den `header`[73] setz’ ich mal gleich nach ganz oben. Nach dem Check.“

Hier informiert der Driver den Observer darüber, was er als nächstes zu tun gedenkt. Die Verwendung des Konzeptes *propose_step* liegt also nahe. Andererseits handelt es sich um eine Aussage zur konkreten Umgestaltung des Kodes: Die `header`-Zeile soll an den Anfang der PHP-Datei verschoben werden. Dies würde für die Konzeptualisierung mittels *propose_design* sprechen.

Um dieses Problem zu lösen, existieren zwei Möglichkeiten:

[72] Genau genommen reicht es natürlich nicht aus zu ermitteln, ob die angesprochene Methode zum Zeitpunkt der Äußerungen bereits existiert. Entscheidend ist vielmehr, ob der Sprecher weiß oder annimmt, dass diese Methode existiert bzw. nicht existiert. Dies ist ungleich schwieriger, in manchen Fällen vielleicht sogar gar nicht herauszubekommen. Im vorliegenden Beispiel wurde folgendes Indiz verwendet: Im späteren Verlauf der Sitzung zeigte sich der Sprecher in keiner Weise verwundert, als sein Partner zur betreffenden Methode navigiert.

[73] Der Driver bezieht sich hier auf die PHP-Funktion `header`, mit der ein HTTP-Header (siehe RFC 2616: Hypertext Transfer Protocol – HTTP/1.1 (http://tools.ietf.org/html/rfc2616 (Abruf: 16.02.2009))) gesendet wird.

1. Die fragliche Äußerung wird mit beiden Konzepten annotiert.
2. Nur der zentrale Aspekt wird identifiziert und kodiert.

Da bei der Herleitung der BS/BKM versucht wurde, diese dahingehend zu optimieren, dass Doppelkodierungen nur selten notwendig werden, wurde zu Gunsten des zweiten Ansatzes entschieden. Für Aussagen vom obigen Typ bedeutet dies: Da der Gestaltungsaspekt im Vordergrund steht und der Handlungsvorschlag nur eine Konsequenz aus diesem darstellt, werden sie mit *propose_design* und nicht mit *propose_step* annotiert.

Weitere diese Entscheidung illustrierende Beispiele sind in Tabelle 7.14 und 7.15 zu finden.

HHI-Kode	**Äußerung**	**Kontext/Kommentar**
P2.propose_design	„Wir können einfach, äh, einmal das⌣Ding in 'ne, 'ne normale `id` umwandeln, also `code_to_id` und dann wieder `id_to_code`. (~~)."	Der Observer schlägt eine Modifizierung des Kodes vor, genau genommen die Erweiterung der Bedingung einer `if`-Anweisung. Mittels der Änderung soll verifiziert werden, ob ein übergebener Wert bestimmte Kriterien erfüllt.
P1.agree_design	„O.K."	Der Driver stimmt dem Vorschlag ohne Zögern zu.
-	-	Unmittelbar danach, d.h. ohne nennenswerte Pause und ohne weitere Kommentare, insbesondere ohne eine Ankündigung in Form eines explizit formulierten *propose_step*, beginnt der Driver mit der Implementierung des Vorschlags.

Tab. 7.14: Beispiel aus Sitzung PR1.1 (13:53:56–13:54:08) für einen Designvorschlag, der direkt zu einer Handlung führt. Im Rahmen der BS/BKM werden solche Phänomene nicht separat behandelt. Vielmehr ist vorgeschrieben, einzig den Kode *propose_design* zu verwenden. Weitere Differenzierungen oder Ergänzungen werden spezialisierten Untersuchungen überlassen.

– ***propose_step* vs. *ask_knowledge*:** Der Argumentation auf S. 177 folgend erlaubt es die BKM in aller Regel auch dann, ohne weiteres eindeutig zwischen *propose_step*- und *ask_knowledge*-Phänomenen zu unterscheiden, wenn ein Sprecher Designaspekte in Frageform formuliert. Dies hängt mit der Konstruktion der BKM zusammen. Sie zieht eine Trennlinie zwischen Wissen bez. des weiteren Vorgehens und anderem expliziten Wissen (siehe Abschnitt 7.1.1). Fragen der Form

  – „Wollen wir als nächstes den Arbeitsschritt X durchführen?" bzw.

HHI-Kode	Äußerung	Kontext/Kommentar
P2.propose_design	„Und dann (~) noch der Bereich, wenn wieder gelöscht wird.“	Nachdem der Driver mit dem Einfügen von `updateFriendsLastChangeTime` in der Funktion `setFriendship` fertig ist, schlägt der Observer *P2* vor, diese Methode nun an den Stellen aufzurufen, an denen bestimmte Löschoperationen durchgeführt werden.
P1.amend_design[a]	„O.K., dann können wir uns entweder hier noch an den Hook mit dranhängen, (!!...!!)“	Der Driver konkretisiert den Designvorschlag des Observers. Dabei scrollt er zu der Löschfunktion `deleteFriendsIDsFromMemcache` — in der Äußerung mit „hier“ bezeichnet. In dieser findet sich seiner Meinung nach eine der Stellen, die für das Einfügen eines Aufrufs von `updateFriendsLastChangeTime` prädestiniert sind.
P2.propose_step	„Gucken wir mal wie das aussieht, wenn der User gelöscht wird.“	Der Observer unterbricht den Driver und schlägt vor, sich erst einmal diejenige Funktion anzusehen, mit der ein Benutzer aus dem System gelöscht wird. Es mag sein, dass er im Hinterkopf hat, das Einfügen von `updateFriendsLastChangeTime` lieber dort durchzuführen. Zu erkennen ist dies aber nicht. Vielmehr ist nicht einmal zu sehen, ob seine Äußerung überhaupt Bezug auf die gerade von *P1* gemachte nimmt.
P1.challenge_step	„Wir können auch mal gucken, wo das hier überall verwendet wird“	Unmittelbar nachdem *P2* seinen Vorschlag formuliert hat, widerspricht sein Partner, indem er einen gänzlich anderen Vorschlag äußert. Hierbei zeigt er mit dem Cursor auf die Methode `deleteFriendsIDsFromMemcache` und macht damit deutlich, dass er sich die Stellen im Code ansehen möchte, an denen diese Funktion aufgerufen wird.

[a] Die erste in Tabelle 7.16 aufgeführte *explain_completion*-Äußerung bezieht sich auf diesen Designvorschlag.

Tab. 7.15: Episode aus Sitzung PR1.1 (14:51:32–14:51:43), die sowohl den Unterschied wie auch das Zusammenspiel zwischen Designvorschlägen (*propose_design*) und Vorgehensvorschlägen (*propose_step*) verdeutlicht. Die Ausgangssituation ist die Folgende: Das Paar versucht einen bestimmten, für das Erreichen ihres Sitzungszieles zentralen Funktionsaufruf (`updateFriendsLastChangeTime`) in den bestehenden, bereits vor der Sitzung existierenden Kode einzufügen. Der Aufruf muss an mehreren Stellen, genau genommen in mehreren Funktionen, erfolgen. Welche Stellen dies sind, bzw. ob der bestehende Kode vorher umstrukturiert werden sollte, ist dem Paar noch nicht vollständig klar. Mit der ersten Äußerung wird eine der potentiellen Stellen für die Integration adressiert. Konkret zeigt die Episode, dass Designvorschläge (bzw. Erweiterungen von Designvorschlägen) Arbeitsschritte implizieren (siehe Äußerung 1 bzw. 2). Wie bereits erläutert (S. 201 f), werden diese im Rahmen des BS nicht zusätzlich erfasst. In spezialisierten Untersuchungen könnte dies allerdings notwendig werden. Außerdem wird deutlich, dass das Äußern eines Arbeitsschrittes auch die vorläufige Ablehnung eines Designvorschlages beinhalten kann (siehe Äußerung 3). Dieses Phänomen ist aber in der Regel nicht zweifelsfrei ablesbar. Im Rahmen des BS wird deshalb auf eine entsprechende Annotation verzichtet. Evtl. müssen spezialisierte Untersuchungen auch in solchen Fällen mit den Regeln der BS brechen oder Erweiterungen der BKM vornehmen.

– „Wollen wir jetzt lieber Arbeitsschritt X oder Arbeitsschritt Y machen?“

müssen also mit *propose_step* und nicht mit *ask_knowledge* kodiert werden[74].

Obwohl es sich hierbei dem ersten Anschein nach um eine eindeutige Richtlinie handelt, verbleiben trotzdem Abgrenzungsprobleme:

1. **Implizierte Handlungsankündigungen:** Betrachtet man die Frage

   „Weißt Du, wie die (.), wie ich an die Funktion ran komme, 'ne Methode zu verändern?“

   aus Sitzung PR2.1, mit welcher der Driver allem Anschein nach nur eine bestimmte Funktionalität der IDE Eclipse in Erfahrung bringen will, kommen, wenn man nicht nur die Äußerung allein, sondern auch ihren Kontext betrachtet, Zweifel darüber auf, ob das Phänomen durch eine Kodierung mit *ask_knowledge* ausreichend gut beschrieben ist. Der Sprecher stellt zwar einerseits eine Frage, macht aber andererseits auch erstmals Mitteilung darüber, mit welchem Schritt er fortzufahren gedenkt, nämlich dem Ändern einer Methode.[75] Der Partner nimmt beide Aspekte der Äußerung wahr, denn er antwortet:

   „Das bringt dir doch gar nichts.“

   Die Frage sollte also sowohl mit *ask_knowledge* wie auch mit *propose_step* annotiert werden.[76]

   In Abschnitt 9.3.4 wird festgelegt werden, dass solche Implikationen, sofern sie erkannt werden, immer explizit erfasst werden sollten. Hier kann darüber hinaus schon jetzt festgehalten werden, dass Fragen, speziell solche, die Know-how, also Wissen über prozedurale Vorgänge adressieren, potentiell immer auch Ankündigungen bez. der nächsten Vorhaben beinhalten können.

2. **Einen Handlungsvorschlag fragend in Erinnerung rufen:** Es kommt vor, dass ein in der Sitzung bereits geäußerter bzw. diskutierter, aber dann doch nicht umgesetzter Handlungsvorschlag später wieder

[74] Zur Erinnerung: Da konkrete Aktionen benannt werden, kommt das Konzept *propose_step* und nicht das Konzept *ask_step* zum Einsatz.

[75] Welche Methode der Driver ändern möchte, wird durch den Kontext klar. Während er spricht, öffnet er auf der Signatur einer Methode, genau genommen auf einer Methodendeklaration innerhalb eines Interfaces, ein Kontextmenü. Was er genau ändern will, äußert er nicht. Aus diesem Grund handelt es sich nicht um ein *propose_design*. Weitere Informationen zum Kontext sind auch in Tabelle 7.44 zu finden.

[76] Obwohl der Partner zwei Aspekte wahrnimmt, ist in diesem Fall keineswegs klar, ob dem Sprecher die zweite Informationsebene seiner Äußerung, das Bekanntgeben des nächsten Arbeitsschritts, tatsächlich bewusst war. Es kann also nicht ohne weiteres von einem indirekten Sprechakt im Sinne der Sprechakttheorie (siehe Exkurs 6) ausgegangen werden.

angesprochen, quasi hervorgeholt wird. Dieses Hervorholen kann als Frage formuliert sein, z.B. weil der Sprecher sich nicht mehr hundertprozentig sicher ist, wie der Vorschlag ursprünglich genau ausgesehen hat. Ein Beispiel hierfür ist die folgenden Äußerung aus Sitzung PR2.1:

> „Gut, wir haben gesagt `delete` und ähmm, was war da noch?“

Hier erinnert der Driver, nachdem er gerade eine Reihe von Kommentierungen im Kode vorgenommen hat, daran, dass sich das Paar ursprünglich eigentlich mit Änderungen an einer Reihe bereits existierender PHP-Funktionen (unter anderem `deleteFriendship`, hier verkürzt `delete` genannt) beschäftigen wollte. Er geht aber davon aus, dass seine Erinnerung nicht vollständig ist.[77]

Solche Äußerungen haben zwei Ebenen. Einerseits sind sie Wiederholungen eines Handlungsvorschlags und wären somit mit *propose_step* zu annotieren (siehe auch Erläuterungen zu Wiederholen in Abschnitt 9.3.7), andererseits sind sie wirklich als Frage intendiert. Es wird um zumindest teilweise in Vergessenheit geratene Informationen bez. bereits diskutierter Handlungsvorschläge und nicht nur um eine Bewertung (siehe Tabelle 7.5) solcher gebeten. Sie müssten also mit *ask_knowledge* kodiert werden.

Es wurde festgelegt, dass in diesen Fällen beide Konzepte vergeben werden sollten.[78] Erst spezialisierte Untersuchungen können Abweichungen von dieser Regel notwendig machen, z.B. dahingehend, dass eines der Konzepte als Eigenschaft des anderen zu modellieren ist.

– ***disagree_step* vs. *explain_knowledge*/*explain_finding*:** Die Ablehnung eines vorgeschlagenen Arbeitsschritts wird nicht immer explizit formuliert. Sie kann z.B. in einem Wissenstransfer „verpackt“ sein.

[77] An dieser Stelle könnte man sich die Frage stellen, ob der Sprecher nicht eher eine Strategie (siehe Abschnitt 7.5.4) in Erinnerung rufen möchte, als einen einzelnen Arbeitsschritt. Schließlich scheint es sich bei dem, auf was er sich beziehen möchte, explizit um ein mehrteiliges Vorgehen zu handeln. Betrachtet man allerdings den bisherigen Verlauf der Sitzung, so kann man sehen, dass der letzte, ca. sechs Minuten zurückliegende und vom Observer hervorgebrachte Vorgehensvorschlag nur die Bearbeitung der Funktion `deleteFriendship` adressierte. Die Diskussion des Paares hatte sich zudem zu dieser Zeit auf einer eher taktischen Ebene bewegt. Die Erinnerung des Drivers ist somit – entgegen seiner eigenen Wahrnehmung – vollständig, zumindest wenn davon ausgegangen wird, dass sich der Driver nicht auf einen viel weiter, nämlich ca. 16 Minuten zurückliegenden Designvorschlag beziehen wollte, in dem von der Änderung mehrerer Funktionen gesprochen wird. Anhand der Reaktion des Observers kann dies nicht verifiziert werden. Dieser geht nämlich in keiner Weise auf die Äußerung des Drivers ein. Er schlägt vielmehr etwas gänzlich anderes vor.

[78] Dies stellt im erläuterten Fall kein Problem dar, da der Satz in einen *propose_step*- und einen *ask_knowledge*-Teil aufgeteilt und somit eine evtl. unerwünschte Doppelkodierung vermieden werden kann. Man stelle sich aber vor, der Driver hätte nur „Wir haben `delete` gesagt?“ geäußert. In diesem Fall könnte auf der Ebene des Phänomens keine Trennlinie zwischen den beiden Konzepten gezogen werden.

So antwortetet der Observer im eben gesehenen Beispiel mit „Das bringt dir doch gar nichts“, was, betrachtet man den Kontext, soviel wie „Mach das nicht, weil dir das doch gar nichts bringt“ bedeuten soll. Es handelt sich also um einen indirekten Sprechakt (siehe Exkurs 6), der sowohl als Wissenstransfer (Erklärung) wie auch als Ablehnung (evtl. sogar Warnung) fungiert. Beide Aspekte sind im Rahmen der BKM von Interesse. Somit sollte sowohl der primäre wie auch der sekundäre illokutionäre Akt kodiert, d.h. die Äußerung mit *disagree_step* und *explain_knowledge* bzw. *explain_finding* annotiert werden. Das Phänomen ist ein Beispiel dafür, dass Doppelkodierungen selbst im Rahmen der BS/BKM nicht immer verhindert werden können bzw. sollten (siehe auch Abschnitt 9.3.1).

Analoges gilt für *challenge_step*.

- ***amend_step* vs. *explain_knowledge/explain_finding*:** Im Zusammenhang mit Vorschlägen zum weiteren Vorgehen muss zwischen Beschreibungen der durchzuführenden Aktionen und Erläuterungen bez. des Zwecks dieser Handlungen unterschieden werden. In Bezug auf *amend_step* bedeutet dies, dass nur derjenige Teil einer Äußerung mit diesem Konzept konzeptionalisiert werden darf, der einen zuvor gemachten Vorschlag zum weiteren Vorgehen ergänzt oder detailliert. Anmerkungen zu Zielen, Folgen oder auch Voraussetzungen werden hingegen mit dem Konzept *explain_knowledge* bzw. *explain_finding* annotiert.

- ***ask_step* vs. *ask_knowledge*:** Die Abgrenzung verläuft analog zu allen anderen Abgrenzungen bez. der Klasse *knowledge*: Jede offene Frage bez. eines potentiellen nächsten Arbeitsschritts, die selbst keine konkreten Vorschläge enthält, wird mit *ask_step* und nicht mit *ask_knowledge* konzeptualisiert.

- ***ask_step* vs. *ask_design*:** Einer Frage wie

  „Gut, was wollten wir tun?“ (PR1.1)

  lässt sich nicht ohne weiteres ein passendes Element der BKM zuordnen. Denn anhand der Äußerung allein kann nicht entschieden werden, ob sich der Sprecher auf den Arbeitsprozess oder auf die Gestaltung des Programms bezieht, ob also *ask_step* oder *ask_design* verwendet werden sollte. Wie an vielen anderen Stellen auch, ist auch hier eine Analyse des Kontextes unerlässlich. Hierbei darf aber nicht außer Acht gelassen werden, dass es evtl. gar nicht möglich ist, *ein einziges* passendes Basiskonzept zu finden. Schließlich ist es denkbar, dass der Fragesteller – zumindest unbewusst – alle Antwortoptionen offen halten wollte.

Weitere Anmerkungen zu *step*-Konzepten

Bei Äußerungen der Klasse *step* geht es den Akteuren darum, kurzfristige Zwischenziele zu erreichen. Solche Verhaltensweisen haben „ad hoc"-Charakter (im Sinne von „aus dem Augenblick heraus" [215]) und grenzen sich von dem ab, was als *Strategie* bezeichnet wird und grob gesprochen ein längerfristig ausgerichtetes planvolles Vorgehen in Hinsicht auf ein übergeordnetes Ziel meint.

Strategische Äußerungen werden von der BKM über die Klasse *strategy* adressiert. Diese wird in einem der nachfolgenden Abschnitte im Detail erläutert. In diesem Zusammenhang werden dann auch Abgrenzungsprobleme zu Elementen der Klasse *step* besprochen.

7.5.2 *completion*-Konzepte

Fokus der *completion*-Konzepte

Die Konzepte der Klasse *completion* adressieren Äußerungen zum Grad der Fertigstellung eines momentan in Bearbeitung befindlichen Arbeitsschritts, also solche, in denen der Fortschritt von Tätigkeiten auf taktischer Ebene beurteilt wird. Hierbei kann es sich sowohl um Äußerungen zur Einschätzung bzw. Bewertung von Zwischenständen handeln wie auch um Aussagen bez. des erfolgreichen oder auch nicht erfolgreichen Abschlusses einer Aktion.

Oft stehen diese Äußerungen mit im Vorfeld gemachten Bemerkungen der Klasse *step* oder *design* in Verbindung, referenzieren also zumindest implizit bestimmte explizit formulierte bzw. diskutierte Arbeitsschritte bzw. Gestaltungsoptionen.

Dies muss aber nicht sein. Auch wenn ein Arbeitsschritt nie verbalisiert, d.h. explizit vorgeschlagen wurde, kann es dazu kommen, dass über seinen Fortschritt oder seine Fertigstellung gesprochen wird.

Beispiele für *completion*-Äußerungen sind in Tabelle 7.16 dargestellt.

Identifizierte *completion*-Konzepte und ihre speziellen Eigenschaften

In den analysierten Sitzungen wurden nur drei unterschiedliche *completion*-Konzepte identifiziert: *explain_completion*, *agree_completion* und *challenge_completion* (siehe Abbildung 7.3). Über die bisherigen Erklärungen hinaus gilt für sie das Folgende:

1. **Kurzbewertungen der Situation:** Entgegen dem, was durch die Beispiele in Tabelle 7.16 suggeriert wird, besaßen die meisten bisher beobachteten Äußerungen vom Typ *explain_completion* eine eher einfache Struktur. Sie bestanden aus nur einem Wort.

   Am häufigsten wurde hier der Ausspruch „Ò.K." beobachtet, der in der Regel vom Driver geäußert wurde, sich auf gerade durchgeführte Editiervorgänge bezog und als „Ò.K., das ist fertig" interpretiert werden kann.

#	HHI-Kode	Äußerung	Kontext/Kommentar
1	*P1.explain_-completion*	„Mir gefällt das alles immer noch nicht so richtig gut.“	Die Aussage wurde vom Driver am Ende einer ca. 15-minütigen Episode gemacht, an deren Ausgangspunkt vorgeschlagen worden war, als nächstes eine bereits bestehende Methode in Hinsicht auf eine neue Funktionalität zu modifizieren (*amend_design*). Um sicherzustellen, dass mit der geplanten Änderung die neue Funktionalität bestmöglich und ohne Seiteneffekte integriert wird, beschloss das Paar zu evaluieren, an welcher Stelle bzw. in welcher Weise die zur Modifikation ausgewählte Methode eingesetzt wird.[a] Dies führte zu einer Analyse von Kodestellen, die weder Driver noch Observer bekannt waren. Im Verlauf dieser Untersuchungen wurden dann mehrere Vorschläge dahingehend gemacht, ob und wie die im Fokus stehende Methode zu ändern sei. Nur wenige dieser Vorschläge wurden umgesetzt. Die *completion*-Äußerung fasst die Einschätzung des Drivers über den Erfolg der Bestrebungen des Paares in Hinsicht auf die Integration der neuen Funktionalität zusammen. Es handelt sich also um die Beurteilung des Arbeitsergebnisses bez. des ursprünglich implizit festgelegten Arbeitsschritts.
2	*P1.explain_-completion*	„Also ich würd’ sagen, wir haben das Problem jetzt konzeptuell (.) eingekreist und gelöst.“	Die Einschätzung wurde nach einer ca. dreiminütigen Diskussion über einen für die weitere Programmierung relevanten Aspekt abgegeben. Dabei war die Diskussion nicht als notwendiger Arbeitsschritt expliziert worden, sondern hatte sich aus einem Designvorschlag heraus entwickelt.
3	*P1.explain_-completion*	„Na gut, das war jetzt wirklich nicht wild.“	Die Äußerung wurde vom Driver gemacht, nachdem er eine Reihe von *phpDocumentor*-Kommentaren[b] modifiziert hatte. Unmittelbar vor der Äußerung hatte er sich auf seinem Stuhl zurückgelehnt. Diese Geste macht zusätzlich deutlich, dass er den aktuellen Arbeitsschritt, der allerdings im Vorfeld nicht explizit verbalisiert worden war, als erledigt ansieht.
4	*P2.challenge_-completion*	„Na, wir haben doch noch gar nichts gemacht.“	Reaktion des Observers auf die Äußerung des Drivers aus Beispiel 3. Mit ihr weist er darauf hin, dass die eigentlichen Editierschritte noch zu erledigen sind. Bei der Aussage handelt sich nicht um ein *disagree_completion*[c], da der Sprecher mit einer eigenen, konträren Fortschrittsbewertung kontert.

[a] Details hierzu finden sich in Tabelle 7.15.

[b] phpDocumentor (http://manual.phpdoc.org/ (Abruf: 26.01.2011)) wird eingesetzt, um PHP-Quellcode zu dokumentieren.

[c] Phänomene, die mit *disagree_completion* zu annotieren wären, wurden bisher noch nicht beobachtet.

Tab. 7.16: Beispiele aus der Session PR1.1 für Äußerungen, die mit *completion*-Konzepten annotiert wurden.

2. **Indirekte Bewertung der Situation:** Ein *explain_completion* muss nicht direkt formuliert sein. Es ist auch möglich, dass der Sprecher feststellt, dass bestimmte Dinge nicht mehr getan werden müssen. So äußert der Driver in Sitzung PR1.1 am Ende einer ca. vierminütigen Episode:

   „Da muss man eigentlich hier oben auch gar nicht mehr ran, (~~).“

   Er gibt damit zum Ausdruck, dass er den gerade durchgeführten Arbeitsschritt, der durch einen Designvorschlag des Partners eingeleitet worden war, allerdings nachfolgend nicht erfolgreich umgesetzt werden konnte und deshalb als „TODO“ markiert wurde, als vorläufig abgeschlossen betrachtet.

3. **Bewertung des Arbeitsergebnisses:** Im Rahmen der BKM wird nicht unterschieden, ob eine Äußerung den Fortschritt bez. eines Arbeitsschritts bewertet oder die Qualität des resultierenden Ergebnisses (siehe Beispiel 1 in Tabelle 7.16). In beiden Fällen greift das Konzept *explain_completion*. Eine Differenzierung wird spezialisierten Untersuchungen überlassen. Zusätzliche Erklärungen sollten aber, wie schon im Zusammenhang mit anderen Konzepten diskutiert, einzeln mit *explain_knowledge* oder *explain_finding* annotiert werden.

Sowohl das Konzept *agree_completion* wie auch das Konzept *challenge_completion* wurden in den analysierten Sitzungen nur jeweils ein einziges Mal beobachtet (siehe Tabelle 7.16). Detaillierte Aussagen zu diesen Konzepten können also nicht gemacht werden.

Abgrenzungen von *completion*-Konzepten zu anderen Konzepten

Äußerungen zur Fertigstellung bzw. zum Fortschritt können nicht nur taktische Aspekte – also einzelne Arbeitsschritte – betreffen, sondern auch Strategien. In diesen Fällen werden die Phänomene mit Konzepten der Klasse *state* annotiert. Details hierzu sind in Abschnitt 7.5.5 ausgeführt.

Darüber hinaus ist die Differenzierung zwischen *explain_completion* und *explain_finding* von besonderem Interesse. Sie wird in Abschnitt 7.6.1 (S. 283) erläutert. In wie weit bzw. wann es möglich und sinnvoll ist, zwischen dem Feststellen des Fertigstellungsgrades eines momentan in Bearbeitung befindlichen Arbeitsschritts (also einem *explain_completion*) und einer mit geäußerten Begründung einer solchen Feststellung (einem *explain_knowledge* oder *explain_finding*) zu unterscheiden, wird weiterführenden Untersuchungen überlassen.

7.5.3 *todo*-Konzepte

Fokus der *todo*-Konzepte

Ähnlich wie die Konzepte der Klasse *step* adressieren auch die der Klasse *todo* Äußerungen, die sich mit einzelnen Arbeitsschritten beschäftigen. Im Unterschied

zur erstgenannten Klasse geht es bei *todo*-Äußerungen aber nicht um den nächsten durchzuführenden Schritt, sondern um Tätigkeiten, die nach Meinung von zumindest einem der beiden Programmierer erst in Zukunft zu erledigen sind bzw. auf einen späteren Zeitpunkt verschoben werden sollten. Die Äußerungen werden in der Regel mit dem Ziel gemacht, einen Vermerk zu besitzen, auf den der Sprecher oder das Paar später zurückkommen kann.

Für die Einordnung als *todo*-Phänomen spielt es keine Rolle, ob der Sprecher den thematisierten Arbeitsschritt innerhalb der aktuellen Session oder erst nach dieser in Angriff nehmen will.

Beispiele für Episoden, in denen Äußerungen vom Typ *todo* gemacht werden, sind in den Tabellen 7.17 und 7.18 dargestellt.

HHI-Kode	**Äußerung**	**Kontext/Kommentar**
P1.explain_knowledge	„D[a]◡gibt's 'n Test für.“	Nachdem das Paar über Änderungen diskutiert hat, die vor allem eine bestimmte Klasse betreffen, weist *P1* darauf hin, dass zu dieser Klasse eine JUnit-Testklasse[a] existiert. In diesem Zusammenhang greift er nach der Maus und öffnet im *Package Explorer* der IDE Eclipse das zugehörige Paket.
P1.propose_todo	„(~Müssen◡wir) nur dran denken, n[e]?“	Ohne Pause ergänzt *P1*, dass später nicht vergessen werden darf, die Testklasse anzupassen.

[a] JUnit (http: //www.junit.org/ (Abruf: 26.01.2011)) ist ein Framework zum Testen von Javaprogrammen.

Tab. 7.17: Episode, in der ein Hinweis auf einen später anfallenden Arbeitsschritt geäußert wird. Hierbei wird kein genauer Durchführungszeitpunkt genannt und auch keine Reihenfolge von Tätigkeiten expliziert. Die Äußerung hat vielmehr den Charakter eines Vermerkes. Die Episode stammt aus Sitzung PR2.1 (11:58:46–11:58:49).

Identifizierte *todo*-Konzepte und ihre speziellen Eigenschaften

Im Rahmen der Untersuchungen der Sitzungen ST1.1, PR1.1 und PR2.1 wurden nur wenige (11) *todo*-Phänomene, und zwar nur solche vom Typ *propose_todo* und *agree_todo*, beobachtet. Somit können kaum detaillierte, über die Darstellungen in Abbildung 7.3 hinausgehende Aussagen bez. der einzelnen Elemente dieser Klasse gemacht werden.

Es kann aber festgehalten werden:

#	HHI-Kode	Äußerung	Kontext/Kommentar
1	*P2.amend_strategy*	„Das heißt, eigentlich sollte (~~) gleich tun, vielleicht sollten wir auch erst mal einchecken?“	*P2* hat gerade die Driver-Rolle übernommen und erweitert seine kurz zuvor vorgeschlagene zweiteilige Strategie (siehe S. 219) um einen als erstes durchzuführenden Schritt, das „Einchecken“ des Kodes. Im Laufe der Äußerung wendet er sich seinem Partner zu.
2	*P1.explain_knowledge* + *P1.amend_strategy*/ *P1.disagree_step*	„[A]hm, ich würd’ un(!...!). Da gibt’s wenig Tests. Und ich würd’ ungern einchecken (!!...!!)“	Nach einer zweisekündigen Pause weist *P1* darauf hin, dass es bisher nur wenige JUnit-Tests gibt. Primär scheint es *P1* um die Ablehnung des zuletzt gemachten Vorschlags, der Erweiterung der Strategie um einen als erstes durchzuführenden Schritt, zu gehen. Dem ursprünglichen Strategievorschlag hatte er bereits zuvor zugestimmt, so dass die Verwendung des Konzeptes *disagree_strategy* (bezogen auf die ganze Strategie) nicht zutreffend erscheint.[a]
3	*P2.propose_todo*	„Dann schreiben wir noch ein paar Tests.“	*P2* fällt seinem Partner ins Wort und schlägt vor, noch eine Reihe Tests zu schreiben. *P2* äußert nicht, wann dies geschehen soll. Aus dem weiteren Verlauf dieser Episode wird allerdings ersichtlich, dass dieser Vorschlag nicht im Sinne eines als nächstes durchzuführenden Arbeitsschritts verstanden werden soll bzw. verstanden wird (siehe auch Erörterung auf S. 214).
4	*P1.agree_todo*	„Ja, das sollten wir auf jeden Fall auch noch machen.“	*P1* stimmt dem Vorschlag zu. Dabei spezifiziert auch er nicht genauer, wann dieser Arbeitsschritt ausgeführt werden soll.
5	*P1.amend_strategy*	„Äh, ungern einchecken, bevor ich nicht die GUI ausprobiert hab.“	Unmittelbar nach der eingeschobenen Zustimmung beendet *P1* seinen ursprünglichen Satz. Erst jetzt wird klar, dass er eigentlich ein *amend_strategy* formulieren wollte. Er fordert nämlich einen zusätzlichen Arbeitsschritt.[b]
6	*P2.agree_strategy*	„Gut, dann probieren wir die GUI aus.“	*P2* stimmt zu. Als nächstes wird also die grafische Benutzeroberfläche getestet.

[a] Es wurde ein *disagree_step* kodiert, ohne dass vorher ein *propose_step* annotiert worden ist. Auf solche Kodierungen wird in Abschnitt 7.9 eingegangen. Das *amend_strategy* ist in gewisser Weise „entartet“, da es nichts hinzufügt, sondern etwas wegnimmt.

[b] Auf S. 223f wird noch einmal diskutiert, ob der Sprecher an dieser Stelle wirklich die ursprüngliche, vom Partner formulierte Strategie adressiert oder unabhängig von dieser eine neue vorschlägt, also statt des *amend_strategy* ein *propose_strategy* zu kodieren wäre.

Tab. 7.18: Episode, in der im Rahmen einer Diskussion über eine Strategie „on-the-fly“ ein weiterer, später zu erledigender Arbeitsschritt identifiziert bzw. festgelegt wird. Die Episode stammt aus Sitzung PR2.1 (12:20:27–12:20:46). Details zu den Konzepten der Klasse *strategy* folgen im nächsten Abschnitt.

– Bei allen beobachteten *propose_todo*-Äußerungen handelte es sich im Grunde um Aussagesätze[79] wie z.B. „Müssen wir auch drauf achten“[80] (PR2.1).

– Die geäußerten Vorschläge waren dabei meist wenig konkret und explizierten nie einen bestimmten Zeitpunkt.

– Neben einfachen „ToDo“-Vorschlägen wurden auch Äußerungen beobachtet, in denen die Sprecher einen Arbeitsschritt thematisierten, der nur unter bestimmten, noch nicht final entschiedenen Bedingungen relevant werden würde, also Äußerungen, die weniger Vorschläge als mehr Formulierungen von potentiellen Konsequenzen darstellten. Es wurde entschieden, dass auch solche Phänomene als *propose_todo* zu konzeptualisieren sind (siehe Beispiel in Tabelle 7.19).

Abgrenzungen von *todo*-Konzepten zu anderen Konzepten

Im Rahmen der Verwendung der BKM sollte versucht werden, eindeutig zwischen Konzepten bzw. Phänomenen der Klasse *todo* und solchen der Klassen *step* sowie *strategy* zu unterscheiden. Außerdem sollten Abgrenzungen von *todo*-Konzepten zu einer Reihe von Elementen der Klassen *knowledge* und *finding* im Auge behalten werden. Nachfolgend sind die wichtigsten diesbezüglichen Unterscheidungskriterien dargestellt. Hierbei wird vorerst auf die Erörterungen von Abgrenzungen zu Konzepten der Klassen *strategy* verzichtet. Sie werden diskutiert, nachdem detailliert in die Klassen *strategy* eingeführt worden ist (siehe Abschnitt 7.5.4).

– ***propose_todo* vs. *propose_step*:** Die Gemeinsamkeiten und Unterschiede zwischen diesen beiden Konzepten wurden schon im Rahmen der Einleitung dieses Abschnittes, bei der Diskussion der Abgrenzung der Klassen *todo* zur Klasse *step* angerissen. Im Detail gilt hier: Beide Konzepte werden im Zusammenhang mit Äußerungen verwendet, in denen ein einzelner Arbeitsschritt (evtl. inkl. Alternativen) vorgeschlagen wird, wobei das Verb *propose* darüber hinaus anzeigt, dass der Sprecher weder explizit noch implizit Bezug auf einen anderen, vorhergegangenen Vorschlag nimmt. Der Unterschied zwischen den Anwendungsgebieten der Konzepte besteht nur in Hinsicht auf den Zeitpunkt, für welchen der Sprecher den vorgeschlagenen Schritt einplant. Denn während das Konzept *propose_step* solche Vorschläge adressiert, die sich auf den unmittelbar nächsten Arbeitsschritt beziehen,

[79] Fragesätze oder Aufforderungssätze konnten nicht beobachtet werden.

[80] Mit dieser Aussage ergänzte der Driver eine unmittelbar zuvor bei der Durchsicht einer Klasse geäußerte Einsicht. Diese lautete: „Ohh, da gibt es auch noch ein `Set`.“ Mit seinem *propose_todo* hält er fest, dass zu einem späteren Zeitpunkt daran gedacht werden muss, dass auch Änderungen in Hinsicht auf den gerade entdeckten „Setter“ durchzuführen sind.

HHI-Kode	Äußerung	Kontext/Kommentar
P2.decide_design	„Eigentlich darf es nicht in's `Abstract` rein“	Das Paar hatte im Vorfeld über die Gestaltung der Vererbungshierarchie gesprochen. Konkret ging es darum, ob die Klasse `Feature-AttributeConfiguration-Proxy`, die bisher die abstrakte Klasse `AbstractFeature-AttributeConfiguration` erweiterte (hier vom Sprecher als `Abstract` bezeichnet), dies auch in Zukunft tun soll. Es wurde erörtert, dass eine solche Vererbung nur dann sinnvoll ist, wenn ein Teil der Funktionalität aus der abstrakten Klasse entfernt werden würde. Der Driver *P2* entscheidet schließlich, dass die Vererbung beibehalten und die abstrakte Klasse in Zukunft weniger Funktionalität beinhalten soll.
P1.amend_design/ P1.propose_todo	„Na O.K., dann müssen wir das noch an sehr vielen Stellen implementieren.“	*P1* stimmt der Entscheidung seines Partners zu, weist aber darauf hin, dass in Folge dieser eine Reihe weiterer Implementierungsarbeiten nötig werden. Die entfernte Funktionalität muss nämlich zu einer Reihe anderer Klassen hinzugefügt werden, die diese bisher von der abstrakten Klasse geerbt hatten. Siehe auch S. 216.
-	-	Obwohl der Observer *P1* seine Zustimmung nachfolgend zurückzieht und eine weitere Alternative vorschlägt, bleibt der Driver bei seiner Entscheidung und setzt diese auch umgehend um.

Tab. 7.19: Episode, in welcher sich der Observer (*P1*) bez. einer Gruppe später zu erledigender Arbeitsschritte äußert (als eine Teilaufgabe bzw. ein Schritt formuliert). Im Rahmen seiner Anmerkung verdeutlicht er, dass sich diese zusätzlichen Arbeiten aus einer zuvor getroffenen, bisher allerdings noch nicht umgesetzten Designentscheidung ergeben werden. Obwohl es sich im Kern um eine Zustimmung handelt, könnte das Erwähnen der zusätzlichen Aufwände zur Folge haben, dass die Entscheidung revidiert und somit die zusätzlichen Arbeiten hinfällig werden. Das hier dargestellte *propose_todo*-Phänomen besitzt also eine zusätzliche Funktion, die man als „Konsequenzen beachten“ kodieren könnte. Dies geschieht im Rahmen der BKM nicht, sondern wird spezialisierten Untersuchungen überlassen.

kommt *propose_todo* bei Phänomenen zur Anwendung, in denen der Sprecher einen späteren, allerdings nicht notwendigerweise genau spezifizierten Durchführungszeitpunkt anvisiert.

Folgende Überlegungen können bei der Differenzierung zwischen den beiden Konzepten helfen:

– Wie bereits erwähnt hatten alle bisher beobachteten *propose_todo*-Äußerungen die Eigenschaft, dass der Sprecher keinen konkreten Durchführungszeitpunkt[81] nennt. Dies ist keine zufällige bzw. auf spezielle Eigenschaften der betrachteten Sitzungen zurückzuführende Beobachtung, sondern hängt mit den Begriffsbildungen der BS/BKM zusammen. Denn Vorschläge, die konkrete Zeitpunkte in der Zukunft festschreiben, sind in der Regel weniger taktischer als mehr strategischer Natur und gehören somit meist zur Klasse *strategy* (siehe Abschnitt 7.5.4).

  Als direkte Hinweise auf Äußerungen vom Typ *propose_todo* können also vor allem – zumindest theoretisch – adverbiale Bestimmungen der Zeit wie „dann“, „später“ oder „demnächst“, also solche, die den konkreten Zeitpunkt offen lassen, dienen. Die Verwendung solcher Begriffe konnte allerdings bisher noch nicht beobachtet werden. Vielmehr wurden alle Entscheidungen zugunsten des Konzeptes *propose_todo* anhand anderer Indizien, oft auch anhand solcher, die nur im Kontext der Äußerung zu finden waren, getroffen.

  Ein Beispiel ist in Tabelle 7.18 zu finden. Hier bezieht sich das „dann“ auf den Grund für den Vorschlag, nämlich den Umstand, dass nur wenige Tests existieren, ist also keine adverbiale Bestimmung der Zeit. Auch sonst enthält die Äußerung keinen konkreten Hinweis auf einen späteren Durchführungszeitpunkt. Trotzdem wurde sie mit *propose_todo* kodiert. Diese Entscheidung ergab sich aus der Analyse des Kontextes: Der Sprecher hatte ursprünglich etwas anderes vorgeschlagen („Einchecken von Kode“). Mit seiner nachfolgenden Äußerung widerruft er dieses Vorhaben nicht. Es scheint ihm vielmehr darum zu gehen, den Partner zu „beruhigen“, um nachfolgend sein ursprüngliches Anliegen verfolgen zu können. Diese Annahme stützt sich auf zwei Beobachtungen:

  1. Der Sprecher reagiert auf die Bedenken seines Partner, indem er ihm ins Wort fällt. Dies lässt den Eindruck entstehen, dass es ihm vorrangig darum geht, den Einwand des Partners schnell „vom Tisch“ zu bekommen.

[81] Der Begriff „konkreter Durchführungszeitpunkt“ meint hier weniger eine bestimmte Uhrzeit, als mehr eine Position im weiteren Ablauf.

2. Auch nach seiner Äußerung bleibt sein Blick auf den Partner gerichtet. Er scheint sicher gehen zu wollen, dass nun kein weiterer Widerspruch geäußert wird.

Darüber hinaus liefert auch die Reaktion des Partners ein Indiz dafür, dass es sich um ein *propose_todo* handelt bzw. von diesem als ein solches angesehen wird.[82] Dieser verwendet nämlich die Phrase „auch noch“, welche auf einen späteren Durchführungszeitpunkt hindeutet.

Bei der hier dargestellten Interpretation handelt es sich natürlich nicht um einen hieb- und stichfesten Beweis, denn letztendlich hätte nur eine direkte Nachfrage jegliche Unsicherheiten bez. der Intention des Sprechers aus dem Weg räumen können. Diese Tatsache stellt aber, speziell in Hinsicht auf weiterführende Untersuchungen, ein minderschweres Problem dar. Schließlich kommt es nicht unbedingt darauf an, ob eines von vielen Phänomenen mit X oder Y zu etikettieren ist. Viel wichtiger sind die Erkenntnisse, die im Zusammenhang mit diesbezüglichen Betrachtungen erlangt und sukzessive in eine Theorie verwandelt werden.

Hier hilft die BS/BKM als Einstieg, indem sie ein Rahmenwerk zur Verfügung stellt, über dessen Anwendbarkeit in jedem Einzelfall immer wieder neu entschieden werden muss. Dies bedeutet (wie bereits mehrfach erläutert), dass Anpassungen vorgenommen werden sollten, falls es sich als notwendig erweist. Anders ausgedrückt: Die Daten sowie die Erkenntnisse, die aus diesen gezogen werden können, stehen in jedem Moment der Analyse im Vordergrund. Die BS/BKM ist „nur“ ein Werkzeug, welches im Rahmen des Prozesses weiter entwickelt wird, indem in ihm viele der neuen Erkenntnisse subsumiert werden. Diese Weiterentwicklung kann z.B. dadurch vonstatten gehen, dass versucht wird herauszufinden, welche zusätzlichen Funktionen das hier angesprochene „ToDo“ im Speziellen bzw. „ToDos“ im Allgemeinen besitzen.

– Die zentralen Eigenschaften des Konzeptes *propose_step* wurden ausführlich auf S. 194 dargestellt. Diese Erläuterungen sollten bei der Differenzierung zwischen Phänomenen vom Typ *propose_step* und solchen vom Typ *propose_todo* herangezogen werden. Zusammengefasst sprechen folgende Charakteristika dafür, dass es sich um ein *propose_step* handelt:

  1. Es wird eine adverbiale Bestimmungen der Zeit wie „jetzt“ oder „erst mal“ verwendet.
  2. Die Formulierung hat die Form eines Aufforderungssatzes (siehe Beispiel 5 in Tabelle 7.10), bzw. allgemeiner, die Äußerung kann

[82] Wie schon mehrfach angedeutet muss es nicht immer so sein, dass die Intention eines Sprechers vom Partner „richtig“ interpretiert wird. Die Probleme, die sich diesbezüglich für die Anwendung der BKM ergeben, werden ab S. 416 näher erläutert.

als Handlungsaufforderung verstanden werden (siehe Beispiel 4 in Tabelle 7.10).

3. Die im Vorschlag formulierte Handlung wird unmittelbar durchgeführt.

– ***propose_todo* vs. *explain_knowledge/explain_finding*:** Wie schon mehrfach erläutert ist die BKM derart konstruiert, dass bestimmte Typen von Wissen nicht durch eine allgemeine Wissensklasse, sondern durch spezialisierte Klassen wie z.B. *step* (siehe z.B. S. 199) behandelt werden. Zu diesen bez. Wissen spezialisierten Klassen gehört auch *todo*.[83]

– ***propose_todo* vs. *amend_design/propose_design***: Die Erweiterung oder Konkretisierung eines Designvorschlags kann mit der Formulierung eines in Zukunft zu bearbeitenden Arbeitsschritts einhergehen (ein Beispiel findet sich in Tabelle 7.19). Um der besonderen Rolle der *propose_todo*-Äußerungen als „Merker“ gerecht zu werden, sollte in diesen Fällen neben *amend_design* auch *propose_todo* annotiert werden.

  Potentiell ist dies auch bei anderen *design*-Konzepten, z.B. *propose_design*, notwendig.

7.5.4 *strategy*-Konzepte

Fokus der *strategy*-Konzepte

Im Gegensatz zur Klasse *step*, mit der Äußerungen konzeptualisiert werden, in denen es um die Festlegung eines einzelnen Arbeitsschritts geht, beziehen sich die Elemente der Klasse *strategy* auf Phänomene, in denen sich der Sprecher bez. eines längerfristig ausgerichteten, planvollen und in der Regel aus mehreren komplexeren Einzelschritten bestehenden Handelns äußert. Die *strategy*-Konzepte adressieren also diejenigen Teile eines Dialoges (oder auch Monologes), in denen die Entwickler versuchen, ihr Vorgehen für einen gewissen Zeitraum bzw. bez. komplexerer inhaltlicher Aspekte oder eines übergeordneten Zieles[84] festzulegen oder zu koordinieren.

Dieser Strategiebegriff wurde nicht aus vorhandenen Theorien übernommen, sondern genauso wie alle anderen Objekte der BKM direkt aus den in Abschnitt

[83] In diesem Zusammenhang stellt sich die Frage, ob es nicht besser wäre, von *explain_todo* anstatt von *propose_todo* zu sprechen. Denn eigentlich handelt es sich bei den bisher beobachteten *todo*-Äußerungen weniger um Vorschläge als um Informationen bzw. Erkenntnisse. Da aber eine Gleichförmigkeit ähnlicher Klassen, also z.B. aller Klassen, die Äußerungen bez. des Prozesses adressieren, den Umgang mit der BKM erleichtert, wurde zugunsten des Begriffs *propose_todo* entschieden. Schließlich kann das Verb *propose* auch im Zusammenhang mit der Klasse *step* Äußerungen adressieren, die nur als Informationen intendiert waren.

[84] Der Begriff „übergeordnetes Ziel“ wird auf S. 229 näher erläutert. Vorerst soll ein intuitives Verständnis ausreichen. Wichtig ist nur festzuhalten, dass es dafür, dass eine Äußerung als Strategie klassifiziert werden kann, nicht notwendig ist, dass der Sprecher das übergeordnete Ziel expliziert.

4.2 beschriebenen Sitzungsdaten extrahiert. Hierbei wurden drei Typen von Strategievorschlägen identifiziert:

- **Typ OWP – Organisation von Arbeitspaketen (*organising work packages*)**: Die vorliegende Aufgabe oder Teile der vorliegenden Aufgabe werden in einzelne konkrete und in der Regel nicht-atomare Arbeitsschritte, so genannte Teilaufgaben (siehe auch Abbildung 7.8 auf S. 236), zerlegt. In diesem Zusammenhang wird auch fast immer eine Bearbeitungsreihenfolge festgelegt.
- **Typ DPR – Festlegung von Vorgehensregeln (*determining procedure rules*)**: Anstatt dass ein konkreter Plan formuliert wird, der aus einzelnen, im Großen und Ganzen wohldefinierten Schritten besteht, werden Regeln bzw. Leitlinien vorgeschlagen, die zumeist in einer kontinuierlichen Art und Weise helfen sollen, nachfolgende Arbeiten durchzuführen oder zu strukturieren. Ziel dieser Leitlinien ist es häufig, den Prozessverlauf bzw. die durchzuführenden Schritte zu vereinfachen, z.B. dadurch, dass bestimmte Arten von Tätigkeiten (z.B. das Schreiben von Tests) bis auf weiteres nicht durchgeführt oder bestimmte Randbedingungen vorerst nicht berücksichtigt werden.
- **Typ EXS – Expansion einzelner Arbeitsschritte zu einer Strategie (*expansioning step*)**: Manchmal passiert es, dass Äußerungen vom Typ *step* oder *design* – oft auch solche des Partners – zu einer Strategie (in der Regel vom Typ OWP) ausgebaut werden. Der Sprecher äußert hierbei einen einzelnen Arbeitsschritt, betrachtet diesen aber im Kontext eines zuvor gemachten Vorschlags und fügt beides, zumindest implizit, zu einer Strategie zusammen, also zu etwas mit einem übergeordneten Ziel. So kann der Sprecher z.B. betonen, dass nachfolgend nur noch eine bestimmte Tätigkeit X durchgeführt werden muss, um zu einem (mehr oder weniger wohldefinierten) Zwischen- oder Endzustand zu kommen.

Die beschriebenen Typen helfen dabei, den Strategiebegriff der BKM besser zu verstehen und entsprechende Phänomene leichter in den Daten entdecken zu können. Sie bilden aber nur eine sehr grobe Klassifizierung, bei der es, wie noch gezeigt werden wird, passieren kann, dass einzelne Strategievorschläge nicht eindeutig einem bestimmten Typ zugewiesen werden können.

Um die Identifizierung von Strategieäußerungen darüber hinaus zu erleichtern, sollten deshalb zwei weitere Eigenschaften von *strategy*-Phänomenen beachtet werden:

1. **Art der Darstellung:** Eine Strategie im Sinne der BKM kann extensional oder intensional formuliert sein. Es werden also entweder einzelne Schritte bzw. Teile separiert und explizit aufgezählt, oder es wird ein Oberbegriff bzw. ein Kriterium genannt, welches die Einzelteile zusammengefasst darstellt. Strategien vom Typ OWP und EXS sind meist extensional formuliert,

Strategien vom Typ DPR intensional. Dies muss aber nicht sein. Z.B. kann die folgenden Äußerung als intensional formulierte Strategie vom Typ OWP interpretiert werden:

> „Äh, wollen wir hier, ähm, diese Anleitung zum Ausführen von, ähm, XPetstore uns angucken und danach vorgehen?“ (ST1.1)

Denn obwohl die Aussage oberflächlich betrachtet aus der Aufzählung der Schritte „Anleitung ansehen“ und „Nach Anleitung vorgehen“ besteht, zielt sie im Kern darauf, dass die Entwickler einer festen Liste von Anweisungen (zu erledigende Teilaufgaben) folgen sollen. Hierbei zählt der Entwickler die einzelnen Schritte aber nicht explizit auf.

Das Beispiel verdeutlicht darüber hinaus, dass die Unterscheidung zwischen den Typen OWP und DPR nicht immer hundertprozentig eindeutig vorgenommen werden kann. Schließlich ist es möglich, eine vorgegebene Liste als Regel zu interpretieren. Dieses Problem ist typisch für intensional formulierte Strategien vom Typ OWP.[85]

2. **Reichweite der Festlegung:** Strategien im Sinne der BKM können sowohl langfristig wie auch mittelfristig angelegt sein.[86] Eine Strategie kann sich entweder auf die gesamte zu erledigende Aufgabe bzw. deren verbleibenden Rest beziehen, also einen Blick auf das „große Ganze“ werfen (langfristig), oder nur eine Reihe von als nächstes anstehenden Arbeitsschritten oder als nächstes zu bearbeitenden Teilaufgaben, also einen abgegrenzten Bereich, adressieren (mittelfristig).[87] Hierbei ist es in Hinsicht auf die Abgrenzung zu Konzepten der Klasse *step* wichtig festzuhalten, dass sich die Begriffe mittel- und langfristig nicht primär auf die letztendlich notwendige Arbeitszeit, sondern nur auf die Art der Arbeitsplanung beziehen. Es ist aber wahrscheinlich, dass die Umsetzung einer mittelfristigen Strategie mehr Arbeitszeit in Anspruch nimmt als das Ausführen eines einzelnen Arbeitsschritts.[88]

   Einen Sonderfall bilden hierbei die Strategien vom Typ DPR. Sie sind oft derart formuliert, dass sie „bis auf weiteres“ gelten sollen, ohne dass spezifiziert wird, was dies genau bedeutet.

[85] Es wurden bisher nur solche intensional formulierten Strategien vom Typ OWP beobachtet, die auch als Strategien vom Typ DPR interpretiert werden könnten. Es ist fraglich, ob überhaupt andere intensional formulierte Strategieäußerungen vom Typ OWP vorkommen können.

[86] Die Bezeichnung „kurzfristige Strategie“ wird im Rahmen der BKM nicht verwendet. Bezieht sich eine Äußerung nur auf den nächsten Arbeitsschritt, so wird sie, wie bereits erläutert, mit Konzepten der Klasse *step* annotiert. Ob hieraus gefolgert werden kann, dass der Unterschied zwischen Arbeitsschritten und Strategien nur ein gradueller ist, wird später diskutiert werden.

[87] Wenn der verbleibende Rest der zu erledigenden Aufgabe nur noch einen kleinen Teil der Gesamtaufgabe ausmacht, sind Strategien, die sich auf diesen Teil beziehen, natürlich nicht mehr als langfristig zu bezeichnen.

[88] Genau genommen handelt es sich hier um eine noch zu verifizierende Hypothese.

Es muss noch einmal festgehalten werden, dass die hier erörterten Typen und Eigenschaften nicht in jedem Fall eine eindeutige Kategorisierung erlauben. Beispielsweise können Strategieäußerungen sowohl intensional wie auch extensional formulierte Teiläußerungen beinhalten oder es kann wie bereits gesehen dazu kommen, dass nicht – zumindest nicht ohne weiteres – eindeutig zwischen Typ OWP und Typ DPR unterschieden werden kann. Außerdem ist der Übergang zwischen den Begriffen „mittelfristig“ und „langfristig“ fließend.

Dies ist aber – zumindest was die BKM betrifft – weitgehend unerheblich, da der Wert der beschriebenen Typen und Eigenschaften an einer anderen Stelle zu finden ist. Sie gehören nämlich zu den primär charakterisierenden Eigenschaften und helfen dabei, dass sich der Forscher über die Bandbreite möglicher Strategieäußerungen bewusst werden kann. Es wird ihm somit erleichtert, diese zu identifizieren. Darüber hinaus bilden sie einen Ausgangspunkt für weiterführende Überlegungen in spezialisierten Untersuchungen.

Beispiele für Äußerungen der Klasse *strategy* sind in Tabelle 7.20 zu finden. Details bez. der Abgrenzung zur Klasse *step* werden im Zusammenhang mit der Erörterung der Differenzierung zwischen *propose_strategy* und *propose_step* ausgeführt (siehe S. 229).

Identifizierte *strategy*-Konzepte und ihre speziellen Eigenschaften

Wie bereits beschrieben wurden sieben *strategy*-Konzepte identifiziert: *propose_strategy*, *agree_strategy*, *disagree_strategy*, *decide_strategy*, *amend_strategy*, *challenge_strategy* und *ask_strategy* (siehe Abbildung 7.3). Über die bisherigen Erklärungen hinaus sollte das Folgende berücksichtigt werden:

1. **Vorschlagstypen/Ziele von Vorschlägen:** Neben der geschilderten Differenzierung in die Typen OWP, DPR und EXS und der Unterscheidung zwischen langfristig und mittelfristig können *propose_strategy*-Äußerungen auch in Hinsicht auf die in Tabelle 7.5 geschilderten Vorschlagstypen untersucht werden, wobei in den bisher analysierten Sitzungen Strategievorschläge vom Typ „Weitere Meinung einholen“ sowie vom Typ „Information liefern/Anweisung geben“ beobachtet werden konnten. Ein Beispiel für den ersten Typ liefert die folgende Aussage:

   > „Jetzt würd’ ich gern (.) noch ’ne Methode einbauen, mit der man die so roh, wie sie sind, abholen kann, auch. (.) Und, dann würd’ ich gern mal die Sache mit dem Dialog und so mal kurz anschauen“[89] (Kontext: Der Sprecher wendet sich im Verlauf seiner Äußerung dem Partner zu. Es ist zu erkennen, dass er auf diese Weise deutlich machen will, dass er einen Kommentar erwartet. (PR2.1))

[89] Der Strategievorschlag ist mittelfristig ausgelegt und vom Typ OWP, wobei der erste Schritt als grober Designvorschlag formuliert ist. Eine Doppelkodierung scheint an dieser Stelle naheliegend. Auf S. 231 wird hierauf genauer eingegangen.

#	Session und HHI-Kode	Äußerung	Kontext/Kommentar	Ggf. Typ
1	*ST1.1: P2.ask_strategy*	„Wie fangen wir da an?“	Zu Beginn der Sitzung befragt *P2* seinen Partner nach einem geeigneten Einstieg. Die Äußerung zielt nicht offensichtlich auf die Festlegung einer Strategie. Auf den ersten Blick scheint es vielmehr um die Festlegung von taktischem Verhalten zu gehen. Allerdings antwortet *P1* mit einem Strategievorschlag (siehe Beispiel 2). Aus diesem Grund wurde die Äußerung der Klasse *strategy* zugeschrieben.[a]	-
2	*ST1.1: P1.propose_strategy*	„Also erst mal würd’ ich den `EmailSpy` komplett außen vor lassen. Und erst mal nur den⌣den XPetstore umstellen.“	Der Entwickler antwortet auf die unter Punkt 1 aufgeführte Frage seines Partners. Mit der Antwort teilt er das weitere Vorgehen in zwei Schritte auf: Zuerst sollen die Klassen des XPetstore vollständig umgestellt, danach an der neuen Funktionalität `EmailSpy` gearbeitet werden.	OWP /l
3	*ST1.1: P1.propose_strategy*	„Weißt Du, was wir jetzt machen? Unseren vorhandenen `EmailSpy` benutzen wir zum Debuggen.“	Der Driver legt eine Strategie fest, wie vorgenommene Kodeänderungen im weiteren Verlauf getestet werden können. Hierbei soll ein Programm verwendet werden, welches vom Paar im Vorfeld der Sitzung – allerdings zu einem anderen Zweck – entwickelt worden war. Dieses spricht genau die Funktionalitäten an, die momentan vom Paar editiert werden. Das Paar erspart sich hierdurch kompliziertere Testaufrufe.	DPR /m
4	*PR2.1: P1.propose_strategy*	„Und ich pass’ da jetzt nicht groß irgendwelche Tests von irgendwelchen Leuten an.“	Intensional formulierter Strategievorschlag, welcher vom Driver im Verlauf eines Editiervorganges geäußert wurde. Es handelt sich um eine Vermeidungsstrategie, welche die Einzelschritte des weiteren Vorgehens mitbestimmt.	DPR /m
5	*PR1.1: P1.propose_strategy*	„Und dann müssen wir nur noch dieses Ding überall da einbauen wo (~sich Freunde ändern).“	P1 ergänzt einen Vorschlag des Partners um einen weiteren Arbeitsschritt. Mit der adverbialen Bestimmung „nur noch“ stellt er heraus, dass man auf diese Weise zum Ziel gelangt.	EXS /m

[a] Weitere Anmerkungen hierzu sind auf S. 232 zu finden.

Tab. 7.20: Beispiele aus den Sessions ST1.1, PR1.1 und PR2.1 für Äußerungen, die mit *strategy*-Konzepten annotiert wurden. Es sind sowohl extensional wie auch intensional formulierte Strategien berücksichtigt. In der letzten Spalte werden Strategievorschläge nach Typ und Reichweite (l ≡ langfristig; m ≡ mittelfristig) klassifiziert.

Ein Beispiel für den zweiten Typ ist unter Nummer 4 in Tabelle 7.20 zu finden.

Auch wenn nicht immer eindeutig entschieden werden kann, welchem Vorschlagstyp eine bestimmte Äußerung zuzurechnen ist, hilft die Berücksichtigung der durch die Typen abgedeckten Variationsbreite bei der Identifikation einzelner Strategievorschläge. Außerdem können auch diese Typen als Ausgangspunkt für die in spezialisierten Untersuchungen notwendig werdenden Differenzierungen verwendet werden.

Wie schon bei Designvorschlägen gesehen sollte auch bei der Identifizierung von Strategievorschlägen im Auge behalten werden, dass diese nicht immer explizit als solche formuliert sind, die Illokution also nicht immer direkt aus der Art der Formulierung abgelesen werden kann.

2. **Vorschläge mit Alternativen:** Genauso wie alle anderen *propose*-Phänomene können auch Strategievorschläge Alternativen beinhalten. Als Beispiel sei hier eine Äußerung aus Sitzung PR2.1 aufgeführt:

   > „Dann sollten wir das zuerst machen oder danach. Aber nicht gleichzeitig."

   Hier schlägt der Sprecher zwei Möglichkeiten vor, wann gerade beschlossene Restrukturierungen bestimmter Teile des Kodes durchgeführt werden sollten. Beide zielen darauf, dies nicht parallel zu den anderen Implementierungstätigkeiten zu tun. Weitere Details zum Kontext der Äußerung sind in Tabelle 7.22 zu finden.

3. **Zustimmung vs. Auswahl:** Für die Differenzierung zwischen *decide_strategy* und *agree_strategy* gilt dasselbe, wie für die zwischen *decide_design* und *agree_design* (siehe S. 174).

4. **Sekundäre Inhalte:** Strategievorschläge beziehen manchmal auch Aspekte mit ein, die nicht unmittelbar etwas mit der Bewältigung der aktuellen Aufgabe zu tun haben. Hierbei kann es sich z.B. um Unterbrechungen aufgrund anderer Verpflichtungen oder um Pausen handeln. Strategievorschläge reichen also in Einzelfällen über die aktuelle Session hinaus. Eine diesbezügliche Beispielepisode ist in Tabelle 7.21 und Abbildung 7.6 dargestellt.

5. **Unterschiedliche Formen der Präzisierung:** Eine Äußerung vom Typ *amend_strategy* kann einen vorhergegangenen Strategievorschlag auf zwei unterschiedliche Weisen ergänzen:

   (a) Die geäußerte Strategie, vornehmlich eine solche vom Typ OWP, wird um einen oder mehrere zusätzliche Schritte erweitert (siehe z.B. Äußerung 1 in Tabelle 7.18).

HHI-Kode	Äußerung	Kontext/Kommentar
P1.explain_knowledge + *P1.propose_strategy*	„Aber wir haben ja eh in zwei Minuten (.) Stand-up. (~Und) wenn wir dann essen gehen. Das heißt, wir haben auch noch Zeit, das im Kopf (~~).“	*P1* erläutert zuerst ein Faktum, um darauf aufbauend eine Strategie vorzuschlagen. Diese ist nur rudimentär formuliert. Das Ende der Äußerung ist unverständlich, da *P2* ins Wort fällt. Mit dem Artikel „das“ (in „das im Kopf“) bezieht sich *P1* auf Inhalte der vorangegangenen Diskussion.
P2.explain_standard of knowledge[a]	„Du willst direkt nach dem Stand-up essen, essen gehen und‿damit, und‿damit jetzt aufhören?“	*P2* fällt *P1* ins Wort um zu erklären, wie er die bisherigen Worte des Partners verstanden hat.
P1.propose_strategy[b]	„Ich will direkt nach dem Stand-up essen gehen.“	*P1* wartet nicht ab, bis *P2* seine Äußerung fertig formuliert hat. Er wiederholt noch einmal seinen gerade geäußerten Vorschlag.[c] *P1* ist mit seiner Äußerung fertig, noch bevor *P2* seine Formulierung abgeschlossen hat. Dieser braucht hierfür noch eine gute Sekunde.
P1.amend_strategy	„Ne, ich will nach'm Essen weitermachen (~wollen), mit dem Arbeitspunkt.“	Direkt nachdem *P2* aufgehört hat zu sprechen, konkretisiert *P1* seinen Vorschlag.
-	-	Es wird eine nebensächliche Frage diskutiert. Hier aus Platzgründen nicht wiedergegeben.
P2.agree_strategy	„O.K., ich hätt' halt gern fast noch (.) 'n bisschen weiter gemacht. Aber, O.K.“	*P2* stimmt dem Vorschlag zu. Die Zustimmung kann als „sich fügen“ gedeutet werden.
P1.explain_state[d]	„Also, wir sind halt vom Einchecken usw. meiner Meinung nach an 'nem Punkt, wo wir 'n Break machen können.“	*P1* erklärt, an welchem Punkt des Gesamtprozesses sich das Paar seiner Meinung nach befindet. Im Kern handelt es sich um eine Fortschrittsbewertung und nicht um einen Vorgehensvorschlag.
P2.agree_state	„Das stimmt natürlich. [...]“	

[a] Das Konzept *explain_standard of knowledge* wird in Abschnitt 7.6.3 erläutert.
[b] Auf S. 228 wird erläutert, warum es sich nicht um ein *propose_todo* handelt.
[c] In Abschnitt 9.3.7 wird erläutert, wie mit Wiederholungen umgegangen wird.
[d] Details zur Verwendung der Konzepte der Klasse *state* werden in Abschnitt. 7.5.5 erläutert.

Tab. 7.21: Beispiel für eine Episode (PR2.1: 12:58:55–12:59:38), in der Details eines Strategievorschlags geklärt werden. Dem Vorschlag ging eine längere Diskussion über Art und Zeitpunkt eines potentiell durchzuführenden „Refactorings“ voraus. Weitere Details zum Ablauf der Episode sind in Abbildung 7.6 dargestellt.

(b) Die bereits vorgeschlagenen Schritte (bei Typ OWP oder EXS) oder Regeln (bei Typ DPR) werden detailliert.

Als Beispiel sei auf ein Phänomen verwiesen, welches sich kurz nach der in Tabelle 7.21 dargestellten Episode ereignete. Hier konkretisierte *P1* sein Vorhaben, nach dem „Stand-up“ essen gehen zu wollen, indem er äußerte:

> „Um hier weiter zu gehen, (.) fänd' ich auch nicht dumm, 'ne halbe Stunde (~im) Kopf darüber zu reflektieren, (~wie) man wirklich (..) (~das) Weg gerne gehen möchte?‿Oder gehen wir lieber den vorsichtigeren Weg wie Du?“[90]

Solche Konkretisierungen können dazu führen, dass einzelne Schritte weiter zerlegt werden, bzw. geplante aber nicht ausführlich erläuterte Zerlegungen expliziert werden (siehe Abbildung 7.6).

6. **Strategievorschläge gegeneinander abgrenzen:** Bei der Analyse von Diskussionen über extensional formulierte Strategien vom Typ OWP kommt es vor, dass die Konzeptualisierung einzelner Äußerungen mit Elementen der Klasse *strategy* Probleme bereitet. Insbesondere ist es oft nicht ohne weiteres möglich zu entscheiden, ob es sich bei einer Äußerung um die Erweiterung einer bereits formulierten oder um den Vorschlag einer zusätzlichen oder evtl. sogar alternativen Strategie handelt (*amend_strategy* vs. *propose_strategy* vs. *challenge_strategy*). Dieses Problem hängt vorrangig damit zusammen, dass in Strategiediskussionen in der Regel nur auf einzelne Teile einer Strategie und nicht auf das Ganze Bezug genommen wird.

   Folgende Heuristik kann bei der Lösung solcher Entscheidungsprobleme helfen:

   – Wenn der Sprecher signalisiert, dass er grundsätzlich mit allen Teilschritten der vorgeschlagenen Strategie einverstanden ist und nur einzelne Schritte hinzufügen oder detaillieren möchte, sollte *amend_strategy* verwendet werden.

[90] In diesem Zusammenhang stellt sich die Frage, ob es sich bei der Äußerung wirklich um eine Detaillierung des Schrittes „essen gehen“ der ursprünglich vorgeschlagenen Strategie handelt, oder um eine Begründung für eben diesen, also ein *explain_knowledge*. Da davon auszugehen ist, dass das Nachdenken der eigentlich relevante Teil des vorgeschlagenen Schrittes ist, wurde zugunsten des Konzeptes *amend_strategy* entschieden.

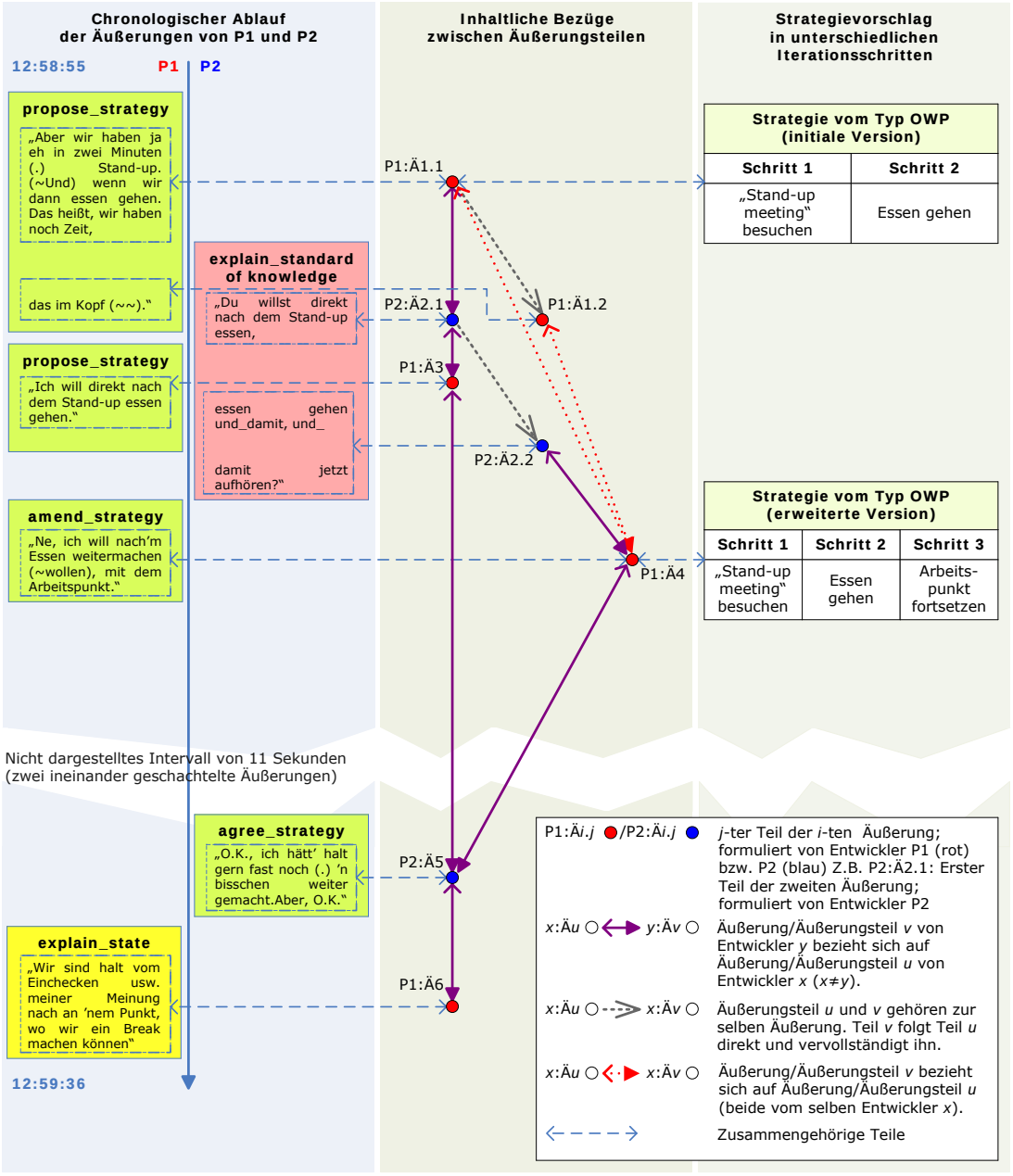


Abb. 7.6: Darstellung einer Strategiediskussion (siehe auch Tabelle 7.21). Neben der Chronologie sind inhaltliche Bezüge von (Teil-)Äußerungen veranschaulicht. Es fallen zwei Dinge auf: 1. Äußerungen können sich auf mehr als eine andere Äußerung beziehen. Dies liegt hier unter anderem daran, dass sich die Entwickler gegenseitig ins Wort fallen. Folge: Der Strategievorschlag ist erst nach einer Unterbrechung und einem nachfolgenden *amend_strategy* vollständig ausformuliert. 2. Abschließend bewertet *P1* den bisherigen Fortschritt. Diese Bewertung fungiert als Begründung. *P1* hält sie wohl deshalb für notwendig, weil sein Partner seine Zustimmung mit einem relativierenden Zusatz versehen hatte. Eine auf diese Weise abgemilderte Form von Zustimmung könnte dazu verleiten, zumindest Teile der Äußerung mit *disagree_strategy* oder auch *propose_step* zu annotieren. Eine solche Kodierung würde dem Phänomen allerdings kaum gerecht werden, da es sich trotz des Zusatzes im Kern um eine Zustimmung handelt.

– Wenn er entscheidende Schritte[91] der Strategie ablehnt oder durch andere ersetzen möchte, sollte *challenge_strategy* annotiert werden, es sei denn, die Ablehnung einzelner Schritte ist einer Ablehnung der gesamten Strategie gleichzusetzen. In diesem Fall muss *disagree_strategy* annotiert werden (siehe auch S. 226).

– Wenn er einen einzelnen Schritt herausgreift, diesen in Teilschritte aufspaltet und alle anderen ursprünglich geäußerten Schritte außer Acht lässt, muss geprüft werden, ob es sich um die Formulierung einer neuen Strategie handelt, ob also die ursprüngliche, längerfristig ausgelegte Strategie durch eine mittelfristige ergänzt wird.

Diese Überlegungen helfen aber nicht in allen Fällen weiter, da die in den obigen Regeln angegebenen Voraussetzungen nicht immer eindeutig identifiziert werden können. So kann beispielsweise nicht immer ermittelt werden, ob einige der ursprünglich formulierten Schritte im Laufe der Diskussion wirklich außer Acht gelassen wurden. Ein Beispiel hierfür ist in Tabelle 7.18 zu finden: Nach einem Strategievorschlag vom Typ OWP (1. „Weitere Methode einbauen“; 2. „Dialog ansehen“) wird dieser von *P2* um einen zusätzlichen, als erstes durchzuführenden Schritt („Einchecken“) ergänzt. In der folgenden kurzen Diskussion geht es dann nur noch darum, ob diesem nachträglich ergänzten Schritt noch ein weiterer („GUI ausprobieren“) vorausgehen sollte. Die anderen ursprünglich vorgeschlagenen Schritte geraten dabei soweit aus dem Blickfeld, dass am Ende der Episode nicht mehr eindeutig zu erkennen ist, ob diese Schritte zu diesem Zeitpunkt noch von beiden Personen als Bestandteil eines dann vierteiligen Planes (1. „GUI ausprobieren“; 2. „Einchecken“; 3. „Weitere Methode einbauen“; 4. „Dialog ansehen“) angesehen werden oder ob sich das Paar lediglich auf die von

[91] Die Entscheidung darüber, welche Rolle ein einzelner Schritt im Rahmen einer Strategie spielt, kann vom Untersuchenden umso besser gefällt werden, je größer sein Verständnis bez. der sich in Arbeit befindlichen Aufgabe (inkl. möglicher Lösungswege) ist. Anders ausgedrückt: Um feststellen zu können, ob sich zwei Strategievorschläge soweit voneinander unterscheiden, dass tatsächlich lieber von zwei unterschiedlichen Strategien gesprochen werden sollte, muss in der Regel ein gutes Verständnis des Problems und möglicher Lösungsansätze vorhanden sein. Dies ist bei Analysen von Sitzungen aus dem professionellen Umfeld aber nur selten der Fall. Um eine konkrete Situation besser zu verstehen, kann es trotzdem hilfreich sein, über derartige Abgrenzungen nachzudenken.

P1 zuletzt genannte zweiteilige Strategie („1. GUI ausprobieren; 2. Einchecken“) geeinigt hat.[92]

Das Beispiel verdeutlicht noch einmal, dass es bei der BKM in erster Linie darum geht, Begrifflichkeiten und vorläufige Strukturierungen zur Verfügung zu stellen, die das Nachdenken über die Phänomene bei der PP erleichtern. So können Überlegungen dazu, ob es sich bei einer bestimmten Äußerung um ein *amend_strategy* oder ein *challenge_strategy* handelt, den Ausgangspunkt für die Identifizierung weiterer Aspekte einer Episode bilden und somit dabei helfen, die Bedeutung der Episode für den Verlauf der gesamten Sitzung besser zu verstehen. Ein „stumpfes“ Kodieren mit der BKM wird hingegen selten hilfreich sein.

7. **Zustimmung zu Strategievorschlägen:** Die beobachten affirmativen Erwiderungen auf Strategievorschläge waren in der Regel einsilbig/kurz (z.B. „Ja“, „Hm‿Hm“). Nur selten wurde die Zustimmung in ganzen Sätzen formuliert (z.B. “O.K. Gut. (.) Dann‿machen‿wir‿das.“). Außerdem kamen nur bei einer als *agree_stategy* ins Auge gefassten Äußerung Zweifel darüber auf, ob es sich wirklich um eine hundertprozentige Zustimmung handelt (siehe Tabelle 7.21 und Abbildung 7.6).

8. **Ablehnung von Strategievorschlägen:** Im Rahmen der in der vorliegenden Arbeit erläuterten Untersuchungen konnte nur ein *disagree_strategy* beobachtet werden (und zwar in Sitzung PR2.1).[93] Es bezieht sich auf folgenden Vorschlag von *P2*:

   > „Wir könnten im Prinzip im ersten Schritt nur das `TableModel` ändern und das `FeatureProxy` so lassen wie es ist. Könnten wir machen.“

   Auf diesen Strategievorschlag vom Typ DPR[94] erwidert der Partner:

[92] Da *P1* der ursprünglich zweiteiligen Strategie (1. Weitere Methode einbauen; 2. Dialog ansehen) bereits im Vorfeld zugestimmt hatte, wurde entschieden, die fünfte Äußerung in Tabelle 7.18 („Äh, ungern einchecken, bevor ich nicht die GUI ausprobiert hab.“) mit *amend_strategy* und nicht mit *propose_strategy* (vom Typ EXS) oder gar *challenge_strategy* zu annotieren. Potentiell hätte natürlich auch analysiert werden können, welche Teilschritte nachfolgend wirklich durchgeführt wurden, um so entscheiden zu können, welche Strategie am Ende der Diskussion verabschiedet wurde. Diese Analyse würde aber nur dann zu einem eindeutigen Ergebnis führen, wenn später tatsächlich alle vier Schritte derart durchlaufen worden wären. Denn das Auslassen einzelner Schritte muss nicht zwangsläufig bedeuten, dass diese nicht geplant waren. Sie könnten später auch schlichtweg vergessen worden sein. Anmerkung: Genau genommen könnte man noch nicht einmal dann mit Sicherheit ein Aussage machen, wenn später in den vier Schritten vorgegangen wird (siehe hierzu auch die Diskussion über die Rolle der Einbeziehung des Kontextes in Abschnitt 9.3.3).

[93] Aufgrund der Tatsache, dass nur ein Phänomen vom Typ *disagree_hypothesis* beobachtet werden konnte, sind die Ausführungen zu diesem Konzept nur als rudimentär bzw. außerordentlich vorläufig zu betrachten.

[94] Die Regel, die aus dieser Äußerung eine Strategie macht, besteht darin, dass das `FeatureProxy` vorläufig nicht modifiziert werden soll.

„Also ich glaub' nicht, dass wir am `FeatureProxy` was ändern müssen."

Da es *P1* somit für wahrscheinlich hält, dass an der Klasse `FeatureProxy` keine Änderungen vorgenommen werden müssen, muss davon ausgegangen werden, dass er die Strategie als Ganzes ablehnt. Diese kann nämlich unter dieser Bedingung nicht mehr wirklich als eine solche bezeichnet werden.

Achtung: Das Beispiel demonstriert einen weiteren Fall, in dem es angebracht sein kann, schon im Rahmen einer BS-Konzeptualisierung eine Doppelkodierung vorzunehmen. Denn wenn man davon ausgeht, dass der primäre illukutionäre Akt des Sprechers in der Ablehnung besteht, muss *disagree_strategy* annotiert werden. In Hinblick darauf, dass es für die weiteren Untersuchungen wichtig sein könnte zu erfassen, dass die Ablehnung aber nur implizit auf Basis einer Vermutung erfolgt ist, sollte auch *propose_hypothesis* kodiert werden. Weitere Anmerkungen zum Beispiel werden im Zusammenhang mit Erörterungen zur Klasse *hypothesis* gemacht (siehe Tabelle 7.36).

9. **„Vorschlagsfreie" Fragen:** Genauso wie *ask_design*- (siehe S. 176) werden auch *ask_strategy*-Konzepte genau dann eingesetzt, wenn ein Entwickler eine Frage formuliert, mit der er keinen eigenen Vorschlag zur Diskussion stellt, sondern fordert bzw. wünscht, dass der Partner einen solchen äußern möge.

   Im Gegensatz zu *ask_design* kann aber bei *ask_strategy* (und auch bei *ask_step*) die charakteristische Eigenschaft des Nichtnennens eines eigenen Vorschlags zu Problemen bei der Kodierung führen. Denn oft ist es kaum möglich zu entscheiden, ob es dem Sprecher um strategisches oder taktisches Verhalten geht, ob also *ask_strategy* oder *ask_step* zu kodieren ist (siehe Beispiel 1 in Tabelle 7.20).[95]

   Auf S. 232 werden diese Schwierigkeiten bei der Abgrenzung zwischen *ask_strategy* und *ask_step* näher erläutert.

Abgrenzungen von *strategy*-Konzepten zu anderen Konzepten

Einige Kriterien, um zwischen *strategy*-Konzepten und Konzepten anderer Klassen differenzieren zu können, sind in Abbildung 7.7 in Übersicht dargestellt. Details und weitere Abgrenzungsfälle, insbesondere die zu Konzepten der Klasse *step*, werden nachfolgend erläutert.

- ****_strategy* vs. *explain_knowledge*/*explain_finding*:** Ähnlich wie *propose_step*-Äußerungen können auch Strategievorschläge in gewisser Weise als Wissen interpretiert werden, sind aber im Rahmen der BKM nicht

[95] Bisher konnten keine Äußerungen vom Typ *ask_strategy* beobachtet werden, in denen der Terminus „Strategie" verwendetet wurde.

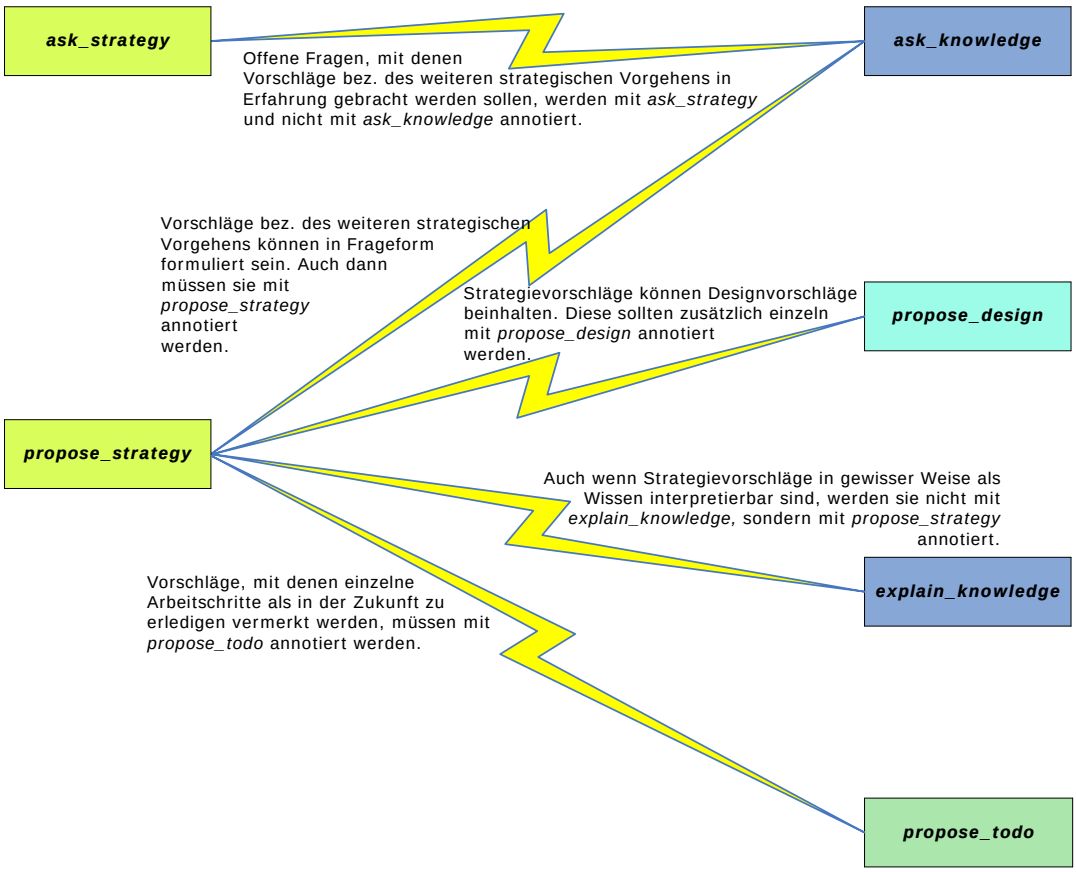


Abb. 7.7: Übersicht der wichtigsten Kriterien zur Differenzierung zwischen Elementen der Klasse *strategy* und solchen der Klassen *knowledge*, *design* und *todo*. Das Argument bez. der Differenzierung zw. *propose_strategy* und *explain_knowledge* lässt sich direkt auf die Abgrenzung zw. *propose_strategy* und *explain_finding* übertragen.

mit *explain_knowledge* (und auch nicht *explain_finding*), sondern mit *propose_strategy* zu annotieren. Hierbei sind etwaige Begründungen genauso wie die Begründungen zu andersartigen Vorschlägen auch einzeln und in der Regel mit *explain_knowledge* oder *explain_finding* zu konzeptualisieren.

Dasselbe gilt für Phänomene vom Typ *agree_strategy*, *challenge_strategy*, *amend_strategy*, *disagree_strategy* und *decide_strategy*.[96]

– ***propose_strategy* vs. *propose_todo*:** Während Äußerungen vom Typ *propose_todo* nur das Ziel haben zu vermerken, dass bestimmte einzelne Arbeitsschritte in einer im Allgemeinen nicht näher spezifizierten Zukunft erledigt werden müssen, betten Strategievorschläge vom Typ OWP und EXS zumindest zwei Arbeitsschritte in eine feste Abfolge ein. In der Regel

[96] Inwieweit Phänomene vom Typ *agree_strategy* und *disagree_strategy* wirklich als Transfer von Wissen interpretiert werden können bzw. sollten, spielt im Rahmen der Verwendung der BKM keine entscheidende Rolle und soll hier nicht weiter diskutiert werden.

beinhaltet ein Strategievorschlag auch, dass mit seiner Umsetzung unmittelbar begonnen werden soll.

– ***propose_strategy* vs. *propose_step***: Wie bereits geschildert ist das wesentliche Merkmal von Strategievorschlägen das *sichtbare Identifizieren und Inbeziehungsetzen von bestimmten „Dingen“* derart, dass eine Vorgehensvorschrift entsteht. So werden bei Strategievorschlägen vom Typ OWP und EXS mehrere (in der Regel nicht-atomare) Schritte, so genannte Teilaufgaben, identifiziert und in eine Reihenfolge gebracht, während bei Strategien vom Typ DPR Regeln sowie der Bereich, auf den diese anzuwenden sind[97], festgelegt werden.

  Der Prozess des Identifizierens und Inbeziehungssetzens geschieht dabei meist in einem *schöpferischen Akt*, bei dem die momentane Situation besondere Berücksichtigung findet. Der Sprecher kombiniert in der Regel unterschiedliches Wissen (auch Annahmen) bzw. verschiedene zuvor erlangte Erkenntnisse, um zu einer *neuen*, in seinen Augen gewinnbringenden Vorgehensidee zu kommen.

  Der Gewinn, z.B. einen bestimmten Zwischenstand überhaupt oder schneller erreichen zu können, ist das, was als *übergeordnetes Ziel* des Vorschlags bezeichnet werden kann. Das Adjektiv „übergeordnet“ wird deshalb verwendet, weil der Gewinn in der Regel nicht an einem einzelnen Teil des Strategievorschlags allein festgemacht werden kann.

  In Äußerungen vom Typ *propose_step* wird hingegen nur ein einzelner Arbeitsschritt identifiziert bzw. postuliert. Dieser wird nicht mit anderen in Beziehung gesetzt. Der Arbeitsschritt wird als monolithisch und unabhängig dargestellt. Dementsprechend ist so etwas wie ein übergeordnetes Ziel nicht vorhanden bzw. zumindest nicht erkennbar. Eine schöpferische Leistung, speziell eine solche, bei der etwas Neues, für sich allein Stehendes geschaffen wird, ist in der Regel nicht von Nöten. Dies schließt allerdings nicht aus, dass der Sprecher eine kognitive Leistung, z.B. in Form einer Schlussfolgerung, erbringen muss.

  Auf den ersten Blick scheinen die Konzepte *propose_strategy* und *propose_step* also hinreichend genau spezifiziert zu sein, um entsprechende Phänomene identifizieren bzw. gegeneinander abgrenzen zu können. Detaillierte Analysen einzelner Beobachtungen und weitergehende theoretische Überlegungen zeigen allerdings, dass eine Reihe von Abgrenzungsproblemen, zumindest potentiell verblieben sind. Für den Kodier- und Erkenntnisprozess sind vor allem die folgenden interessant:

  1. **Strategien ohne eigenen schöpferischen Akt**: Auch wenn es bisher nicht beobachtet werden konnte ist es plausibel, dass Entwickler

[97] Allerdings wird der Bereich, auf den die Regel angewendet werden soll, oft nur sehr unpräzise beschrieben.

„strategieartige“ Vorschläge äußern – also solche, die zumindest implizit Teile identifizieren und in Beziehung setzen – die mit Sicherheit nicht das Ergebnis eines gerade beendeten schöpferischen Prozesses sind, sondern „alte“ bzw. „recycelte“ Ideen darstellen, evtl. sogar solche dritter Personen – z.B. den „Test-First“-Ansatz aus XP [14, S. 50]. Die BKM sieht vor, dass auch solche Äußerungen mit *propose_strategy* zu kodieren sind.[98]

2. **Arbeitsschritte, die Bezug nehmen**: Manche „step-artige“ Äußerungen, also solche, die nur einen einzelnen Arbeitsschritt explizieren, scheinen mit der Intention gemacht zu werden, hervorzuheben, dass sich das unmittelbare Durchführen des vorgeschlagenen Schrittes günstig auf die Bewältigung der nachfolgenden Arbeiten auswirken wird (siehe Beispiel 2 in Tabelle 7.10 und Tabelle 7.40). Solche Äußerungen sollten, falls sie nur unspezifisch und implizit (z.B. durch die Verwendung der Phrase „erst mal“) auf nachfolgende Schritte verweisen, mit *propose_step* annotiert werden. Im Rahmen der BKM werden sie als *Arbeitsschritte mit strategischem Charakter* bezeichnet.

   Achtung: Es ist keinesfalls so, dass Äußerungen, die Phrasen wie „erst einmal“ verwenden und darüber hinaus nur einen einzigen Schritt explizit nennen, immer als Arbeitsschritt mit strategischem Charakter und nicht als Strategie konzeptionalisiert werden sollten. Oft handelt es sich bei solchen Aussagen auch um Regeln, also um Strategien vom Typ DPR. Ein Beispiel hierfür ist die folgende Äußerung aus Sitzung PR1.1:

   > „Wollen wir erst mal die einfachen Fälle nehmen?“

   Der Sprecher bezieht sich auf eine Reihe von Änderungen, die an unterschiedlichen Stellen des Kodes, genau genommen in unterschiedlichen Funktionen, vorgenommen werden müssen. Da dem Paar noch nicht für alle Anwendungsfälle klar ist, an welche Stellen bzw. in welchen Funktionen die Änderungen am besten vorgenommen werden sollten, schlägt der Sprecher vor, erst einmal die einfachen, sprich zweifelsfreien Änderungen vorzunehmen. Er stellt also eine Regel auf, welche die Abarbeitung in zwei bzw., wenn man das Lösen der offenen Fragen als eigenständige Aufgabe sieht, eigentlich sogar drei bez. ihrer Ziele wohldefinierte und eben nicht unspezifische Schritte zerlegt.

3. **Arbeitsschritte mit übergeordnetem Ziel**: Nicht nur Strategien, sondern auch die gerade erläuterten Arbeitsschritte mit strategischem Charakter können derart formuliert sein, dass in gewisser Weise ein übergeordnetes Ziel zu erkennen ist (auch wenn dieses, genauso wie bei Strategieäußerungen, oft nicht explizit ausgeführt ist).

[98] Noch einmal zu Erinnerung: In weiterführenden Untersuchungen muss entschieden werden, ob es sinnvoll ist, derartige Regeln aufrecht zu erhalten (siehe auch Abschnitt 9.3).

Beispielsweise ist bei der zweiten Äußerung in Tabelle 7.10 („[...] zeig' ich Dir erst mal, was ich gemacht habe“) zu vermuten, dass es dem Sprecher auch darum geht hervorzuheben, dass die Durchführung seines Vorschlags dazu führen wird, dass sich die nachfolgende Zusammenarbeit verbessert, da dafür gesorgt wird, dass beide Personen dasselbe Wissen bez. der bereits existierenden Kodebasis besitzen.

4. **Mehrteilige Vorgehensvorschläge, die keine Strategie bilden**: Betrachtet man eine Äußerung wie „Lass uns das Programm starten, damit wir testen können“[99], so scheint es auf den ersten Blick naheliegend, diese als Strategie zu klassifizieren. Schließlich werden zwei Arbeitsschritte identifiziert und in Beziehung gesetzt. Dem Inbeziehungsetzen liegt allerdings kein schöpferischer Akt zu Grunde. Schließlich sollte jedem Entwickler bekannt sein, dass ein Programm immer zuerst gestartet werden muss, damit man es testen kann. Außerdem ist kein übergeordnetes Ziel zu erkennen. Vielmehr geht es allem Anschein nach nur darum, dass das Programm getestet wird, wobei der erste Schritt, der zumal als atomar interpretiert werden kann, genau genommen einen Teil des zweiten bildet. Es handelt sich also bei der Äußerung nicht um etwas, was im Rahmen BKM als Strategievorschlag bezeichnet werden sollte.[100] Nicht jede Nennung von mehreren Schritten kann also automatisch als Strategie interpretiert werden.
5. **Der schöpferische Prozess kann nicht beobachtet werden**: Der wie oben beschriebene schöpferische Prozess ist im Allgemeinen nicht sichtbar. Außerdem kann in vielen Fällen kaum objektiv festgestellt werden, ob eine Person eine neue Strategie ersonnen hat oder auf eine bekannte zurückgreift.
6. **Schöpferische Leistungen auf niedrigem Niveau**: Die schöpferische Leistung kann sich auf einem sehr niedrigen Niveau bewegen, d.h. nur wenige bzw. sehr einfache Ideen beinhalten, oder entscheidend durch Randbedingungen und eben nicht durch den Einfallsreichtum des Entwicklers determiniert sein (siehe z.B. die erste Äußerung in Tabelle 7.21).

– ***propose_strategy* vs. *propose_design*:** In Abschnitt 7.5.1 wurde auf S. 201 erläutert, dass Designvorschläge mit Handlungsvorschlägen einher gehen können. Ähnliches kann auch bei Äußerungen vom Typ *propose_strategy* vorkommen, speziell bei solchen vom Typ OWP. Denn einzelne Teilschritte einer Strategie können als Designvorschläge formuliert sein (siehe Beispiel auf S. 219).

[99] Das Beispiel ist fiktiv. Eine solche Äußerung konnte in den untersuchten Sitzungen nicht beobachtet werden.

[100] Detaillierte Aussagen sind letztendlich nur unter Berücksichtigung des Kontextes der Äußerung sinnstiftend. Dieser ist aber, da es sich um ein fiktives Phänomen handelt, nicht bekannt. Sicher ist zumindest, dass einmal das Konzept *propose_step* vergeben werden sollte.

In solchen Fällen ist es angebracht, nicht nur den gesamten Vorschlag mit *propose_strategy* zu annotieren, sondern zusätzlich die entsprechenden Teilphrasen mit *propose_design* (oder ggf. auch *amend_design*). Dies kann dazu führen, dass, falls das Paar nachfolgend über den Strategievorschlag diskutiert, weitere Doppelkodierungen notwendig werden. Potentiell kann es sogar dazu kommen, dass die Strategiediskussion in eine Designdiskussion übergeht, also im weiteren Verlauf keine *strategy*-Konzepte mehr vergeben werden dürfen. Solche Episoden konnten aber in den vorliegenden Daten nicht beobachtet werden.

- ***propose_strategy* vs. *ask_knowledge*:** So wie schon im Zusammenhang mit *propose_design* (siehe S. 177) oder *propose_step* (siehe S. 202) erläutert, definiert die BKM auch für die Klasse *strategy* eine klare Regel in Hinsicht auf die Abgrenzung zur Klasse *knowledge*: Alle Äußerungen, die Nennungen von Strategien beinhalten, werden mit Konzepten der Klasse *strategy* annotiert, auch wenn sie als Frage formuliert sein sollten. Der Begriff „vorschlagen“ (*propose*) wird also für vielfältige Arten und Ziele der Formulierung verwendet, z.B. für das Einholen einer weiteren Meinung oder das Suchen nach Orientierung (siehe Klassifizierung in Tabelle 7.5).

- ***agree_strategy* vs. *agree_knowledge*:** Manche Strategievorschläge werden von den Sprechern zumindest in Teilen begründet. Wie bereits an anderer Stelle, beispielsweise im Zusammenhang mit *propose_step* diskutiert (siehe S. 192), sollten solche Begründungen einzeln mit *explain_knowledge* (evtl. auch mit *explain_finding*) konzeptualisiert werden. Aus dieser Aufteilung kann sich allerdings ein Problem ergeben: Denn stimmt der Partner nachfolgend zu (z.B. durch ein „O.K.“), ist nicht immer eindeutig identifizierbar, ob er sich auf den Strategievorschlag selbst oder den Inhalt der Begründung, die ja evtl. nur einen Teil der Strategie adressiert hat, bezieht, ob es sich also um ein *agree_strategy* oder ein *agree_knowledge* handelt.[101]

  Ein Beispiel hierfür ist in Tabelle 7.22 dargestellt. In dem dort wiedergegebenen Fall scheint es angemessen, sowohl *agree_strategy* wie auch *agree_knowledge* zu annotieren. Ob eine Doppelkodierung angebracht ist, hängt aber nicht nur von der Eindeutigkeit der Situation, sondern auch davon ab, welches Ziel mit der Analyse verfolgt wird, d.h. unter welchem Blickwinkel diese stattfindet und wie die betrachtete Episode diesbezügliche Erkenntnisse liefern soll.

- ***ask_strategy* vs. *ask_knowledge*:** Im Kern gilt hier dasselbe wie bei *ask_step* vs. *ask_knowledge* (siehe S. 206).

- ***ask_strategy* vs. *ask_step*:** Auf S. 227 wurde bereits angesprochen, dass nicht immer direkt, d.h. anhand einer Äußerung selbst, entschieden

[101] Potentiell betrifft dies natürlich alle affirmativen Äußerungen, in denen der Bezug nicht explizit artikuliert wird.

HHI-Kode	Äußerung	Kontext/Kommentar
P2.propose_strategy	„Dann sollten wir das zuerst machen oder danach. Aber nicht gleichzeitig.“	*P1* äußerte sich bez. möglicher Restrukturierungsstrategien (für Details siehe S. 221).
P2.propose_hypothesis[a] + *P2.explain_knowledge*[b]	„Ich fürcht' halt, dass bei dem Arbeitspunkt, dass wir sowieso die Zeit überschreiten werden, (.) weil das relativ komplex wird.“	Nach ca. vier Sekunden Stille macht *P2* eine Hypothese bezüglich des weiteren Prozessverlaufs und erklärt, wie er zu dieser gekommen ist.
P2.decide_strategy	„Vielleicht, dass man's danach macht?“	Nach vier Sekunden Stille entscheidet *P2* selbst, welche der beiden von ihm vorgeschlagenen Strategiealternativen durchgeführt werden sollte. Er äußert diese Entscheidung allerdings in Frageform (Vorschlag vom Typ 3 (siehe Tabelle 7.5)).
P2.explain_standard of knowledge[c]	„Ich bin mir da aber auch nicht sicher.“	Weitere zwei Sekunden später betont er, dass er nicht genug Wissen besitzt, um eine zumindest aus seiner Sicht als richtig/sinnvoll zu erachtende Entscheidung zu treffen.
P1.challenge_strategy	„Ich würd' das (~zuerst machen).“	*P1* widerspricht der Auswahl von *P2* und wählt die Alternativstrategie.
P2.agree_strategy	„Hm, ja, ich (!...!)“	Obwohl *P2* sich gerade noch anders geäußert hatte, stimmt er dem Gegenvorschlag von *P1* zu.
P1.explain_knowledge	„Denn wenn wir es danach machen wollen, denn, dann ähh (.), dann passiert das nicht“	*P1* begründet seine Strategieentscheidung mit Erfahrungen, die er gemacht hat.[d]
P2.agree_knowledge/ *P2.agree_strategy*	„O.K.?“	Es kann nicht entschieden werden, ob sich die Zustimmung von *P2* auf das gerade geäußerte Erfahrungswissen von *P1* bezieht oder nur eine Wiederholung der Zustimmung zum Strategievorschlag ist. Die Zustimmung ist derart intoniert, dass verbleibende Zweifel zu erkennen sind.

[a] Details zum Konzept *propose_hypothesis* werden ab S. 283 erläutert.
[b] Evtl. handelt es sich auch um eine Erkenntnisäußerung vom Typ T (siehe S. 254).
[c] Details zum Konzept *explain_standard of knowledge* werden ab S. 297 erläutert.
[d] Siehe S. 341.

Tab. 7.22: Bei einer Zustimmung kann nicht immer zweifelsfrei entschieden werden, auf welche vorhergegangene Äußerung sich diese bezieht, und somit auch nicht, mit welchem Konzept sie annotiert werden sollte. Beispielsweise endet die hier dargestellte Strategieepisode (PR2.1: 12:58:29–12:58:52) mit einer affirmativen Äußerung, die sich potentiell sowohl auf das unmittelbar zuvor übermittelte Wissen wie auch auf einen davor gemachten Strategievorschlag beziehen kann. Da in diesem Fall das übermittelte Wissen in direktem Zusammenhang mit der geäußerten Strategie steht, wurde sowohl *agree_knowledge* wie auch *agree_strategy* annotiert. Hinweis: *P2* sitzt zurückgelehnt, *P1* zum Display gebeugt da. Gelegentlich wendet sich *P1* *P2* zu.

werden kann, ob es sich um ein *ask_strategy* oder um ein *ask_step* handelt. Somit muss wie bei vielen anderen Zweifelsfällen auch der Kontext der Äußerung analysiert werden.

Was bisher noch nicht erläutert wurde, aber speziell im hier betrachteten Fall relevant wird, ist die Tatsache, dass eine solche Untersuchung zu Fehlinterpretationen führen kann. Denn bei Kontextanalysen fokussiert sich das Augenmerk des Untersuchenden oft automatisch auf die Reaktion des Partners. So wird im vorliegenden Fall in der Regel überprüft, ob dieser mit einem strategischen oder mit einem taktischen Vorschlag antwortet. Dabei wird leicht übersehen, dass natürlich auch der Partner die Intention der Frage falsch verstehen kann oder sogar verstehen will. Es ist also keineswegs für alle Untersuchungen, d.h. für jeden Blickwinkel zweckmäßig, die Antwort des Partners bei der Kodierung der ursprünglichen Frage als Indiz für das eine oder das andere Konzept zu verwenden.

Weitere Anmerkungen zu *strategy*-Konzepten

Der Begriff Strategie wird heutzutage in vielerlei Zusammenhängen verwendet, z.B. in der Spieltheorie (Teilgebiet der Mathematik), der Wirtschaft (z.B. beim strategischen Management) oder der Softwareentwicklung/Programmierung (z.B. beim Entwurfsmuster Strategie [70]). Seiner Wortbedeutung nach stammt er aber aus dem Militärischen und bezeichnet die Kriegskunst. Diesbezüglich wird er von Carl von Clausewitz im Rahmen der Abhandlung „Vom Kriege“ [47] näher erläutert und gegen den Begriff Taktik abgegrenzt. Obwohl knapp 200 Jahre alt und in einer gänzlich anderen Domain verankert, können Clausewitz' Ausführungen dabei helfen, die in der vorliegenden Arbeit dargestellten Begriffsbildungen besser zu verstehen bzw. die theoretische Sensibilität (siehe S. 73) in Bezug auf Strategie- und Taktikphänomene zu verbessern. Clausewitz' Überlegungen werden deshalb im Exkurs 9 auf S. 235 vorgestellt. Auf Verwendungen der Begriffe Strategie und Taktik in qualitativen oder qualitativ-quantitativen Untersuchungen kognitiver Phänomene bei der Softwareentwicklung/PP wird hingegen erst in Abschnitt 11.3 eingegangen.

Abschließend werden in Abbildung 7.8 noch einmal die Zusammenhänge zwischen den zentralen Prozesssteuerungsbegriffen (Tätigkeit, Arbeitsschritt, Teilaufgabe sowie Strategie) dargestellt. Achtung: In der Abbildung geht es nur darum, potentiell hilfreiche Begrifflichkeiten zu erläutern. Es handelt sich in keiner Weise um eine erste Version eines Modells der Prozesssteuerungsmechanismen bei der PP.

Exkurs 9: *Analogien zum Strategiebegriff von Carl von Clausewitz*
In seinem 1832–1834 veröffentlichten und heute noch als relevant eingeschätzten Werk „Vom Kriege“ bezeichnet Carl von Clausewitz mit „*Taktik* die Lehre vom Gebrauch der Streitkräfte im Gefecht“ und mit „*Strategie* die Lehre vom Gebrauch der Gefechte zum Zweck des Krieges“ [47, S. 113f]. Bei einer Taktik geht es nach Clausewitz darum, „Gefechte in sich anzuordnen“, während eine Strategie darauf zielt, Gefechte „zum Zweck des Krieges zu verbinden“.
Ausgangspunkt dieser Einteilung ist die Erkenntnis, dass der Kampf als wesentlicher Teil der Kriegsführung kein einzelner Akt ist, sondern aus einer „mehr oder weniger großen Zahl einzelner, in sich geschlossener Akte“, den so genannten Gefechten besteht. Der Begriff Gefecht markiert also für Clausewitz diejenige (gedankliche) Einheit, unter deren Zuhilfenahme eine Grenze zwischen Taktik und Strategie gezogen werden kann. Dementsprechend versucht er den Gefechtsbegriff bestmöglich zu definieren, stellt aber auch heraus, dass Zweifelsfälle vorkommen können, z.B. solche, „wo mehrere Gefechte auch [...] als ein einziges betrachtet werden können“. Er betont, dass dies dem „Einteilungsgrund nicht zum Vorwurf gereichen“ darf, da er diese Eigenschaft „mit allen Einteilungsgründen wirklicher Dinge gemein [hat], deren Verschiedenheit immer durch abgestufte Übergänge vermittelt wird. Es kann also [...] einzelne Tätigkeitsakte geben, die eben so gut, und zwar ohne Veränderung des Gesichtspunktes, zur Strategie als zur Taktik zu zählen sind [sic]“.
Die Begriffsbildungen von Clausewitz lassen sich, wenn man davon absieht, dass sie als „Lehre von etwas“ definiert sind, wie folgt in Beziehung zu denen der BKM setzen: Wie der Kampf ist auch der Lösungsprozess bei der PP kein einzelner Akt, sondern besteht in der Regel aus einer Reihe von Teilen, die man aber nicht als Gefechte, sondern als Bearbeitungen einzelner Teilaufgaben (nicht-atomarer Arbeitsschritte) bezeichnen würde. Innerhalb dieser wird versucht, gewisse mehr oder weniger genau spezifizierte bzw. erkennbare Zwischenziele zu erreichen. Äußerungen vom Typ *step* dienen dazu, aktuell anstehende „Bearbeitungsgefechte“ auszutragen, indem sie Teile einer im allgemeinen erst im Entstehen befindlichen Taktik zur Bewältigung der Teilaufgabe explizieren. In Äußerungen vom Typ *strategy* wird hingegen in der Regel über den Einsatz von Teilaufgaben zum Zwecke der Lösung der Gesamtaufgabe (bei Clausewitz der Kampf) gesprochen.[a] Dies kann beispielsweise durch deren Identifizierung und Anordnung geschehen. Für eine konsistente Identifizierung von Strategien ist es also wichtig, dass eine möglichst präzise Charakterisierung der Begriffe Arbeitsschritt, Teilaufgabe, Beziehung und auch Ziel existiert (siehe Abbildung 7.8). Die Arbeitsschritte mit strategischem Charakter (siehe S. 230) geben einen Eindruck davon, wo Schwierigkeiten mit diesen Begriffsbildungen beginnen können. Diesen muss in spezialisierten Untersuchungen mittels einer Ausdifferenzierung der Konzepte begegnet werden.

[a] Wie bereits erläutert muss es nicht immer um die Lösung der Gesamtaufgabe gehen. Es kann auch nur ein Teilaspekt dieser adressiert werden.

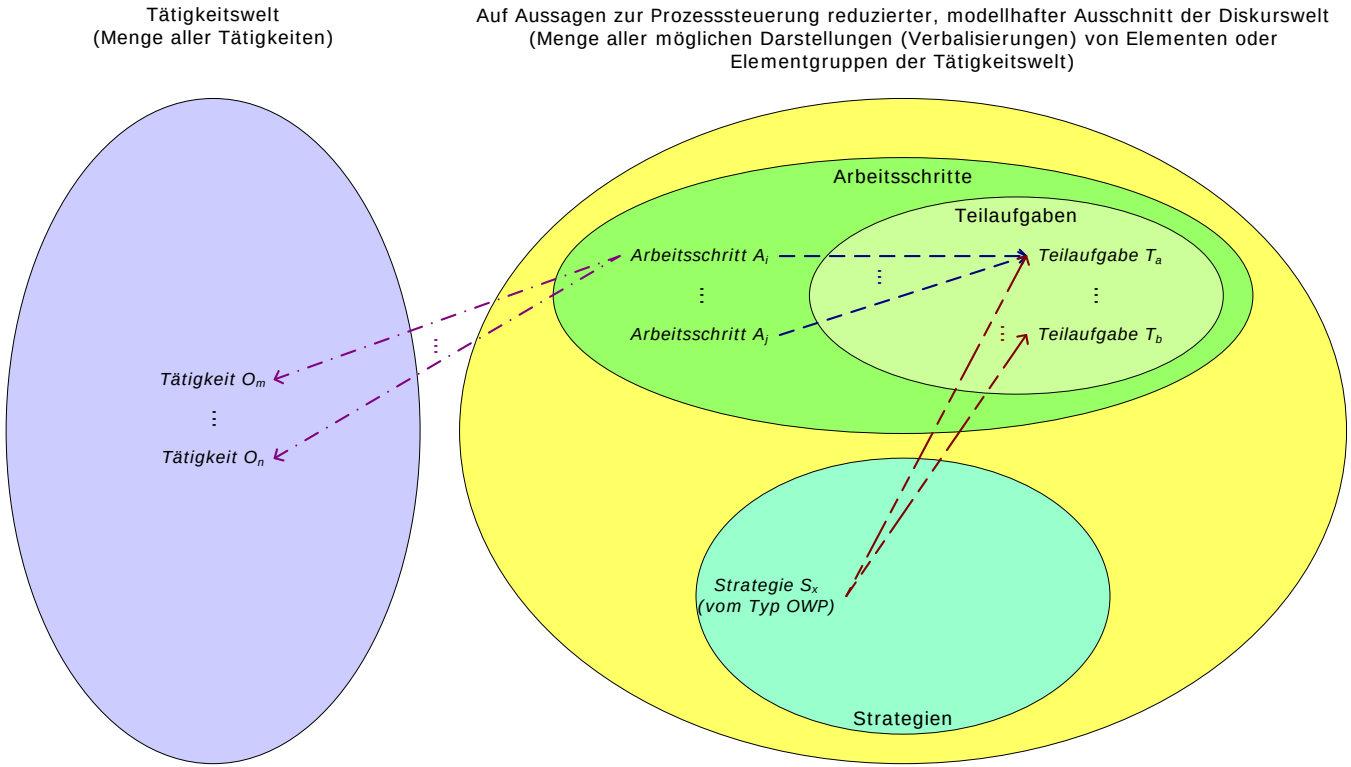


Abb. 7.8: Darstellung der Zusammenhänge zwischen den wichtigsten Begrifflichkeiten, die bei der Konzeptionalisierung von Äußerungen zur Prozesssteuerung eine Rolle spielen. Hierbei wird vom Zweck der Äußerungen (z.B. Vorschlagen oder Detaillieren) abstrahiert: In Strategien (hier z.B. S_x), genau genommen Strategieäußerungen (also Phänomenen vom Typ *strategy*) – zumindest in solchen vom Typ OWP und EXS – werden mehrere Teilaufgaben (in der Regel nicht-atomare Arbeitsschritte, hier z.B. T_a bis T_b) in Beziehung gesetzt (auch wenn diese nicht immer explizit genannt werden). Die Abarbeitung von Teilaufgaben (z.B. T_a) kann durch eine Reihe atomarer oder auch nicht-atomarer, ad hoc geäußerter Arbeitsschritte (*steps* wie z.B. A_i bis A_j) gesteuert werden. Jeder geäußerte Arbeitsschritt (z.B. A_i) kann zu einer Reihe von Tätigkeiten (hier z.B. O_m bis O_n) führen oder auf eine solche verweisen. Tätigkeiten sind im Sinne der BKM Objekte, die unabhängig von einer Verbalisierung existieren. Sie erfahren bei einer Kodierung mit HHI-Konzepten keine explizite Berücksichtigung. Ihre Adressierung im Rahmen einer Verbalisierung transformiert bzw. verpackt sie in Objekte der Diskurswelt, also in etwas, was je nach Kontext als teilbare oder als nicht-teilbare Einheit gesehen bzw. behandelt wird.

7.5.5 *state*-Konzepte

Fokus der *state*-Konzepte

Konzepte der Klasse *state* dienen der Annotation von Äußerungen, in denen der Abarbeitungsstand bzw. Fortschritt einer Strategie oder Folge von Teilaufgaben thematisiert wird. Äußerungen vom Typ *state* verhalten sich also zu Äußerungen vom Type *strategy* wie Äußerungen vom Typ *completion* zu solchen vom Typ *step* (siehe Seite 207). Im Verlauf der Analysen wurde analog der Regel zur Klasse *completion* festgelegt, dass die Folge von Handlungen, auf die sich eine Äußerung vom Typ *state* bezieht, im Vorfeld nicht explizit verbalisiert bzw. festgelegt worden sein muss. Dies geschah aus folgenden Gründen:

1. Es ist möglich, dass ein einzelner Entwickler einer Strategie folgt, ohne diese dem Partner mitgeteilt zu haben; sei es, weil er davon ausgeht, dass diesem ohnehin klar ist, welches Vorgehen gerade angebracht ist, oder weil er einer Strategiediskussion aus dem Weg gehen möchte und damit rechnet, sein Vorhaben, auch ohne es abgesprochen zu haben, umsetzen zu können.[102]

2. Bei der Rekapitulation eines Sitzungsverlaufs können bestimmte Abfolgen von Handlungen als Strategie erscheinen, auch wenn ursprünglich keine solche erdacht bzw. formuliert wurde.

Konzepte der Klasse *state* adressieren also nicht nur Äußerungen, in denen der Erfolg von explizit formulierten Strategien bewertet wird, sondern alle Aussagen zum Prozessfortschritt, die sich nicht auf einem taktischen Niveau (siehe Klasse *completion*) bewegen.

Beispiele für *state*-Äußerungen sind in Tabelle 7.23 dargestellt.

Identifizierte *state*-Konzepte und ihre speziellen Eigenschaften

Bisher konnten drei unterschiedliche Typen von *state*-Phänomenen beobachtet werden: *explain_state*, *agree_state* und *challenge_state*. Sie weisen eine Reihe von speziellen Eigenschaften auf:

1. **Fehlender Bezug zu einem konkreten Strategievorschlag:** Wie schon angedeutet, adressiert das Konzept *explain_state* nicht nur Äußerungen, in denen explizit auf eine bestimmte Strategiefestlegung Bezug genommen wird, sondern auch solche, die (mehr oder weniger) ad hoc definierte

[102] Die Diskussion der sei es absichtlich oder auch unabsichtlich „verschwiegenen“ Strategien verdeutlicht die Grenzen des behavioristischen und zugleich im Wesentlichen auf die in den Sitzungen gemachten Äußerungen beschränkten Untersuchungsansatzes. Viele Fragestellungen lassen sich auf diese Weise nicht analysieren. Oft wird es unumgänglich sein, auch andere Daten heranzuziehen. Beispielsweise kann es im Zusammenhang mit Untersuchungen zur strategischen Planung notwendig werden zu überlegen, auf welche Weise nicht explizit formulierte Strategien aufgedeckt werden können – z.B. durch Befragungen, die den Sitzungen nachfolgen. Siehe hierzu Abschnitt 9.3.

#	Session: HHI-Kode	Äußerung	Kontext/Kommentar
1	*ST1.1: P1.explain_state* (+ *P2.disagree_step*)	„Ó.k. (.) Jetzt haben wir (!...!) Das haben wir erledigt. Das haben (!!...!!) (.) Doch,‿doch. Ne,‿ne. Wir haben den `Topic`[a] umgestellt. Das haben wir vorhin gemacht. Jetzt haben wir den `Spy` erledigt. Das hier funktioniert immer noch. Da hat sich nichts dran geändert.“	Nachdem die zuletzt durchgeführten Änderungen erfolgreich getestet worden sind, greift sich der Driver *P2* das Anforderungsdokument und überprüft, welche Anforderungen bisher umgesetzt wurden. Dabei deutet er auf bestimmte Stellen im Dokument. Auch der Observer sieht in das Dokument. Er unterbricht den Driver kurz durch ein *challenge_state* (siehe nächstes Beispiel). Dieser lässt sich aber nur sehr kurz beirren, widerspricht und führt seine Äußerung fort, bevor *P2* mit dem zweiten Satz seines Einwurfs beginnt. Kurzzeitig sprechen beide parallel.
2	*ST1.1: P2.challenge_state* (+ *P2.propose_step*[b])	„Ne‿ne, wartet, nicht ganz. Wir müssen diss da noch übertragen.“	*P2* fällt dem Driver ins Wort (siehe Beispiel 1). Dabei zeigt er auf eine bestimmte Stelle im Dokument.
3	*PR1.1: P1.explain_state*	„Ja,‿jetzt‿sind wir (~strenggenommen) fertig, n[e]?“	Nach einer Reihe von – für das Erreichen des Sitzungszieles – zentralen und miteinander zusammenhängenden Kodeänderungen in verschiedenen Dateien, Diskussionen über deren Qualität und potentiell notwendige Refaktorisierungen hält der Driver fest, dass man nun so gut wie fertig ist.[c] Die einzelnen absolvierten Schritte wie Diskussionen, Analysen des bestehenden Kodes und Implementierungen waren im Vorfeld nicht als Strategie formuliert und verabschiedet worden. Vielmehr ergaben sie sich mehr oder weniger spontan und ungeordnet auseinander.
4	*PR2.1: P1.explain_state*	„Also, wir sind halt vom Einchecken usw. meiner Meinung nach an ’nem Punkt, wo wir ’n Break machen können.“	Siehe Tabelle 7.21 bzw. Abbildung 7.6.
5	*PR2.1: P2.agree_state*	„Das stimmt natürlich. (.) Das stimmt.“	Antwort von *P2* auf die in Beispiel 4 wiedergegebene Einschätzung.

[a] `javax.jms.Topic` (siehe Tabelle 7.4)

[b] Die Intention des Sprechers konnte nicht mit hundertprozentiger Sicherheit ermittelt werden. Evtl. handelt es sich nicht um ein *propose_step*, sondern um ein *propose_todo*. Probleme dieser Art werden in Abschnitt 9.3.1 behandelt.

[c] Diese Aktivitäten nahmen über 30 Minuten in Anspruch. In Tabelle 7.15 wird ein kurzer Ausschnitt aus dieser Episode erläutert.

Tab. 7.23: Beispiele für Äußerungen, die mit *state*-Konzepten annotiert wurden.

Prozessabschnitte referenzieren (siehe beispielsweise Äußerung 3 in Tabelle 7.23). Genau genommen konnte bisher sogar kein einziges Phänomen beobachtet werden, in dem wirklich explizit auf ein bestimmtes *propose_strategy* verwiesen wurde.

2. **Teilweiser Widerspruch:** Aus Abbildung 7.3 geht hervor, dass das Konzept *challenge_state* Äußerungen adressiert, in denen einer Fortschrittsbewertung widersprochen wird, indem der Sprecher eine alternative Beurteilung abgibt. Hierbei ist *challenge_state* derart definiert, dass es ausreicht, wenn der Sprecher bez. eines Teilschritts eine andere Meinung vertritt.

   Solche Meinungsbekundungen können mit Vorschlägen, z.B. vom Typ *propose_step* oder *propose_todo* einhergehen. Diese sollten ggf. zusätzlich annotiert werden (siehe Beispiel 2 in Tabelle 7.23).

   Achtung: Da bisher nur eine Äußerung vom Typ *challenge_state* beobachtet werden konnte, ist es nicht möglich, detailliertere Aussagen über die Phänomene dieses Typs zu machen.

3. **Knapp formulierte Zustimmungen:** Genauso wie die *agree*-Äußerungen anderer Klassen scheinen auch die Zustimmungen zu Statusbeschreibungen in der Regel eher kurz zu sein (z.B. „Ja“ (PR1.1)). An dieser Stelle muss allerdings festgehalten werden, dass bisher nur drei Phänomene des Typs (*agree_state*) beobachtet werden konnten.

Abgrenzungen von *state*-Konzepten zu anderen Konzepten

Die wichtigste Unterscheidung ist die zwischen *explain_state* und *explain_completion*. Sie ist im Wesentlichen durch die Differenzierung zwischen *strategy* und *step* definiert.

Darüber hinaus muss ein besonderes Augenmerk auf die Unterscheidung der Konzepte *explain_state* und *explain_finding* gerichtet werden. Dies passiert in Abschnitt 7.6.1 auf S. 283. In wie weit bzw. wann es möglich und sinnvoll ist, zwischen dem Feststellen des Abarbeitungsstands einer Strategie (also einem *explain_state*) und einer mit geäußerten Begründung einer solchen Feststellung (einem *explain_knowledge* oder *explain_finding*) zu unterscheiden, wird weiterführenden Untersuchungen überlassen.

7.6 Universelle Konzepte

Während sich die bisher beschriebenen, im Kontext von Vorschlägen oder Fortschrittsbewertungen angesiedelten Konzepte bzw. Konzeptklassen immer auf genau einen bestimmten Typ von „Äußerungsinhalten“, z.B. nur auf Strategien oder nur auf das Design bezogen haben, ist dies bei den universellen Konzepten (UK) nicht der Fall. Bei ihrer Verwendung ist es unerheblich, ob mit einer Äußerung ein Aspekt des Produktes oder des Prozesses referenziert wird. Stattdessen

wird mittels der universellen Konzeptklassen (mit Ausnahme von *activity*[103]), d.h. denjenigen Teilmengen, in die die Menge der universellen Konzepte bez. ihrer Objekte zerfällt, zwischen Äußerungen, die sich auf unterschiedliche „Formen" von Wissen beziehen, differenziert – allerdings nur dann, wenn die Äußerungen nicht durch produkt- oder prozessbezogene Konzepte adressierbar sind. So wird im Rahmen der UK so weit dies möglich ist bzw. angebracht erscheint zwischen Äußerungen, mit denen bestimmte Formen von Bestandswissen und solchen, mit denen bestimmte gerade gewonnene Einsichten verbalisiert werden, unterschieden (siehe Abbildung 7.1). Der in der vorliegenden Arbeit verwendete Wissensbegriff umfasst, wie bereits in Abschnitt 7.1.1 erläutert, nur verbalisierbare bzw. explizierbare Inhalte (wobei Kommunikation unter Zuhilfenahme von Schrift und Bild an dieser Stelle auch zulässig sei), adressiert in diesem Rahmen aber mehr als das, was gemeinhin unter Wissen verstanden wird (vergleiche z.B. mit den in Exkurs 10 erläuterten Begriffsbildungen). Er umfasst nämlich nicht nur wahre, gerechtfertigte Meinungen[104], sondern auch

- als wahr angenommene („wahrhaftige") Meinungen (z.B. in Form von Behauptungen[105]),
- wahre, aber nicht gerechtfertigte Meinungen sowie
- Vermutungen und Hypothesen.[106]

Das derartig umrissene Wissen eines Entwicklers zu einem festen Zeitpunkt wird im Rahmen der vorliegende Arbeit mit W_{BS}^{+} bezeichnet. Nur ein Teil von W_{BS}^{+} kommt aber tatsächlich in Frage, um explizit durch Elemente der BKM adressiert zu werden. Dieser Teil wird wie bereits erläutert als W_{BS}^{-} bezeichnet. Er umfasst nur das (bis zu einem festen Zeitpunkt) tatsächlich verbal geäußerte Wissen – allerdings nicht komplett. Die Beschränkung hat sich aus erkenntnisbasierten Entscheidungen ergeben, die im Rahmen der Herleitung der BS getroffen wurden und die es (erst) möglich machten, inhaltliche Schwerpunkte zu setzen. So gilt für W_{BS}^{-}:

1. **Ausgrenzung von Vorschlägen:** Informationen, die in Form von Vorschlägen (*propose*), sei es zur Gestaltung des Arbeitsprozesses oder des

[103] Genau genommen sind hier alle Fassadenklassen auszunehmen, nicht nur die Klasse *activity*. Allerdings wird im Rahmen der BS/BKM nur die Fassadenklasse *activity* eingeführt. Die Einführung weiterer Fassadenklassen wird nachfolgenden Untersuchungen überlassen (siehe auch Abschnitt 7.7).

[104] Wie Edmund L. Gettier [71] diskutiert, umfasst diese Definition auch bestimmte Formen von „Wissen durch Glück": Die Wissensbedingungen (Wahrheit der Aussage, Überzeugung des Sprechers von der Wahrheit und die Tatsache, dass er seine Überzeugung rechtfertigen kann) sind in diesen Fällen nur deshalb erfüllt, weil der Sprecher Glück hat.

[105] Es wird noch erörtert werden, dass im Rahmen der BS/BKM nicht jede Behauptung bzw. jede Äußerung in Form eines Deklarativsatzes als Wissen zu konzeptualisieren ist.

[106] Bewusst geäußerte – und als solche erkannte – Unwahrheiten werden von der BKM nicht (explizit) als solche adressiert.

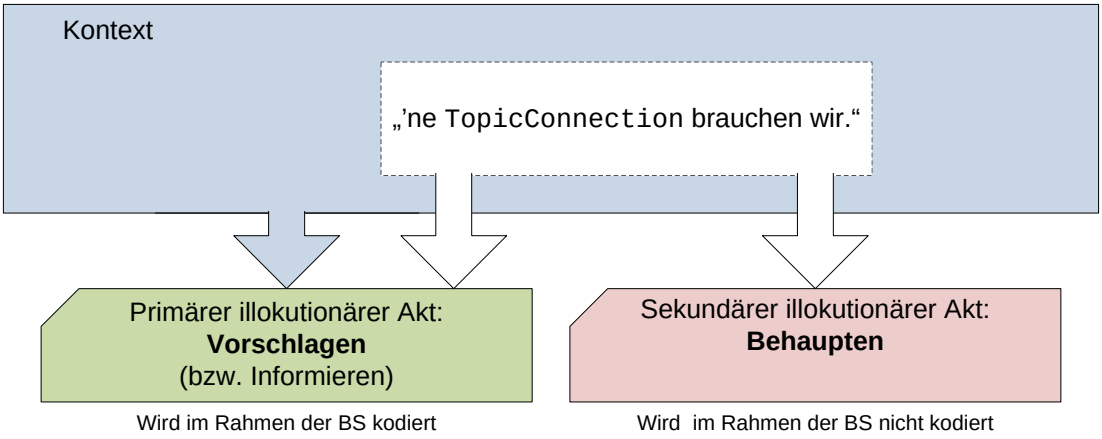


Abb. 7.9: Beispiel für das Vorgehen bei der Interpretation von indirekten Sprechakten: Wörtlich genommen handelt es sich bei der Äußerung um eine Behauptung, genau genommen um eine Äußerung, mit der der Sprecher seine Ansicht – im Kontext der BS also Wissen – kund tut. Bezieht man allerdings den Kontext der Äußerung (siehe Tabelle 7.4) und das Vertrauen des Sprechers darin, dass er und der Zuhörer über gleiche Hintergrundinformationen verfügen und der Partner die Fähigkeit zur Vernunft und zum Schlussfolgern besitzt (siehe hierzu auch die Hypothese von Searle zur Funktionsweise von indirekten Sprechakten [190, S. 60f]) mit ein, so kann man erkennen, dass es sich in erster Linie um eine Information über die Art der nächsten gestalterischen Handlung handelt, im Rahmen der BKM also um das, was als Designvorschlag zu bezeichnen ist (siehe Tabelle 7.5 über die Typisierung von *propose_design*-Äußerungen).

Produktes, bzw. in Form von Bewertungen solcher Vorschläge (*agree*, *decide*, *disagree*, *challenge*, *amend*) „mit geäußert“ werden, sind nicht W_{BS}^{-} zuzurechnen.[107] Dies gilt unabhängig davon, in welcher Weise diese Vorschläge formuliert sind. Entscheidend für die Zuordnung ist die in der Regel unter Einbeziehung des Kontextes zu identifizierende (primäre) Illokution der Äußerung als Vorschlag oder Vorschlagsbewertung bzw. als etwas, was im Rahmen der BS/BKM unter *propose* zu subsumieren ist (Details hierzu folgen in Abschnitt 9.1). So kann es sein, dass Inhalte von Äußerungen, die in Form von Aussagesätzen gemacht werden und somit wörtlich genommen wahre bzw. zumindest wahrhaftige Meinungen darstellen, z.B. „‚ne `TopicConnection` brauchen wir“[108], nicht W_{BS}^{-} zuzurechnen sind (siehe Abbildung 7.9).

In diesem Zusammenhang sollte beachtet werden, dass der Vorschlagsbegriff der BKM den vollen Umfang von dem adressiert, was man als Vorschlag

[107] Genau genommen zählen hierzu auch Informationen, die in Form von produktorientierten oder prozessorientierten Anfragen (*ask*) bzw. durch *remember_requirement* mit übertragen werden. Auch sie sind nicht W_{BS}^{-} zuzurechnen.
[108] Details zum Kontext dieser Äußerung finden sich in Beispiel 1 in Tabelle 7.4.

verstehen kann, so z.B. auch das Geben von Anweisungen.[109] Details hierzu wurden bereits in Tabelle 7.5 erläutert.

Achtung: Für die Behandlung eines Vorschlags ist es unerheblich wie er entstanden ist, ob er also z.B. aus einer spontanen Idee oder aus Bestandswissen heraus entwickelt wurde.[110]

2. **Ausgrenzung von Fortschrittsbewertungen:** Auch Informationen, die in Form von Fortschrittsbewertungen und Bewertungen von Fortschrittsbewertungen artikuliert werden, gehören nicht zu W_{BS}^{-}. Äußerungen, in denen Informationen auf diese Weise verbalisiert werden, sind mit Elementen der Klassen *state* oder *completion* zu annotieren.

Es werden also alle „Wissensinhalte“ ausgegrenzt, die im Rahmen von Äußerungen verbalisiert werden, die mit *P&P*-Konzepten zu annotieren sind. Die „Wissensinhalte“ aller anderen Äußerungen – sofern sie im Zusammenhang mit der Programmier- bzw. Entwicklungsaufgabe stehen[111] – sind W_{BS}^{-} zuzurechnen.

Achtung: Wie bereits erörtert (siehe Abschnitt 7.1.1) umfasst W_{BS}^{+} bestimmte Formen von Wissen einer Person unabhängig davon, ob die Person das Wissen verbalisiert hat, während sich W_{BS}^{-} nur auf tatsächlich verbalisiertes Wissen bezieht. Was dem Wissen W_{BS}^{-} einer Person zuzurechnen ist, entscheidet sich somit erst im Laufe einer Sitzung entlang der von ihr gemachten Äußerungen. Hierbei spielt es – wie gerade erläutert – eine Rolle, in welcher Form Wissen geäußert wird. So ist ein „Wissenspartikel“, der nur im Rahmen von produkt- oder prozessbezogenen Vorschlägen artikuliert wird, nicht W_{BS}^{-} zuzurechnen. Vielmehr handelt es sich bei dem Wissen, welches W_{BS}^{-} zuzurechnen ist, in vielen Fällen um solches, welches im Rahmen von Äußerungen artikuliert wird, deren (primäre) Illokution „Erklären“ ist, wobei Widersprüche und Ergänzungen zu Erklärungen hier auch als solche gewertet werden sollen.

Die $UK \setminus \mathit{activity}$, also alle universellen Konzepte bis auf die aus der Klasse *activity*, sind derart konstruiert, dass sie Äußerungen, mit denen Wissen vom Typ W_{BS}^{-} transportiert oder angefragt wird, explizit adressieren können. Mit ihnen können also Erklärungen sowie Bewertungen von Erklärungen markiert werden – darüber hinaus auch Fragen nach Erklärungen. Es wurde allerdings nicht versucht, das gesamte Wissen im Sinne von W_{BS}^{-} explizit durch jeweils einzelne Elemente der Menge der $UK \setminus \mathit{activity}$ adressierbar zu machen. Somit ist das von

[109] Vorschläge können auch direkt in Form eines Aufforderungssatzes formuliert sein, z.B. „Überschreibe die!“ (Paraphrase von Beispiel 2 in Tabelle 7.5).

[110] Prozess- und produktorientierte Konzepte, zumindest die der Klassen *design*, *requirement*, *step*, *todo* und *strategy*, werden nur in direktem Zusammenhang mit Vorschlägen verwendet. Dass sie trotzdem einen Großteil der Elemente der BKM bilden, spiegelt die Bedeutung wider, die Vorschläge im Rahmen eines konstruktiven, kooperativen Prozesses besitzen.

[111] Äußerungen, die nichts mit der Programmieraufgabe bzw. mit dem Lösungsprozess zu tun haben oder unverständlich bzw. nicht deutbar sind, gehören nicht zur Klasse derjenigen Phänomene, die mit universellen Konzepten zu annotieren sind. Sie werden stattdessen mit *say_off topic* oder *mumble_sth* markiert (siehe Abschnitt 8.2).

den $UK \setminus activity$ adressierte Wissen $W_{UK \setminus activity}$ eines Entwicklers (zu einem festen Zeitpunkt) nicht mit W_{BS}^{-} gleichzusetzen. Dies hat vor allem folgenden Grund: Mit einer einzelnen Äußerung werden in der Regel verschiedene Informationen oder sogar verschiedene „Formen von Wissen“ transferiert. Man betrachte hierzu folgende Beispiele:

– Mit einer Bemerkung wie „Null ist der erste Januar 1972, oder irgendsowas“ (PR1.1) wird sowohl technisches Wissen, hier über die Unixzeit[112] sowie auch eine Information darüber, dass die Angabe evtl. nicht hundertprozentig präzise ist, übermittelt.[113] Im Rahmen der BS/BKM wird der erste Wissensaspekt explizit kodiert (*explain_knowledge*), der zweite nicht (weitere Details hierzu folgen weiter unten). Es gibt zwar das Konzept *explain_standard of knowledge*, welches potentiell zur Markierung von Phänomenen der zweiten Form geeignet wäre, im Rahmen der Herleitung der BS wurde aber festgelegt, dass es in einem solchen, wie gerade beschriebenen Zusammenhang nicht eingesetzt werden soll (siehe S. 297 f). Diese Kodierregel wie auch viele andere wurden formuliert, um den Aufwand bei der Verwendung der BS/BKM in einem bewältigbaren Rahmen zu halten. In weiterführenden Untersuchungen muss allerdings darüber nachgedacht werden, ob es nötig wird, diese Regel (wie jede andere Regel auch) zu modifizieren bzw. ob der hier nicht adressierte Wissensaspekt anderweitig, z.B. durch Eigenschaften oder die Abtrennung und separate Kodierung des Zusatzes „oder irgendsowas“, zu markieren ist.

  Achtung: Genau genommen sagt eine Kodierung einer solchen Äußerung mit *explain_knowledge* nur, dass bestimmte nicht durch andere Konzepte der BKM adressierbare Formen von Bestandswissen in gewisser Weise[114] zum Zwecke der Erklärung geäußert werden. Welcher der in einer Äußerung evtl. zusammen formulierten „Wissenspartikel“ durch eine Annotation genau referenziert wird, wird durch das Konzept bzw. durch das Objekt *knowledge* nicht spezifiziert. Theoretisch könnte man also *explain_knowledge* einsetzen, um alle Partikel zusammen anzusprechen – evtl. um sie später auseinander zu dividieren. Bei der Verwendung der BS/BKM steht allerdings meist technisches Wissen oder Prozesswissen im Vordergrund, so dass *explain_knowledge* oft mit dem Ziel eingesetzt wird, die Äußerung derartiger „Wissenspartikel“ zu markieren. Dementsprechend ist der zweite Aspekt der oben diskutierten Äußerung, also das Beurteilen der Genauigkeit der Information, nicht als das Wissen zu betrachten, welches primär übermittelt wer-

[112] Die Unixzeit ist eine für das Betriebssystem UNIX entwickelte Zeitdarstellung. Mit ihr werden die Sekunden seit dem 1. Januar 1970 00:00 Uhr UTC angegeben.

[113] Für den Wert der Aussage ist es nicht entscheidend, ob der Sprecher ein präzises Datum angibt. Wichtig ist nur, die Deutung von Null als Unixzeit. Hieran lässt der Sprecher keinen Zweifel. Er artikuliert also Wissen im Sinne der Klasse *knowledge* und nicht etwa eine Vermutung im Sinne der Klasse *hypothesis*. Details hierzu werden ab S. 283 ausgeführt.

[114] Warum hier die Einschränkung „in gewisser Weise“ verwendet wird, wird auf S. 316 erläutert. Sie kann erste einmal ignoriert werden.

den soll. Er muss deshalb, wenn er in einer weiterführenden Untersuchung von Bedeutung ist, anderweitig – z.B. über eine Eigenschaft – festgehalten werden.

– Mit einer Äußerung wie „Hä, warum kennt er das hier unten nicht?“ – vom Driver in Sitzung PR2.1 geäußert, nachdem er eine Fehlermeldung des Compilers (*V cannot be resolved*) entdeckt hat – stellt der Sprecher nicht nur eine Frage, sondern verbalisiert auch seine Erkenntnis darüber, dass er die Ursache der Meldung nicht versteht. Derartige in Frageform verpackte Erkenntnisse werden von der BS/BKM nicht explizit bzw. separat als solche adressiert – es sei denn, die Frage lässt sich als rhetorisch interpretieren (siehe Beispiel zum Erkenntnistyp D_O in Tabelle 7.27). Also nur dann, wenn man in derartigen Situationen zu dem Schluss kommt, dass als primäre Illokution der Äußerung nicht Fragen festgemacht werden kann, ist eine Kodierung mit einem anderen Konzept wie *explain_finding* oder *explain_standard of knowledge* angebracht. Im vorliegenden Beispiel wurde ein *ask_knowledge* annotiert.

In Tabelle 7.24 werden die Charakteristika von W^{+}_{BS}, W^{-}_{BS} und $W_{UK\setminus activity}$ noch einmal zusammengefasst.

Die Menge der UK unterteilt sich im Wesentlichen in sechs Klassen, *knowledge*, *finding*, *standard of knowledge*, *gap in knowledge*, *hypothesis* und *activity*, wobei die Elemente der Klasse *activity* eine Sonderrolle einnehmen und deshalb separat in Abschnitt 7.7 erläutert werden. Darüber hinaus gehören noch das Element *remember_source of information* zu den UK (siehe auch Abbildung 7.3).

Grundlegende Eigenschaften der Klassen *knowledge*, *finding* und *standard of knowledge* wurden bereits in Abschnitt 7.1.1 erläutert. Diese Beschreibungen werden in den nachfolgenden Unterabschnitten präzisiert und ergänzt. Hierbei gilt:

– Wie bereits in Abbildung 7.1 erläutert wurde, unterscheidet die BKM nur in bestimmten Fällen explizit zwischen „reinen“ Externalisierungen von bereits vorhandenem Wissen und Verbalisierungen von Ergebnissen gerade stattfindender Wissensverarbeitungsprozesse (genauer: Verbalisierungen (von Ergebnissen) gerade stattfindender kognitiver Prozesse, die über das „reine“ Erinnern/Abrufen hinausgehen). In diesem Zusammenhang wurde in Hinblick auf die Klasse *knowledge* bemerkt, dass eine solche Differenzierung nicht immer mit hundertprozentiger Sicherheit vorgenommen werden kann. Es kann nicht davon ausgegangen werden, dass Erkenntnisse immer als solche identifizierbar sind. Betrachtet man den Begriff Bestandswissen diesbezüglich genauer, so wird klar, dass er im Rahmen der Verwendung der BKM eher theoretischer Natur ist und sich nur eingeschränkt operationalisieren lässt. Auch aus diesem Grund fungiert die Klasse *knowledge* innerhalb der UK als eine Art Sammelbecken. Ein anderer Grund hierfür

Klasse	Eigenschaften
W_{BS}^{+}	Explizites Wissen; geäußertes wie nicht geäußertes; keine Unterscheidung zwischen wahren und nicht wahren Meinungen – entscheidend ist nur die Wahrhaftigkeit; umfasst auch nicht gerechtfertigte Meinungen sowie Hypothesen und Vermutungen
W_{BS}^{-}	Explizites Wissen; nur bis zum Zeitpunkt t geäußertes und im Zusammenhang mit der Programmier- bzw. Entwicklungsaufgabe stehendes; keine Unterscheidung zwischen wahren und nicht wahren Meinungen – entscheidend ist nur die Wahrhaftigkeit; umfasst auch nicht gerechtfertigte Meinungen sowie Hypothesen und Vermutungen; umfasst bestimmte geäußerte „Wissenspartikel“ nur dann, wenn sie nicht nur in Äußerungen „verpackt“ gemacht wurden, die mit prozess- oder produktorientierten Konzepten zu annotieren sind.
$W_{UK\setminus activity}$	Explizites Wissen; nur bis zum Zeitpunkt t geäußertes und im Zusammenhang mit der Programmier- bzw. Entwicklungsaufgabe stehendes; nur explizit durch $UK \setminus activity$ adressierbares (impliziert unter anderem: Keine Unterscheidung zwischen wahren und nicht wahren Meinungen – entscheidend ist nur die Wahrhaftigkeit; umfasst auch nicht gerechtfertigte Meinungen sowie Hypothesen und Vermutungen)

Tab. 7.24: Charakteristika der Klassen W_{BS}^{+}, W_{BS}^{-} und $W_{UK\setminus activity}$. Alle Angaben beziehen sich auf das Wissen eines einzelnen Entwicklers zu einem festen Zeitpunkt t. Vernachlässigt man die Tatsache, dass Entwickler auch innerhalb einer Sitzung Informationen vergessen, gilt zu jedem festen Zeitpunkt $W_{UK\setminus activity} \subset W_{BS}^{-} \subset W_{BS}^{+}$, wobei W_{BS}^{-} im Rahmen des Herleitungsprozesses und auf Basis struktureller oder strategischer Überlegungen sukzessive definiert und $W_{UK\setminus activity}$ unter Berücksichtigung von bzw. im Wechselspiel mit Vorversionen von W_{BS}^{-} sukzessive aus den Daten konstruiert wurde.

besteht darin, dass alle Äußerungen, die nicht von den anderen Objektklassen (abgesehen von *activity*) adressiert werden erst einmal in die Klasse *knowledge* fallen. Es wird aber davon ausgegangen, dass es sich bei den durch Elemente der Klasse *knowledge* zu annotierenden Äußerungen zum weitaus überwiegenden Teil um Verbalisierungen von Bestandswissen handelt – nicht immer aber zwangsläufig um solche, die primär eine Erläuterung dieses Wissens zum Ziel haben (für Details siehe S. 315 f).

– Über bestimmte Konzepte wird die vom Sprecher artikulierte Sicherheit in Hinsicht auf die Richtigkeit seiner Äußerung adressiert. So werden initiative Äußerungen, an deren Korrektheit der Sprecher Zweifel lässt, von Konzepten der Klasse *hypothesis* adressiert. Details hierzu sind in Matrix 7.25 dargestellt. Die sich hieraus ergebenden „Vorfahrtsregeln“ bei der Kodierung werden in Abbildung 7.10 erläutert.

– In den nachfolgenden Diskussionen werden die Bezeichnungen Wissen und *knowledge* nicht synonym verwendet. In der Regel bezieht sich der deutsche

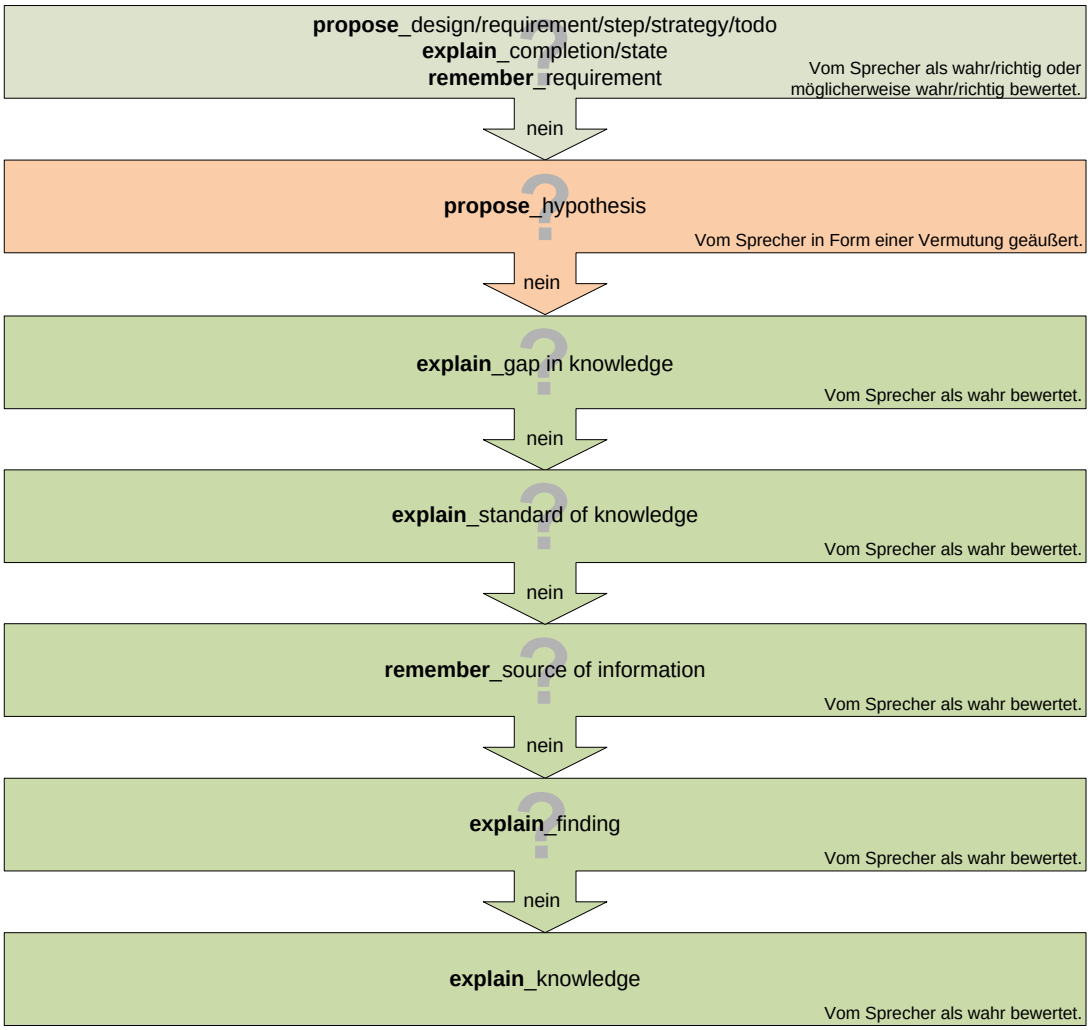


Abb. 7.10: Darstellung der Sequenz, in der initiative Äußerungen in Hinsicht auf ihre Bedeutung zu analysieren sind (im Weiteren auch *Vorfahrtsregeln* genannt). So sollte als erstes überprüft werden, ob es sich um einen Vorschlag zur Gestaltung von Artefakten bzw. des Prozesses, eine Fortschrittsbewertung oder um eine initiative Äußerung in Hinsicht auf Anforderungen handelt. Falls dem nicht so ist, sollte sich der Forscher fragen, ob es sich bei der Äußerung um die Verbalisierung einer Hypothese/Vermutung handelt usw. Die Abbildung macht darüber hinaus deutlich, dass die Untersuchung einer Äußerung zuerst ungeachtet der Überzeugung des Sprechers in Hinsicht auf die Korrektheit (bzw. Angemessenheit) seiner Aussage verlaufen kann. Erst wenn sichergestellt wurde, dass es sich nicht um eine mit prozess- oder produktorientierten Konzepten zu annotierende Äußerung handelt, wird eine solche Unterscheidung notwendig. Dann sollte zuerst überprüft werden, ob es sich um eine Vermutung bzw. Hypothese handelt. Folgende Punkte sollten beachtet werden: (a) Die Abbildung berücksichtigt nicht, dass auch im Rahmen der BS/BKM Doppelkodierungen notwendig und sinnvoll sein können (siehe z.B. S. 205). (b) Die Abbildung berücksichtigt keine Äußerungen vom Typ *ask*. (c) Im Rahmen der Verwendung der BKM wird es in der Regel nicht notwendig sein, dem hier erläuterten Vorgehen strikt zu folgen, da oft ohnehin sofort ersichtlich ist, welches Konzept geeignet ist.

	Bestandswissen	**Erkenntnisse**
Der Sprecher geht von der Richtigkeit seiner Äußerung aus.	*explain_knowledge* *explain_gap in knowledge* *explain_standard of knowledge* *remember_source of information*	*explain_finding* *explain_gap in knowledge* *explain_standard of knowledge*
Der Sprecher äußert eine Vermutung.	*propose_hypothesis*	*propose_hypothesis*

Tab. 7.25: Einteilung der initiativen und bivalenten *UK* in Klassen. Hierbei wird einerseits zwischen Äußerungen, in denen Bestandswissen artikuliert wird, und Äußerungen, in denen gerade erlangte Erkenntnisse verbalisiert werden, unterschieden und andererseits danach differenziert, ob der Sprecher sein Wissen als Wahrheit oder als Vermutung artikuliert. *activity*- und *ask*-Konzepte sind in der Matrix nicht berücksichtigt. Man beachte auch – speziell in Hinsicht auf die Trennschärfe von *explain_knowledge*-Annotationen – die Ausführungen in Abbildung 7.1 auf S. 155.

Begriff Wissen erst einmal auf $W_{UK\setminus activity}$ oder W_{BS}^{-}. Mit *knowledge* wird hingegen nur ein Teil dieses Wissens referenziert (siehe Abbildung 7.1). Abweichungen von dieser Regel, die gelegentlich bei der Erörterung einzelner Konzepte oder Klassen auftreten, z.B. aus Mangel an anderen geeigneten Begrifflichkeiten (z.B. wird im Zusammenhang mit Vorschlägen von „mit übermittelten“ Wissen gesprochen, das eben nicht von UK adressiert wird), sind jeweils aus dem Kontext ersichtlich.[115]

– Die in der Informatik und dort im Speziellen im Wissensmangement verbreitete Differenzierung nach Daten (*data*), Informationen (*information*), Wissen (*knowledge*) und Weisheit (*wisdom*), kurz *DIKW hierarchy* (siehe z.B. [177]), ist im Rahmen der BS/BKM nur von eingeschränkter Bedeutung. Es sollte aber beachtet werden, dass von der BS/BKM nur durch Sprache und in Ausnahmefällen durch Bilder externalisiertes Wissen adressiert wird. Solches wird oft nicht mehr als Wissen, sondern als *Information* bezeichnet (siehe z.B. auch [9]). Im Rahmen der vorliegenden Arbeit werden die beiden Begriffe aber in der Regel, zumindest sofern es um Erläuterungen

[115] Es stellt sich die Frage, inwieweit es überhaupt sinnvoll ist, von Wissen oder *knowledge* zu sprechen bzw. ob nicht besser andere Begrifflichkeiten gewählt werden sollten. Im Rahmen der Herleitung der BKM konnten allerdings keine geeigneteren Konzeptnamen und Bezeichnungen identifiziert werden, abgesehen davon, dass es sinnvoll sein könnte, statt von *knowledge* besser von *memory* (verbalisierbare Erinnerung – im Sinne des „sich erinnern“) zu sprechen (siehe S. 315f). Begriffe wie z.B. Kenntnisse, Fachwissen, Nachrichten oder auch *knowledge base*, *skills* oder *cognition* sind spezieller, werden zum Teil mit im Rahmen der BKM nicht gewünschten Bedeutungen assoziiert und sind in der Regel letztendlich genauso schwer einzugrenzen wie der Terminus Wissen. In spezialisierten Untersuchungen kann es aber durchaus sinnvoll sein, solche Begriffe näher zu spezifizieren und zu verwenden, z.B. zur Bezeichnung von Subkonzepten oder Eigenschaften.

zum Wissensbegriff der BS geht, synonym verwendet (siehe z.B. Fußnote auf S. 197 oder auf S. 240). Hierbei wird sowohl das verbalisierte Wissen im Sinne von $W_{UK\backslash activity}$ (siehe z.B. Fußnote auf S. 197) wie auch sonstiges verbalisiertes Wissen (siehe z.B. S. 240) als Information bezeichnet.

Exkurs 10: *Der Wissensbegriff der BS/BKM im philosophischen Kontext*

Der Philosophie ist es bisher nicht gelungen, eine allgemein akzeptierte Definition des Begriffs Wissen hervorzubringen.[a] Vielmehr existieren unterschiedliche Begriffsbildungen bzw. Begriffsbildungsansätze. So stimmen diejenigen Vertreter der Gegenwartsphilosophie, „die ihre Definitionsversuche am alltäglichen Sprachgebrauch“ messen und deren „alltagssprachliche Formel für Wissen [...] 'S weiß, dass p'“ (S ist eine Person, p eine Aussage) lautet, „mehrheitlich darin überein, dass Wissen eine bestimmte Form *wahrer Überzeugung* ist“ [84, S. 3] (‚S glaubt, dass p' und ‚p ist wahr'). Um letztendlich von Wissen sprechen zu können, wird darüber hinaus eine so genannte Begründungsanforderung, z.B. „S kann seinen Glauben, dass p, rechtfertigen oder beweisen“ oder „Die Annahme von S, dass p, ist fundiert“ (siehe [112, S. 17]), hinzugefügt[b], wobei Uneinigkeit darüber herrscht, wie diese Anforderung genau zu formulieren ist bzw. auf welche Weise sie erfüllt sein sollte (siehe [71]). Definitionen der Art „wahre, begründete Meinung“[c] werden allerdings von vielen Seiten kritisiert bzw. hinterfragt. Ein Kritikpunkt[d] besteht darin, dass diese „viele Verwendungsmöglichkeiten von ‚Wissen'“ ausschließen [84, S. 4], d.h. nur die intersubjektive Erklärung adressieren und von anderen Formen wie der intuitiv-subjektiven Evidenz oder dem nicht oder nicht völlig sprachlich explizierbaren Können (praktisches Wissen) wegführen [84, S. 5]. Die BS und insbesondere die BKM entzieht sich dieser Diskussion weitgehend dadurch, dass HHI-Konzepte nur verbale Äußerungen adressieren, also ausschließlich explizites Wissen, bzw. wahrhaftige Meinungen im Fokus haben, und dabei nicht zwischen intersubjektiver Erklärung und intuitiv-subjektiver Evidenz unterscheiden. Die BS/BKM folgt in gewisser Weise – ohne dass dies vorher festgelegt wurde – der konstruktivistischen Sicht von Ernst von Glasersfeld [78, S. 202][e]: „Wissen ist kein Bild oder keine Repräsentation der Realität, es ist vielmehr eine Landkarte dessen, was die Realität uns zu tun erlaubt. Es ist das Repertoire an Begriffen, begrifflichen Beziehungen und Handlungen oder Operationen, die sich in der Verfolgung unserer Ziele als viabel, d.h. gangbar erwiesen haben.“[f]

[a] Gottschalk [84, S. 5] stellt in Frage, ob es überhaupt sinnvoll ist, mit einer allgemeingültigen Definition zu arbeiten und plädiert dafür, Wissen als mehrdeutigen *Komplexbegriff* (bewusst offen angelegtes Containerwort) zu betrachten, der bei Diskussionen in einem bestimmten Kontext von den einzelnen Diskussionspartnern „aus ihrem Vorverständnis und gemäß ihrer Diskussionsziele hinreichend“ aufgefüllt werden kann (Merkmalsbildung in einem Kontext).

[b] Wird dann auch als propositionales Wissen bezeichnet.

[c] Dieser Definitionsversuch, oft auch als die *Standardanalyse von Wissen* bezeichnet, scheint sehr nahe an dem zu sein, „was bereits Platon als Erläuterung des Wissensbegriffs erwog“ [60, S. 70].

[d] Im Rahmen der vorliegenden Arbeit ist es weder möglich noch angezeigt, alle Kritikpunkte bzw. alle philosophischen Ansätze in Hinsicht auf den Begriff Wissen zu erläutern.

[e] Ernst von Glaserfeld gilt als Begründer des *Radikalen Konstruktivismus*, welcher vereinfacht gesprochen bestreitet, dass es dem Menschen möglich ist, objektive Realität zu erkennen, „daß alles Wissen, wie auch immer man es definieren mag, nur in den Köpfen der Menschen existiert, und daß das denkende Subjekt sein Wissen nur auf der Grundlage seiner Erfahrung konstruieren kann“ [79, S.11]. Es muss festgehalten werden, dass obwohl hier eine Begriffsbildung aus dem Umfeld des Radikalen Konstruktivismus zur Erläuterung herangezogen wird, die BS nicht unter spezieller Berücksichtigung dieser Theorie entwickelt wurde.

[f] Inwieweit die Einordnung durch das Adjektiv „gangbar“ mit der Tatsache verträglich ist, dass im Rahmen der BS/BKM auch Erkenntnisse dem Wissen zugerechnet werden, kann hier aus Platzgründen nicht erörtert werden.

7.6.1 *finding*-Konzepte

Fokus der *finding*-Konzepte

Wie bereits in Abbildung 7.1 illustriert wurde, adressieren die Konzepte der Klasse *finding* nicht alle, sondern nur bestimmte im Laufe einer PP-Sitzung entstehende Erkenntnisse und Erkenntnisbewertungen bzw. genau genommen nur bestimmte Verbalisierungen dieser. So werden diejenigen Äußerungen, die schon durch die Konzeptklassen *standard of knowledge*, *gap in knowledge*, *hypothesis*, *source of information* oder produkt- bzw. prozessorientierte Konzepte behandelt werden, auch dann nicht *finding* zugerechnet, wenn es sich bei ihnen offensichtlich um Verbalisierungen neuer Erkenntnisse handelt. Alle anderen Verbalisierungen von Erkenntnissen fallen hingegen potentiell in die Klasse *finding*. Es stellt sich allerdings die Frage, welche Formen von Äußerungen als Artikulationen von Erkenntnissen akzeptiert werden sollen oder anders ausgedrückt: Was genau soll unter einer Erkenntnis verstanden werden und wann kann man sich hinreichend sicher sein, dass es sich bei einer konkreten Äußerung um die Verbalisierung einer solchen handelt? Mit einer Antwort auf den ersten Teil der Frage soll geklärt werden, wie die BS den Begriff Erkenntnis prinzipiell verstanden haben will, mit einer Antwort auf den zweiten Teil, in welchen Situationen und anhand welcher Indizien derartige Erkenntnisse als identifizierbar wahrgenommen werden sollen. Achtung: Diese Situationen und Indizien wurden im Laufe der Analyse aus den Daten extrahiert. Hierbei wurden nur Erkenntnisse/Phänomene berücksichtigt, die nicht schon anderweitig adressiert werden. Das heißt, dass Erkenntnisse, die z.B. im Rahmen von Prozessvorschlägen geäußert wurden, keine Berücksichtigung bei der Herleitung der Erkenntnistypen fanden. Schließlich wird hier definitionsgemäß nicht zwischen Bestandswissen und Erkenntnissen differenziert. Dies wird weiterführenden Untersuchungen bzw. anderen Schichten überlassen.

Der Erkenntnisbegriff der BS: Im Rahmen der BS wird der Begriff Erkenntnis nicht im Sinne der Wissenschaftstheorie bzw. Philosophie verwendet, sondern als ein der Psychologie nahe stehender Terminus, der das Ergebnis (nicht den Weg) kognitiver Prozesse bzw. das Ergebnis der Ausübung kognitiver Fähigkeiten bezeichnet. Zu diesen Prozessen und Fähigkeiten gehören Gedächtnis, Orientierung (das heißt, das sich in Hinsicht auf Zeit, Ort, Situation und eigene Person Zurechtfinden), Aufmerksamkeit und Konzentration, Reizleitungs- bzw. Reaktionsgeschwindigkeit, Sprache, abstraktes und logisches Denken, Handlungsplanung und Handlungsabfolgen vollziehen sowie Kulturtechniken (Rechnen, Schreiben, Lesen) [184]. Natürlich ist es nicht sinnvoll, alle Ergebnisse kognitiver Prozesse als Erkenntnisse zu bezeichnen. Im Rahmen der BS gilt vielmehr: Damit ein solches Ergebnis als Erkenntnis bezeichnet werden kann, muss es eine während einer PP-Sitzung zumindest in Teilen eigenständig neu erlangte Einsicht darstellen. So darf es sich beispielsweise nicht nur um das Resultat „reiner Er-

innerungsprozesse“[116] handeln. Damit eine Erkenntnis als eine vom Typ *finding* bewertet werden kann, muss sie darüber hinaus geäußert und dabei als zweifelsfreie Wahrheit dargestellt werden.[117]

Nachfolgend wird erläutert, inwieweit genau eine über das reine Erinnern hinausgehende kognitive Leistung von Nöten ist, damit eine Äußerung als das Kundtun einer Erkenntnis, genau genommen einer Erkenntnis vom Typ *finding* gewertet werden soll bzw. wann eine solche Leistung als erkennbar gelten soll. Vorher beachte man aber noch folgendes: Übermittelt ein Entwickler seinem Partner Teile seines Wissens und bestätigt der Angesprochene, dass er dieses „verstanden“ hat, z.B. durch ein „Jetzt ist es mir klar“, so wird dies im Rahmen der BS nicht explizit als Erkenntnis bzw. Erkenntnisäußerung betrachtet und markiert. Die BS verhält sich an dieser Stelle „neutral“, indem sie vorschreibt, die Rückmeldung mit dem Konzept *explain_standard of knowledge* zu annotieren (für Details siehe S. 297f). Dies ist auch in anderen Zusammenhängen derart vorgesehen und zeigt, dass im Rahmen der BS neues Wissen nicht unbedingt explizit als Erkenntnis klassifiziert wird.

Achtung: Wie bereits erläutert werden im Rahmen der BS/BKM nicht Erkenntnisse selbst, sondern nur Äußerungen, in denen Erkenntnisse verbalisiert werden, konzeptualisiert. Da aber eine solche Erkenntnisäußerung nicht unbedingt zeitlich mit dem Erlangen der Erkenntnis zusammenfallen muss, stellt sich die Frage, wann eine Erkenntnisäußerung noch als eine solche gewertet werden soll bzw. ab wann man eher von einer Äußerung sprechen sollte, mit der Bestandswissen vermittelt wird. Grob gesprochen soll diesbezüglich gelten:

– Erkenntnisse, die bereits vor einer Sitzung erlangt wurden, werden im Rahmen einer Sitzungskodierung – soweit es die BS/BKM betrifft – als Be-

[116] Der Terminus „reiner Erinnerungsprozess“ lässt sich kaum präzise fassen und soll hier für einen „weitgehend verarbeitungsfreien“ Abruf von gespeicherten Informationen stehen, also für einen solchen, bei dem z.B. keine Kombinationen von Wahrnehmungs- und Gedächtnisinhalten stattgefunden haben. Es soll an dieser Stelle nicht interessieren, ob oder inwieweit es einen solchen überhaupt gibt. Die Beschreibung soll nur grob veranschaulichen, welche Formen von kognitiven Leistungen für sich allein genommen nicht zu dem führen sollen, was im Rahmen der BS Erkenntnis genannt wird. Es sollte aber beachtet werden, dass diese Charakterisierung nicht in allen denkbaren Fällen unbedingt zu einer in jeder Hinsicht befriedigenden Beschreibung von dem, was als Erkenntnis zu bezeichnen ist führt. So würde man beispielsweise die Verbalisierung von anscheinend vergessenem und dann plötzlich (unmittelbar) wiedererlangtem Wissen in der Regel gerne mit *explain_finding* annotieren, auch wenn nicht klar ist, ob oder welche über das Erinnern hinausgehenden kognitiven Prozesse zu dieser Erkenntnis geführt haben. Für die Klassifizierung als Erkenntnis ausreichend könnte man hier schon eine Phrase wie „Jetzt fällt es mir wieder ein ...“ ansehen.

[117] Diese Festschreibung kann zu Schwierigkeiten bei der Kodierung führen. Man betrachte hierzu eine Äußerung wie z.B. „Ahhh“, mit der *P1* in Sitzung PR1.1 zum Ausdruck bringt, dass er erkannt hat, dass bei einem Testaufruf des gerade modifizierten Kodes etwas passiert ist (Details hierzu werden erst in Tabelle 8.5 in Abschnitt 8.1.4 erläutert). Wie noch erläutert werden wird, sind solche Wahrnehmungsäußerungen als Erkenntnisäußerungen zu kodieren, auch dann, wenn die eigentliche Erkenntnis gar nicht expliziert wird. Allerdings ist ihnen direkt kein Wahrheitswert zuzuschreiben. Erst wenn man sie geeignet paraphrasiert, z.B. in ein „Ich habe gerade erkannt, dass etwas passiert ist“, ist dies möglich.

standswissen gewertet. Dies ist natürlich nur dann möglich, wenn „alte“ Erkenntnisse als solche erkennbar geäußert werden.

– Bereits geäußerte Erkenntnisse, die zu einem späteren Zeitpunkt wiederholt werden, werden dann als Bestandswissen gewertet. Es wird davon ausgegangen, dass ein Lernprozess stattgefunden hat. Dieser wird von der BS/BKM allerdings nicht explizit adressiert. Diese Regel gilt nicht, wenn die Wiederholung zeitnah[118] erfolgt, z.B. um sicherzustellen, dass der Partner die Erkenntnis mitbekommen hat. Details hierzu werden auf S. 267f und in Abschnitt 9.3.7 erläutert.

Indikatoren für Äußerungen, die der Klasse *finding* zugerechnet werden müssen:[119] Bei der Herleitung der BKM war es möglich, eine Reihe von Äußerungen zu identifizieren, für die unter Berücksichtigung ihres aktuellen Kontextes (also der Situation) hinreichend eindeutig gezeigt werden kann, dass es sich um Verbalisierungen von Erkenntnissen handelt.[120] Auf Basis dieser Äußerungen/Situationen wurden so genannte *Erkenntnisäußerungstypen*, im Weiteren verkürzt *Erkenntnistypen* genannt, spezifiziert, wobei hierbei wie bereits erwähnt nur solche Phänomene Berücksichtigung fanden, die nicht den Klassen *design*, *requirement*, *step*, *completion*, *strategy*, *state*, *todo*, *standard of knowledge*, *gap in knowledge*, *hypothesis* oder *source of information* zuzurechnen waren. Die Erkenntnistypen sollen in erster Linie dabei helfen, den vollen Umfang der Bedeutung der Klasse *finding* – zumindest soweit es den hier dargestellten Wissensstand über die BS/BKM betrifft – zu begreifen und *finding*-Konzepte als solche identifizieren zu können. Im Einzelnen handelt es sich um die folgenden Typen:

– **Erkenntnistyp P (*perceived event*)**: Eine Äußerung ist vom Typ P, wenn in ihr die Wahrnehmung (in der Regel durch Sehen oder Hören) von sich zeitnah in der Umgebung der Person abgespielten Ereignissen bzw.

[118] Bei der Angabe „zeitnah“ handelt es sich um kein präzises bzw. in jedem Fall objektiv überprüfbares Kriterium. Sie bedeutet in diesem Zusammenhang soviel wie: In derselben Situation bzw. im selben näheren Kontext. Mit „zeitnah“ soll dasjenige Intervall bezeichnet sein, in dem eine (eigene) Erkenntnis noch als neu interpretierbar scheint.

[119] Achtung: Im weiteren Verlauf wird erläutert werden, dass es konstruktive Äußerungen gibt, in denen der Sprecher keine eigenen BS-Erkenntnisse formuliert und die trotzdem der Klasse *finding* zugerechnet werden. Dies betrifft z.B. Ergänzungen zu Erkenntnissen (siehe z.B. S. 335).

[120] Die Phrase „hinreichend eindeutig gezeigt werden kann“ ist an dieser Stelle nicht mit „sicher nachgewiesen werden kann“ gleichzusetzen. Ein sicheres Nachweisen ist in der Regel schon aufgrund der nicht vollständig eindeutigen Definition des Begriffs „Erkenntnis“ kaum möglich. Die Phrase soll vielmehr bedeuten, dass nachvollziehbar dargelegt werden kann, warum eine Äußerung als Verbalisierung einer Erkenntnis gewertet wird.

Geschehnissen verbalisiert wird.[121] Ausgeschlossen seien hierbei nur Äußerungen des Partners, es sei denn, sie sind vom Typ *finding*. Dabei kann ein Ereignis die Folge einer vorhergehenden Benutzeraktion sein, z.B. die eines Testaufrufs. Die Äußerungen einer Erkenntnis vom Typ P beinhalten oft auch die Interpretation des Geschehens als eine bestimmte Form von Ereignis, z.B. als erwünschtes oder erwartetes Ergebnis einer Operation.

Der Erkenntnistyp P verdeutlicht, dass es sich bei Erkenntnissen im Sinne der BS/BKM nicht immer um solche handelt, die die Entwickler selbst explizit als solche bezeichnen würden.

– **Erkenntnistyp D (*discovered issue*)**: Äußerungen, die einen direkten Bezug zu auf dem Display bzw. Monitor oder in einem Dokument dargestellten und offensichtlich auch gerade betrachteten Informationen, im Weiteren auch kurz *gerade wahrgenommene passive Informationen* genannt, besitzen. Hierbei kann es sich um die Verbalisierung der Entdeckung von (potentiellen) Defekten, Problemen, Auffälligkeiten bzw. sonstiger in der Situation besonders relevant erscheinender Stellen handeln. Auch Beurteilungen von gerade betrachteten Artefakten oder Artefaktteilen werden dem Typ D zugerechnet (inkl. Beurteilungen gerade geschriebener Kodezeilen). Schließlich handelt es sich bei einem neu gebildeten Urteil um das Ergebnis kognitiver Prozesse, die über das reine Erinnern hinausgehen. Erkenntnisse vom Typ D unterscheiden sich vor allem dadurch von solchen vom Typ P, dass sie sich nicht direkt auf gerade stattfindende oder stattgefundene Ereignisse beziehen.

  Anmerkungen:

  – Eine auf dem Display dargestellte Information soll hier auch dann noch als sichtbar gelten, wenn sie gerade aus dem sichtbaren Bereich verschwunden ist, aber anhand der betrachteten Äußerung bzw. ihres näheren Kontextes davon ausgegangen werden kann, dass sie vorher vom Sprecher wahrgenommen wurde.

[121] Diese Festlegung impliziert beispielsweise, dass es sich bei Äußerungen eines Observers, die sich direkt auf das Scrollen durch Kode beziehen, um Verbalisierungen von Erkenntnissen vom Typ P handelt, ebenso bei Observeräußerungen, die sich auf das Erscheinen bestimmter Kodeteile beim Scrollen beziehen. Demgegenüber stellt die Verbalisierung der Entdeckung eines Fehlers in diesen Kodeteilen zwar auch eine Erkenntnisäußerung dar, allerdings keine vom Typ P. Schließlich wird nicht das Erscheinen an und für sich erkannt und verbalisiert. Anders verhält es sich, wenn der Driver neuen Kode schreibt und ein Observer diesen sofort als fehlerhaft bewertet. In dieser Situation bezieht sich ein Observer nämlich in der Regel direkt auf das Ereignis „Schreiben von fehlerhaftem Kode“. Achtung: Inwieweit die hier dargestellten Formen von Differenzierungen sinnvoll und pratikabel sind, um Erkenntnistypen als Annotationen zu verwenden, soll hier nicht weiter diskutiert werden – obwohl die Tatsache, dass im dritten hier diskutierten Fall auch von dem Ereignis „Erscheinen einer fehlerhaften Kodestelle“ gesprochen werden könnte, Abgrenzungsprobleme andeutet (man beachte hierzu die Anmerkungen auf S. 257).

– Beim Vorlesen von Inhalten handelt es sich oft auch um Äußerung von Erkenntnissen vom Typ D, wenn auch in gewisser Weise um entartete.

Es wurden folgende Untertypen von D beobachtet:

– **Erkenntnistyp D_U (*uncatalyzed discovered issue*)**: Äußerungen, in denen eine eigenständige und nicht durch eine Bemerkung des Partners oder ein anderes Ereignis beförderte Entdeckung von (potentiellen) Defekten, Problemen oder Auffälligkeiten verbalisiert wird. Der Erkenntnistyp D_U adressiert nur solche Verbalisierungen, die sich auf den Inhalt von Artefakten (z.B. Programmkode) beziehen.

– **Erkenntnistyp D_C (*catalyzed discovered issue*)**: Äußerungen, in denen eine Entdeckung von (potentiellen) Defekten, Problemen oder Auffälligkeiten verbalisiert wird und denen eine Bemerkung des Partners vorausgegangen ist, welche auf diese Defekte, Probleme oder Auffälligkeiten hindeutet, ohne diese aber tatsächlich zu benennen bzw. ein Erkennen dieser zu artikulieren. Statt einer Bemerkung des Partners kann auch ein anderes Ereignis mit dieser Eigenschaft vorausgegangen sein. Wie der Erkenntnistyp D_U adressiert auch der Typ D_C nur solche Verbalisierungen, die sich auf den Inhalt von Artefakten (z.B. Programmkode) beziehen.

  Achtung: Auch *amend_finding*-Äußerungen können vom Typ D_C sein. Dies ist z.B. der Fall, wenn das zugehörige *explain_finding* vom Typ D_U war und der Partner diese Erkenntnis aufgrund eigener Beobachtungen um zusätzliche Informationen erweitert (siehe z.B. Tabelle 7.26).

– **Erkenntnistyp D_O (*observation*)**: Äußerungen vom Typ D, in denen sonstige Beobachtungen verbalisiert werden. Dies können z.B. Beobachtungen sein, die sich auf die Konfiguration von Entwicklungswerkzeugen beziehen.[122]

– **Erkenntnistyp T (*thought*)**: Äußerungen, mit denen die Ergebnisse von Überlegungen oder spontanen bzw. spontan erscheinenden Einfällen und Einsichten verbalisiert werden.[123] Solche Äußerungen haben in der Regel keinen erkennbaren unmittelbaren und direkten Bezug zu gerade stattgefundenen Ereignissen oder gerade wahrgenommenen passiven Informationen. Für den Typ T wurden folgende Untertypen beobachtet:

[122] Auf eine Differenzierung zwischen unkatalysierten und katalysierten Äußerungen wird in Bezug auf Erkenntnisse der Klasse D_O verzichtet.

[123] Die Begriffe „spontan“ und „spontan erscheinend“ sollen hier soviel wie „ohne lange Überlegung“ bzw. „ohne als solche wahrnehmbare lange Überlegung“ bedeuten. Es muss also keineswegs so sein, dass es sich aus Sicht des Sprechers wirklich um spontane Einfälle bzw. Einsichten handelt.

HHI-Kode	Äußerung	Kontext/Kommentar
P1.explain_finding	„Dis `get` (..) (fehlt).“	**Erkenntnistyp** D_U: Nachdem sich beide Entwickler ca. 10 Sekunden lang den Kode der Klasse angesehen haben (dieser passte vollständig auf den Bildschirm), bemerkt *P1*, dass eine bestimmte `get`-Methode nicht implementiert ist.
P1.amend_finding	„Der holt das aus dem `Abstract`.“	**Erkenntnistyp** D_U: Nach ca. zwei Sekunden Stille, in denen der Partner nicht reagiert, erweitert *P1* seine Erkenntnisäußerung um weitere Informationen. Hierbei zeigt er auf die Kopfzeile der Klasse, aus der hervorgeht, dass `FeatureAttributeConfigurationProxy` die Klasse *`Abstract`*`FeatureAttributeConfiguration` erweitert. Allem Anschein nach will er zum Ausdruck bringen, dass die in seiner vorherigen Äußerung erwähnte `get`-Methode aus dieser Klasse geerbt wird, was seiner Meinung nach nicht richtig ist.
P2.amend_finding	„Der darf nicht den `Abstract` nehmen.“	**Erkenntnistyp** D_C: Der Driver fällt *P1* ins Wort. Seine Äußerung entspricht inhaltlich weitgehend derjenigen des Partners. Es wird nicht klar, ob es sich wirklich um eine eigenständige Erkenntniserweiterung handelt.
P1.agree_finding	„Ja.“	*P1* stimmt *P2* unmittelbar zu.
P1.propose_design	„Du musst die überschreiben.“	Direkt im Anschluss schlägt *P1* vor, die *get*-Methode zu überschreiben,
P2.mumble_sth+ P2.popose_step	„Der darf überhaupt nicht (!...!). Ja. Was ist im `Abstract` alles implementiert - mal kurz schauen.“	*P2* schlägt vor, erst einmal nachzusehen, welche Methoden in der Klasse *`Abstract`*`FeatureAttributeConfiguration` implementiert sind.
		Fortsetzung auf der nächsten Seite.

		Fortsetzung der Tabelle der vorherigen Seite.
P1.challenge_design	„Eigentlich dürfte er gar nicht vom `Abstract` erben. (..) Oder wir dürften das nicht im Abstract drin haben. Eins von beiden.“	Unmittelbar nach seinem Vorgehensvorschlag widerspricht er dem Designvorschlag von *P1*, indem er eine Alternative formuliert. Diese enthält zwei Optionen. Während er spricht, wechselt er in die Klasse `Abstract`FeatureAttributeConfiguration

Tab. 7.26: Beispielepisode aus Sitzung PR2.1 (12:42:37–12:42:57), in der mehrere Erkenntnisse verbalisiert werden. Die Episode startet damit, dass das Paar beginnt, bestehenden Kode durchzusehen. Hierzu hat der Driver gerade die Klasse `Feature-AttributeConfigurationProxy` in einem Eclipse-Editor geöffnet. Diese Klasse erweitert `AbstractFeatureAttributeConfiguration`. *P2* nimmt in dieser Episode die Rolle des Drivers ein. Die Erkenntnisse beziehen sich auf einen (vermeintlichen) Fehler im bestehenden Kode.

- **Erkenntnistyp T_B (*bettering*)**: Äußerungen, mit denen vorhergegangene eigene Äußerungen (vom Typ *knowledge* oder *finding*) korrigiert oder als nicht richtig bewertet werden, ohne dass identifizierbare äußere Ereignisse dies ausgelöst haben. Es handelt sich also um eigenständig erlangte Erkenntnisse darüber, dass die ursprünglichen Äußerungen zumindest nicht vollständig richtig oder angemessen waren.
- **Erkenntnistyp T_U (*uncatalyzed idea*)**: Äußerungen, mit denen spontane Einfälle und Einsichten verbalisiert werden, ohne dass zuvor diesbezüglich relevante, von außen kommende Ereignisse stattgefunden haben, die solche Erkenntnisse mit ausgelöst oder angestoßen, im Gewissen Sinne also katalysiert haben könnten. In die Klasse T_U fallen nur solche Äußerungen, die nicht schon durch T_B abgedeckt werden.
- **Erkenntnistyp T_C (*catalyzed idea*)**: Äußerungen, in denen spontane Einfälle und Einsichten verbalisiert werden, die sich an zuvor von außen kommenden Ereignissen katalysiert haben. Für die Einordnung als Erkenntnisäußerung von Typ T_C ist es hierbei wichtig, dass aus dem katalysierenden Ereignis selbst die Erkenntnis nicht direkt ohne das Einbeziehen weiterer Informationen entstehen kann. Genau genommen sollte aus dem Ereignis noch nicht einmal die inhaltliche Ausrichtung der Erkenntnis (z.B. „Erkennen eines bestimmten Programmierfehlers“) direkt ablesbar sein. Wie bei T_U fallen in die Klasse T_C nur solche Äußerungen, die nicht schon durch T_B abgedeckt werden.

Indizien bzw. Situationsmuster, anhand derer es möglich ist, diese Erkenntnistypen zu identifizieren, werden in Tabelle 7.27 aufgeführt und mit Beispielen bebildert. Ein weiteres Beispiel wurde bereits in Tabelle 7.13 erläutert. Eine Bei-

spielepisode mit mehreren Erkenntnissen (vom Typ P und T_U) ist in Tabelle 7.28 dargestellt. Weitere Beispiele sind in den Tabellen 7.29 und 7.26 zu finden.

Auch wenn die Beispiele evtl. etwas Gegenteiliges suggerieren, ist das hinreichend eindeutige Identifizieren von Erkenntnissen vom Typ T_U und T_C als solche oft schwierig. Denn während bei den anderen Typen aktuelle und als solche erkennbare Umweltwahrnehmungen (oder Eigenwahrnehmungen) einen entscheidenden Teil der Erkenntnis und in der Regel auch Erkenntnisäußerung bilden, muss dies bei diesen beiden Typen nicht so sein. Erörterungen in Hinsicht auf die Abgrenzung zwischen *explain_knowledge* und *explain_finding* sind auch auf S. 331 zu finden.

Achtung: Eine Zuordnung genau eines der hier dargestellten Erkenntnistypen zu einer Äußerung, die potentiell, das heißt nach einer ersten Vermutung, in die Klasse *finding* fällt, muss nicht immer möglich sein. Dies hat zwei Gründe:

1. Es existieren Überschneidungen bei den Anwendungsbereichen einzelner Erkenntnistypen. Man betrachte beispielsweise die zweite Zeile in Tabelle 7.28 (Erkenntnistyp P). Der Entwickler nimmt hier nicht nur das Ereignis wahr, sondern liest und interpretiert allem Anschein nach die Ausgabe. Somit könnte – zumindest in Teilen – auch der Erkenntnistyp D_O passen. Im Rahmen der Anwendung der BS führen solche Abgrenzungsprobleme allerdings nicht zu weiteren Schwierigkeiten. Der Hauptzweck der Erkenntnistypen besteht nämlich – wie bereits erläutert – darin, zu veranschaulichen, welche Äußerungen überhaupt dem Typ *finding* zugerechnet werden sollen. Die Erkenntnistypen werden also als primär charakterisierende Eigenschaften (siehe Abschnitt 6.2) eingesetzt. Hierbei werden zusammengehörige und zusammen in einer Äußerung artikulierte Erkenntnisse auch zusammen mit einem *finding*-Konzept annotiert – es sei denn, sie betreffen tatsächlich völlig unterschiedliche Aspekte/Themen (siehe auch S.267). Erst in vertiefenden Untersuchungen kann es evtl. notwendig werden, sich derartigen Abgrenzungen zuzuwenden.

   Damit die in der vorliegenden Arbeit konkret vorgenommenen Zuordnungen von Erkenntnistypen zu Phänomenen besser nachvollzogen werden können, ist das diesbezüglich angewandte Vorgehen in Abbildung 7.11 skizziert. Es stellt eine, aber nicht die einzig mögliche Operationalisierung der Regeln zum Zwecke der Identifikation einer *explain_finding*-Äußerung dar.

2. Genauso wie die BS sind auch die Erkenntnistypen direkt aus den analysierten Sitzungen extrahiert worden. Somit ist es möglich, dass bei der Analyse weiterer Sitzungen Äußerungen beobachtet werden, die ein Forscher potentiell der Klasse *finding* zurechnen würde, die aber trotzdem keinem der hier erläuterten Erkenntnistypen entsprechen. Sollte dies der Fall sein, muss, nachdem noch einmal sichergestellt worden ist, dass diese Äußerungen nicht evtl. in andere Konzeptklassen fallen, darüber nachgedacht werden, ob neue *finding*-Erkenntnistypen beschrieben werden sollten.

Bevor im nächsten Abschnitt auf spezielle Eigenschaften einzelner *finding*-Konzepte eingegangen wird, sei noch auf zwei allgemeine Eigenschaften von Erkenntnissen/Erkenntnisäußerungen und den Umgang mit ihnen hingewiesen:

- Zum einen können Erkenntnisse, insbesondere solche vom Typ D, in der Regel nur auf Basis von oder in Zusammenhang mit bereits vorhandenem Wissen erlangt werden. So weist beispielsweise Hayes [91, S. xvi] auf die zentrale Rolle von Wissen (im Sinne von Bestandswissen) bei der Problemlösung hin. Dies kann dazu führen, dass innerhalb von Erkenntnisäußerungen auch Bestandswissen mit artikuliert wird. Dieses wird im Rahmen der BS/BKM in der Regel nicht explizit einzeln annotiert. Hierauf wird noch einmal bei den Erläuterungen zur Klasse *knowledge* eingegangen (siehe S. 331f).

- Zum anderen ist eine unaufgeforderte Verbalisierung von Bestandswissen der Klasse *knowledge* und nicht der Klasse *finding* zuzuordnen, obwohl davon ausgegangen werden kann, dass ihr eine Erkenntnis vorausgegangen ist, nämlich die, dass es angebracht oder hilfreich sein könnte, dieses Wissen zu diesem Zeitpunkt weiterzugeben. Dies bedeutet, dass im Rahmen der BS/BKM derartige Erkenntnisse nicht explizit als solche markiert werden.

Außerdem sei noch bemerkt, dass es möglich ist, dass sich im Zusammenhang mit zukünftigen Fragestellungen andere, zu dem hier dargestellten System weitgehend orthogonale Klassifizierungen von Erkenntnissen als geeigneter erweisen können. Dies sollte bei weiterführenden Untersuchungen im Auge behalten werden.

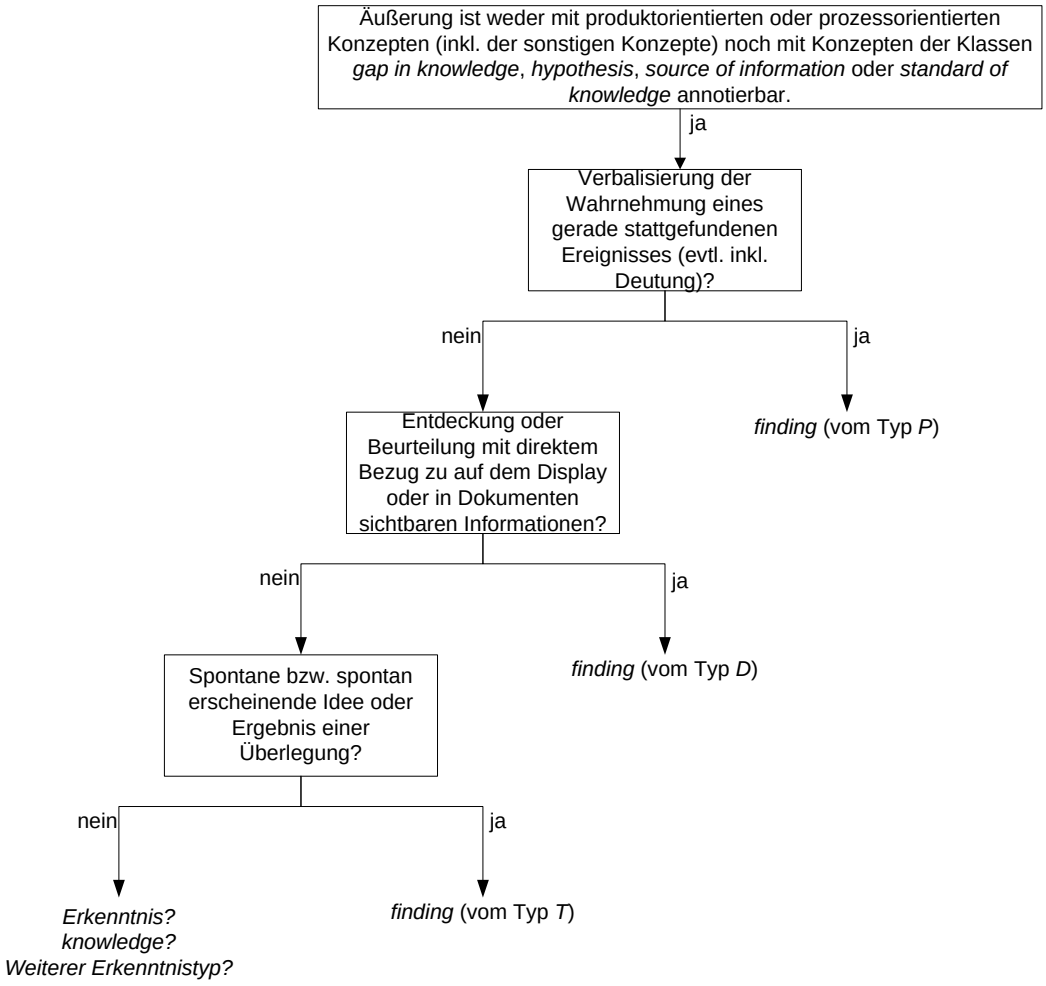


Abb. 7.11: Darstellung des im Rahmen der vorliegenden Arbeit angewendeten Vorgehens bei der Identifizierung von *explain_finding*-Äußerungen. Bei der Beantwortung der dargestellten Fragen wurde nach jeweils passenden Indizien im Videomaterial gesucht, um diese dann in Hinsicht auf ihre „Beweiskraft“ zu bewerten (siehe z.B. Tabelle 7.27 und Tabelle 7.28). Es muss festgehalten werden, dass es sich bei der in der Abbildung wiedergegebenen Vorschrift nicht um eine vordefinierte Regel handelt, sondern um das Ergebnis von kontinuierlich, unter Berücksichtigung diverser Beobachtungen durchgeführten Strukturierungsmaßnahmen. Der mit „Erkenntnis? *knowledge?* Weiterer Erkenntnistyp?“ bezeichnete Fall markiert eine Situation, in der es sich bei der betrachteten Äußerung entweder um eine in die Klasse *knowledge* fallende handelt oder die BS/BKM keine Mittel zur Verfügung stellt, um zu einer angemessenen Kodierung zu kommen. Der zweite Fall kann bedeuten, dass die BS/BKM keine passende Erkenntnisklasse bereit hält. Wird dies angenommen, muss nicht nur über eine neue *finding*-Klasse, sondern auch über eine Restrukturierung des Vorgehens nachgedacht werden. Weitere Anmerkung: Der bei der Formulierung des Identifikationsmerkmals von Erkenntnissen vom Typ *D* verwendete Begriff „direkt“ adressiert den inhaltlichen Zusammenhang zwischen der Äußerung und einer bestimmten Stelle in einem sichtbaren Artefaktteil. Wie dieser Zusammenhang konkret auszusehen hat, um zweifelsfrei als „direkt“ eingeordnet werden zu können, ist nicht festgelegt, bzw. dem Anwender überlassen.

Indizien/ Situationsmuster	Betroffene Erkenntnis-typen	Beispiel
Verbalisierung von Ausgaben innerhalb der Entwicklungsumgebung.	P	Sitzung PR1.1: Während die von *P1* gestartete Eclipse-Suchfunktionalität (Zeichenkettensuche über alle Dateien) noch läuft, verbalisiert der Observer *P2* deren (vorläufige) Ausgabe: „Two matches.“
Beurteilungen der Ergebnisse von Testläufen.	P	Nachdem in Sitzung PR2.1 Änderungen am Kode vorgenommen wurden, ruft das Paar die in Entwicklung befindliche Applikation auf und führt in ihr Operationen durch, mit denen die Korrektheit der Modifikationen überprüft werden soll. Nachdem die Tests wie gewünscht verlaufen sind, quittiert der Driver diese mit „Na also“. Während der Durchführung der Tests äußern sich die beiden Entwickler nicht.
Nachdem ein Sprecher offensichtlich ein Artefakt oder den Teil eines Artefaktes durchgesehen hat, gibt er ein Urteil über dessen Qualität ab.[124]	D_U	In Sitzung PR2.1 inspiziert der Driver eine bereits vorhandene Methode. Diese Durchsicht hatte er explizit angekündigt. Während er mit dem Mauszeiger Zeilen der Methode „abfährt“, äußert er: „Ja, passt.“
Nachdem ein Sprecher ein Artefakt oder den Teil eines Artefaktes durchgesehen hat, äußert er sich zu einer Besonderheit im Kode.	D_U	In Sitzung PR2.1 inspiziert der Driver eine bereits vorhandene Methode. Er fährt dabei die einzelnen Zeilen mit dem Mauszeiger ab. Als er zur Deklaration einer Variablen kommt, stellt er fest: „Das ist ein `IFeatureProxiesTableModel`, kein normales.“
Nachdem ein Entwickler Teile des bereits vorhandenen Kodes nach einer Stelle mit einer bestimmten Eigenschaft durchsucht hat, z.B. nach einer Methode, die seiner Meinung nach noch modifiziert werden muss, verbalisiert er einen Fund.	D_U	In Sitzung PR2.1 durchsucht der Driver bestimmte Teile des Kodes nach einer seiner Meinung nach noch zu bearbeitenden Methode. Da ihm allem Anschein nach der Methodenname sowie der zugehörige Klassenname gerade entfallen sind, sieht er den Kode verschiedener, potentiell infrage kommender Klassen durch. Als er glaubt, die gesuchte Methode gefunden zu haben, artikuliert er dies: „Ahh, da. (..) Genau. (.) Der ist es, mein ich.“ Er markiert die Signatur einer Methode.
		Fortsetzung auf der nächsten Seite.

[124] Äußerungen vom Typ *explain_finding* traten bei den analysierten Sitzungen oft im Rahmen der HCI-Aktivität *verify_sth* auf (siehe Abschnitt 8.1).

		Fortsetzung der Tabelle der vorherigen Seite.
Der Sprecher weist explizit auf einen Fehler im Kode/Artefakt hin, den er offensichtlich vorher nicht als solchen gesehen hat. Das Erkennen wird durch Anmerkungen des Partners befördert.	D_C	In Sitzung PR1.1 versucht der Driver, die Bedeutung einer bereits im Kode vorhandenen `if`-Anweisung zu verstehen, hat aber Probleme damit. Erst als der Observer ihm durch Erklärungsansätze zu helfen versucht, erkennt er, dass der – allem Anschein nach von ihm selbst in einer vorhergehenden Sitzung geschriebene – Kommentar zu einer der Bedingungen falsch ist und somit seinem Versuch, die Bedeutung der Anweisung zu verstehen, entgegengewirkt hat. Er sagt: „(~~) – dann habe ich den Kommentar falsch geschrieben, nur.“
Ein Entwickler bezieht sich mit seiner Äußerung direkt und im Allgemeinen interpretierend auf solche auf dem Bildschirm (neu) dargestellte Informationen, die durch die Arbeitsumgebung an und für sich zur Verfügung gestellt werden. Achtung: An dieser Stelle sind weder Informationen gemeint, die durch die in Bearbeitung befindlichen Artefakte bereitgestellt werden, noch solche, die auf der Wahrnehmung von Ereignissen beruhen.	D_O	Beispiel 1: In Sitzung PR1.1 öffnet der Driver die *Preferences* von Eclipse[125], um die Schriftgröße eines Eclipse-Editors zu verändern. Es erscheint unter anderem ein *TreeView*-Steuerelement, in welchem die veränderbaren Eigenschaften hierarchisch nach Gruppen angeordnet sind. Er öffnet den Knoten *General* und dessen Unterknoten *Editors*, um dann kurz inne zu halten und mit der Maus den Baum auf und abzufahren. In diesem Moment entdeckt der Observer die möglicherweise richtige Gruppe von Eigenschaften als direkten Unterpunkt von *General*: „*Appearence* – ganz oben.“ Beispiel 2: Ebenfalls in Sitzung PR1.1 öffnet *P1* denjenigen Bereich von Eclipse, in welchem Plugins konfiguriert werden können. Er erkennt, dass das von ihm benötigte Plugin (PHPEclipse) nicht mehr installiert ist (es wird nicht in der Liste der installierten Plugins angezeigt)[126]: „Was ist denn mit meinem PHPEclipse passiert, (...) was ich hier mal ursprünglich installiert hatte?“ Achtung: Aus dem Kontext kann entnommen werden, dass es sich um eine rhetorische Frage handelt, dass sie also nicht auf Informationsgewinn zielt.
		Fortsetzung auf der nächsten Seite.

[125] Über die *Preferences* von Eclipse lassen sich alle benutzerspezifischen Eigenschaften der Entwicklungsumgebung einstellen.

[126] Das Plugin PHPEclipse (http://www.phpeclipse.com/ (Abruf: 26.01.2011)) liefert Funktionalitäten, die die Entwicklung in PHP erleichtern.

		Fortsetzung der Tabelle der vorherigen Seite.
Ein Entwickler führt auf eigenes und kundgetanes Bestreben hin eine Änderung im Kode durch, kritisiert diese aber nachfolgend.	T_B	Nachdem der Driver in Sitzung PR1.1 die Frage danach gestellt hat, warum im bestehenden Kode zwei bestimmte Anweisungen nicht zusammen an einer Stelle stehen („Warum steht das hier nicht untereinander?“[127]), verschiebt er – ohne eine Antwort abzuwarten – eine der beiden Anweisungen direkt vor die andere. Danach erfolgt ca. fünf Sekunden lang keine weitere Aktion. Der Driver blickt während dieser Zeit auf den Bildschirm, um seine Änderung dann selbst zu kritisieren: „Naja, das ist natürlich nicht so richtig schön.“[128]
Während des Editierens formuliert ein Entwickler eine Einsicht bez. der in Bearbeitung befindlichen Artefakte, z.B. ein neu erkanntes Problem, welches mit dem gerade geschriebenen Kode zusammenhängt. Eine Äußerung des Partners wirkt dabei als Katalysator.	T_C	In Sitzung PR1.1 fügt *P1* einen phpDocumentor-Kommentar zu einer Funktion hinzu. Als er gerade dabei ist, Angaben zum Rückgabewert der Funktion zu ergänzen (`@return int`), macht sein Partner einen kurzen ablehnenden Einwurf: „[A]hm, mh⌣mh.“ *P1* reagiert hierauf unmittelbar und verbalisiert eine Erkenntnis: „Da haben wir schon (~unser) Problem.“ Aus dem Kontext wird ersichtlich, dass ihm gerade klar geworden ist, dass die Funktion derzeit zwei unterschiedliche Rückgabewerttypen besitzt (`int` und `boolean`). Auch bei dem Einwurf des Partners handelt es sich um eine Erkenntnis. Diese ist mit *disagree_activity* und *explain_finding* zu konzeptualisieren. Sie bezieht sich direkt auf ein Ereignis und zwar auf das Schreiben eines bestimmten Kommentarteils und ist deshalb vom Typ P. Die Erkenntnis des Drivers bezieht sich allem Anschein nach nicht direkt auf die des Observers. Die Erkenntnis des Observers hat eher die Funktion eines Katalysators. Achtung: Bei dieser Zuweisung von Typen handelt es sich nicht um die einzig denkbar sinnvolle. Wie auf S. 257 diskutiert, ist dies aber im Rahmen der BS unerheblich.
		Fortsetzung auf der nächsten Seite.

[127] Natürlich handelt es sich auch bei einer solchen Frage um eine Äußerung, mit der eine gewisse Art von Erkenntnis verbalisiert wird. Diese wird aber im Rahmen der BS über das Konzept *ask_knowledge* adressiert. Solche Äußerungen hatten somit keine Auswirkungen auf die Herleitung der Erkenntnistypen.

[128] Genau genommen gibt es in der beschriebenen Episode keine Äußerung des Drivers, die er im Nachhinein als ungünstig bzw. falsch bewerten kann. Allerdings kann die Tatsache, dass er seine Frage durch eine Handlung direkt selbst beantwortet, als gleichwertig zu einer solchen Äußerung interpretiert werden.

		Fortsetzung der Tabelle der vorherigen Seite.
Der Sprecher zieht eine Schlussfolgerung aus einer zuvor vom Partner hervorgebrachten Äußerung – z.B. aus einem *explain_knowledge* oder *challenge_knowledge*.	T_C	Siehe Zeile 2 in Tabelle 7.48 auf S. 332.
Der Sprecher hebt die Erkenntnis explizit, das heißt mit sprachlichen Mitteln als solche hervor – z.B. durch Phrasen wie „Jetzt wird mir gerade klar, dass...“, „Ich sehe gerade, dass...“ oder „Nee, Moment, ...“ (die beiden ersten wurden in der Form bisher nicht beobachtet) oder durch Ausrufeworte wie z.B. „Ahh“ oder „Ohh“.	P, D, T	In Sitzung PR1.1 äußert sich der Driver zum Namen einer gerade editierten Funktion: „Das ist ja totaler Bullshit, dieser Name, seh' ich gerade“[129]. In derselben Sitzung formuliert er außerdem „Ahh, wir wollen überhaupt gar keinen Traffic reduzieren, natürlich, wir wollen Schreibzugriffe auf (~<**Fremdsystem**>-)Seite sparen. Das ist der ganze Trick. Dass die nicht was updaten, was eh schon da ist“. Der Observer äußert an anderer Stelle „Ach, `account`“.[130] Aus den Kontexten der drei Äußerungen kann abgelesen werden, dass es sich um Erkenntnisse vom Typ D_U, T_C bzw. P handelt.

Tab. 7.27: Indizien und Situationsmuster, anhand derer Erkenntnisse im Sinne der Klasse *finding* identifiziert werden können. Die Indizien verweisen überwiegend auf genau einen Erkenntnistyp. Zu jedem Indiz/Situationsmuster wird ein konkretes Beispiel aus den untersuchten Sitzungen beschrieben. Bei allen Beispielen handelt es sich um Äußerungen vom Typ *explain_finding*.

[129] Details zum Kontext dieser Äußerung sind in Tabelle 7.30 zu finden.

[130] Äußerung einer Erkenntnis darüber, in welchem Paket die Eclipse-Suche ein Vorkommen des Suchbegriffs gefunden hat.

HHI-Kode	**Äußerung**	**Kontext/Kommentar**
P1.propose_step	„Und‿jetzt‿haben‿wir‿gesagt, könnten wir eigentlich mal 'ne kleine Simulation nochmal machen.“	*P1* schlägt vor, das gerade editierte PHP-Skript im Browser aufzurufen. Er setzt seinen Vorschlag unmittelbar in die Tat um.
P1.explain_finding	„Jetzt geben wir Dateien zurück, (.) das heißt, die letzte Änderung (.) war nach der Anfrage.“	**Erkenntnistyp** *P*: *P1* verbalisiert das im Browser erscheinende Ergebnis. Es handelt sich um eine Liste von Strings, die hexadezimale Werte enthalten. Jeder String repräsentiert einen Rückgabewert. Mit seiner Aussage interpretiert er das Ergebnis als eines mit einer bestimmten Eigenschaft.
P1.propose_step	„Halt, warte mal. Nach der letzten Anfrage?“	Nach wenigen Sekunden Pause zieht *P1* seine Erkenntnis, das heißt genau genommen nur den Interpretationsteil, in Zweifel. Der Frageteil der Äußerung ist hierbei als rhetorisch zu werten. Die Aussage lässt sich als „Lass uns mal kurz innehalten, um zu überlegen ob meine Erkenntnis stimmt“ verstehen. Siehe hierzu auch S. 196.[131]
P1.ask_knowledge	„Was übergeben wir denn eigentlich als `timestamp [last_request]`?“	Nachdem *P1* vom Browser zurück in den Eclipse-Editor gewechselt hat, in dem eine `if`-Anweisung markiert ist, die regelt, was in bestimmten Fällen zurückgegeben wird, fragt er nach dem Inhalt einer Variable, die dem Skript übergeben und in der Bedingung der Anweisung verwendet wird. Es kann nicht entschieden werden, ob es ihm um den aktuellen Wert der Variablen oder ihre grundsätzliche Bedeutung geht.
P2.explain_knowledge	„Na da übergibt uns (~<**Fremdsystem**>[132]) (.) seinen *Timestamp*, (..) wann er zuletzt die (~`FriendslistId`) über(~).“	*P2* erklärt ohne zu zögern die Bedeutung der Variablen.
		Fortsetzung auf der nächsten Seite.

[131] Hier ist zusätzlich eine Annotation von *explain_standard of knowledge* angebracht (siehe Abschnitt 7.6.3).
[132] Siehe Abschnitt 4.2.2.

		Fortsetzung der Tabelle der vorherigen Seite.
P1.agree_knowledge+ P1.ask_knowledge	„Genau, aber was macht denn unser Skript gerade?“	*P1* stimmt *P2* unmittelbar zu. Die von *P2* geäußerte Information schien ihm aber schon bekannt gewesen zu sein. Er stellt eine weitere Frage und macht so deutlich, dass er etwas über das Verhalten des Skriptes in Bezug auf die aktuellen Übergabewerte wissen will.
P1.explain_finding	„Da hab’ ich doch einfach irgendeinen beliebigen `Timestamp` reingeschrieben. (, , ,) Da ist es doch eigentlich schon verwunderlich, dass es mit -1000, +1000 geklappt hat gerade.“	**Erkenntnistyp** T_U: Nach ca. vier Sekunden, in denen auch *P2* schweigt, äußert *P1* die Einsicht, dass er den Übergabeparameter, der an das Skript übergeben wird, nicht gezielt ausgewählt hat. Er wundert sich, dass das Skript trotzdem das erwartete Ergebnis liefert. Diese Erkenntnis wird zu einem Zeitpunkt verbalisiert, zu dem der Browser und somit die aufgerufene URL mit dem angesprochenen Übergabeparameter nicht sichtbar ist. Erst während *P1* spricht, wechselt er zum Browser und markiert die entsprechende Stelle in der URL. Diese URL wurde vom Paar via „Reload“ bei allen zurückliegenden Tests verwendet. Es ist nicht klar, ob die Erkenntnis genau zu diesem Zeitpunkt oder schon kurz vor dem letzten *propose_step* entstanden ist. Festgehalten werden kann nur, dass *P1* sie jetzt verbalisiert. Mit den Zahlen „-1000“ und „+1000“ bezieht sich *P1* auf einen vormals „hart kodierten“ Wert im Skript – der aber vor dem letzten Test wieder entfernt worden ist. Randbemerkung: Bez. der Werte irrt sich *P1*, denn im Skript wurden „-100“ bzw. „+100“ verwendet.

Tab. 7.28: Beispielepisode aus Sitzung PR1.1, in der mehrere Erkenntnisse verbalisiert werden. Sie beginnt, nachdem eine Reihe von Änderungen im Kode vorgenommen wurden. Im Rahmen dieser Modifikationen war das in Bearbeitung befindliche PHP-Skript dahingehend berichtigt worden, dass es nur solche Datenbankeinträge zurück gibt, die in einem bestimmten Zeitintervall eine Änderung erfahren haben. Zu Testzwecken war hierbei eine in der späteren Endfassung via URL zu übergebende Vergleichszeit vorläufig „hart kodiert“ worden (durch einen anderen dem Skript zu übergebenden um 100 erhöhten Wert). Diese Änderung wurde allerdings vor dem hier beschriebenen Aufruf wieder entfernt (hierauf bezieht sich das erste hier wiedergegebene *propose_step*).

Beispiel	Typ
Nachdem in Sitzung PR1.1 Änderungen in einem PHP-Skript vorgenommen worden sind, wird dieses zu Testzwecken im Browser aufgerufen. Auf der erscheinenden Rückgabeseite sind unter anderem zwei Datumsangaben (Ergebnis von vorher im Skript eingefügten Testausgaben) zu sehen: „20.06.2007 15:52:27“ und „01.01.1970 01:00:00“. Diese sind mit „Last request“ bzw. „Last change“ beschriftet. Der Driver kommentiert das Ergebnis: „Und hier haben wir auch ‚Last Change‘ (.) 1.1.1970, (~) (..) ein Uhr. Was aber auch sehr interessant ist, weil das wäre auch mein nächster Test gewesen, ob `date` eigentlich [a]hm irgendwie 'nen Check macht, ob's ein `int` ist, oder ob der auch (.) ob der auch für `false` den 1.1. zurück gibt. Denn das ist ja genau (.) die Befürchtung die ich hatte. Das es (~) Null ist. Aber das spielt eigentlich keine Rolle. Null ist raus, und “null“ <*Englisch ausgesprochen*> und alles.“ Der Driver erkennt also, dass ein von ihm vermutetes Problem nicht existiert.	*P*
Sitzung PR1.1: Nachdem *P1* Änderungen an einem PHP-Skript vorgenommen hat, ruft er dieses (erneut) im Browser auf. Der Observer quittiert das Ergebnis des Aufrufs mit einem kurzen „Phantastisch!“ und gibt somit zum Ausdruck, dass er es als richtig bewertet.	*P*
In Sitzung PR2.1: Nach Änderungen im Code startet der Driver *P2* die gerade in Entwicklung befindliche Applikation testweise. Während des Startvorgangs kommt es zu einem Driverwechsel. Nachdem dieser abgeschlossen ist, navigiert *P1* zu einem bestimmten Ausgabefenster in der Applikation. Er quittiert die dort erscheinenden Informationen mit einem „OK“ und gibt somit zum Ausdruck, dass er die Funktionsweise als richtig bewertet.	*P*
In Sitzung PR1.1 sieht sich der Driver die Signatur einer Funktion an. Hierzu hat er den Cursor auf die entsprechende Zeile gestellt. Nach ca. drei Sekunden, in denen nichts weiter passiert, erkennt er: „Das ist ja totaler Bullshit, dieser Name, sehe ich gerade“. Er hält mit dieser Aussage fest, dass der Name der Funktion deren Zweck nicht korrekt beschreibt.	D_U
In Sitzung PR2.1 sehen Driver und Observer zusammen den Kode einer Klasse durch. Nach ca. sieben Sekunden, in denen beide Entwickler auf das Display gesehen und nichts gesagt haben, äußert sich P2: „Ahh, das wird ja erst hier unten verwendet und (!...!)“	D_U
Während der Driver in Sitzung PR1.1 gerade dabei ist, einer Variable ein Objekt zuzuweisen, wird er gefragt, warum diese Variable den Namen `ids` trägt: „Wieso `ids`? Das ist doch ein `timestamp`.“ Hierdurch erkennt der Driver, dass er vergessen hat, die Variable umzunennen: „Ach so, ich bin ja hier – genau – ich hab' die Variable noch gar nicht umgenannt.“	T_C
PR1.1: Nachdem sich die Partner eigentlich darüber geeinigt haben, wann und in welcher Weise das von ihnen weiterzuentwickelnde Skript (zeitabhängige) Daten an ein externes System übermitteln soll und der Observer einen Test auf dem Zielsystem vorgeschlagen hat, verweist der Driver auf ein potentielles Problem: „Ich seh' immer noch ein bisschen ein Problem mit der Uhrzeit. (.) Wenn man das jetzt irgendwie simulieren will, dann bräuchte man die gleiche Uhrzeit wie <**unser**> System, damit das vernünftig funktioniert.“	T_C

Tab. 7.29: Weitere Beispiele für Erkenntnisse vom Typ *explain_finding*.

Identifizierte *finding*-Konzepte und ihre speziellen Eigenschaften

Für die Klasse *finding* konnten die Konzepte *explain_finding*, *agree_finding*, *disagree_finding*, *amend_finding* und *challenge_finding* identifiziert werden. Grob wurden sie bereits in Abbildung 7.3 beschrieben. Details sind im Folgenden dargestellt.

1. **Erkenntnistypen:** In den einführenden Erläuterungen zur Konzeptklasse *finding* wurde eine Klassifizierung beschrieben, mit deren Hilfe Erkenntnisäußerungen vom Typ *finding* identifiziert werden können bzw. die theoretische Sensibilität (siehe S. 59f und S. 72f) in Bezug auf solche Phänomene verbessert werden kann. Im Fokus stand hierbei das Konzept *explain_finding*, da Phänomene dieses Typs *finding*-Episoden einleiten. Inwieweit sich die Erkenntnistypen auch im Zusammenhang mit den anderen *finding*-Konzepten einsetzen lassen, wird weiterführenden Untersuchungen überlassen.

2. **Bewertung der eigenen Erkenntnis:** Wie bereits in den einführenden Erläuterungen ausgeführt, handelt es sich bei Äußerungen vom Typ *finding* um solche, in denen der Sprecher keinen nennenswerten Zweifel an der Richtigkeit der von ihm artikulierten Erkenntnisse formuliert. Dies hängt mit der Konstruktion der BS/BKM zusammen, da Erkenntnisse in Form von Vermutungen als zur Klasse *hypothesis* gehörend definiert wurden. Details hierzu werden im Rahmen der Erläuterungen von *hypothesis* erörtert (siehe S. 292).

3. **Zusammenfassung von Äußerungen**: Werden mehrere Erkenntnisse direkt hintereinander bzw. innerhalb einer Äußerung artikuliert, sollten sie auch zusammen mit nur einem *explain_finding* (bzw. *amend_finding* oder auch *challenge_finding*) annotiert werden. Von dieser Vorgabe muss nur dann abgerückt werden, wenn zwei (oder mehrere) offensichtlich in keiner Weise direkt inhaltlich miteinander in Beziehung stehende Erkenntnisse zusammen formuliert werden. Sollte dies der Fall sein, muss die Äußerung geteilt und jeder Teil separat konzeptualisiert werden.

   Achtung: In weiterführenden Untersuchungen kann es wichtig werden von solchen Zusammenfassungen abzusehen – z.B. dann, wenn Analysen anstehen, in denen weitere Arten von Erkenntnissen auseinandergehalten werden müssen.

4. **Wiederholungen von Erkenntnisäußerungen**: Wird eine Erkenntnis wiederholt, das heißt lediglich ein weiteres mal geäußert bzw. ohne inhaltliche Ergänzungen paraphrasiert, sind folgende Fälle zu unterscheiden[133]:

[133] Natürlich ist nicht immer wirklich eindeutig zu erkennen, ob es sich bei zwei Äußerungen um die Verbalisierung von ein und derselben Erkenntnis handelt. So gibt es hier wie auch an anderen Stellen einen nicht zu eliminierenden Interpretationsspielraum.

- Sind beide Äußerungen vom selben Sprecher und folgen zeitnah aufeinander, ist auch die Wiederholung mit *explain_finding* zu annotieren (siehe auch Abschnitt 9.3.7).
- Sind beide Äußerungen nicht vom selben Sprecher und folgen zeitnah aufeinander, muss geprüft werden, ob es sich bei der zweiten Äußerung evtl. um ein *explain_standard of knowledge* handelt, der Sprecher also nur anzeigen möchte, dass er die Erkenntnis nachvollziehen kann, um eine Zustimmung (*agree_finding*) oder sogar tatsächlich um eine eigene von der Äußerung des Partners unabhängige Erkenntnis (z.B. wenn beide Entwickler zur selben Zeit denselben Einfall hatten). Auf Details der Abgrenzung zwischen den Konzepten *explain_standard of knowledge* und *agree_finding* wird auf S. 309 eingegangen.
- Folgen beide Äußerungen nicht zeitnah aufeinander und die Wiederholung ist nicht sprachlich als neue Erkenntnis hervorgehoben, muss in der Regel von einem Lerneffekt ausgegangen werden. Die zweite Äußerung ist dann mit *explain_knowledge* zu kodieren.
- Folgen beide Äußerungen nicht zeitnah aufeinander und die Wiederholung ist sprachlich als neue Erkenntnis hervorgehoben, ist die zweite Äußerung mit *explain_finding* zu kodieren.

5. **Beurteilungen von Artefakten:** Wie bereits erläutert werden im Allgemeinen auch Beurteilungen von Artefakten oder Artefaktteilen (z.B. Beurteilungen von vorhandenem Quellcode) der Klasse *finding* zugerechnet (siehe S. 253). Solche Beurteilungen können sehr kurz formuliert sein. So äußert der Driver in Sitzung PR2.1 beispielsweise nur ein kurzes „Ja!“, um mitzuteilen, dass er den gerade durchgesehenen und dabei mit dem Mauszeiger „abgefahrenen“ Kode für richtig hält.

6. **„Live“ geäußerte Denkprozesse:** Es kommt vor, dass Entwickler nicht nur das Ergebnis eines kognitiven Prozesses verbalisieren, z.B. das Ergebnis logischen Denkens, sondern große Teile des Prozesses quasi live mitsprechen. Z.B.:

   > „Erste Änderung Null (!...!) (..) Das Problem is' halt, wenn er den aktuellen `TimeStamp` zurückgeben würde (!...!) (..) (Hm.) (.) Wir haben sowieso 'n Problem, dass wir die Uhren synchron halten. (.) Und gerade wenn man jetzt den `TimeStamp` macht, dann könnte es echt sein, dass man um ein paar Sekunden, Millisekunden (!...!) (.....) Ne, Sekunden. Das es um ein, zwei Sekunden hinterher hinkt und (.) dass da unnötig was verschickt wird.“ (PR1.1)[134]

[134] Die Äußerung wurde trotz der Pausen als eine einzige wahrgenommen bzw. interpretiert, da sie sich über die Pausen fortsetzt – im Gegensatz zu den beiden *explain_finding*- bzw. *amend_finding*-Äußerungen, die die in Tabelle 7.26 dargestellte Episode einleiten.

Im Rahmen der BS/BKM werden solche Äußerungen, wenn nicht Teiläußerungen eindeutig in andere Klassen wie z.B. *standard of knowledge* fallen, mit einem *explain_finding* annotiert. Inwieweit dies auch für mit übermitteltes Bestandswissen gilt, wird erst in Rahmen der Erörterungen zu den *knowledge*-Konzepten auf S. 331 diskutiert.

7. **Eigene Erkenntnisse widerrufen und durch neue ersetzen:** Genauso wie schon bei den Erläuterungen zu den *design*-Konzepten ausgeführt (siehe S.175) wurde auch in Hinsicht auf Erkenntnisäußerungen beobachtet, dass es vorkommen kann, dass diese kurze Zeit später vom Sprecher selbst widerrufen und durch andere ersetzt werden. Solche Phänomene sollten mittels *challenge_finding* konzeptualisiert werden. Hierbei ist es egal, ob es sich bei der Änderung zweifelsfrei um die Formulierung einer neuen Erkenntnis handelt oder evtl. „nur“ um die Artikulierung von Bestandswissen.[135] Analoges gilt für alle *challenge*-Konzepte in der Menge der UK. Weitere Anmerkungen hierzu finden sich in Abschnitt 9.3.9.

8. **Präzisierungen von bzw. Ergänzungen zu Erkenntnissen:** Ob mit einer Äußerung ein vorher artikuliertes *finding* konkretisiert/präzisiert bzw. ergänzt wird (*amend_finding*) oder eine neue, von der ersten Äußerung weitgehend unabhängige Erkenntnis formuliert wird (*explain_finding*), ist nicht in jeder Situation offensichtlich. Hier soll gelten:

   (a) Erläutert oder ergänzt ein Sprecher zeitnah eine von ihm selbst zuvor geäußerte Erkenntnis, so ist *amend_finding* zu annotieren. Hierbei ist nur entscheidend, dass ein direkter inhaltlicher Zusammenhang identifiziert werden kann. Anders ausgedrückt: Die zweite, zeitnah formulierte Äußerung muss denselben Aspekt bzw. dasselbe Thema wie die in der ersten Äußerung formulierte Erkenntnis adressieren. Unerheblich ist hingegen, ob in der zweiten Äußerung wirklich eine weitere, mit der ersten zusammenhängende aber erst danach erlangte Erkenntnis, eine so genannte Folgeerkenntnis, geäußert oder „nur“ die ursprüngliche Einsicht ausformuliert wird. Diese Regel trägt der Tatsache Rechnung, dass oft nicht ermittelt werden kann, welcher der beiden Fälle gerade vorliegt. Ausnahmen bilden hier nur solche Folgeäußerungen, die in andere Objektklassen wie z.B. *design* oder *standard of knowledge* fallen. Sie werden mit den entsprechenden Konzepten kodiert, wobei hierbei die Verwendung der Klasse *knowledge* besonderer Erläuterung bedarf (siehe S. 335). Beispiele sind in Tabelle 7.30 und Tabelle 7.34 zu finden.

   (b) Ergänzt ein Sprecher eine Erkenntnisäußerung des Partners, so ist auch eine solche Äußerung in der Regel mit *amend_finding* zu annotieren.

[135] Zumindest muss der Sprecher die Erkenntnis erlangt haben, dass die von ihm zuerst verbalisierte Einsicht nicht (vollständig) richtig ist.

Prinzipiell gelten hierbei dieselben Regeln wie im obigen Fall. Allerdings ist die Abgrenzung zu *explain_knowledge* schwieriger, da es passieren kann, dass der zweite Sprecher innerhalb einer solchen Erkenntnisepisode gar keine eigene Erkenntnis hat bzw. formuliert. Auf dieses Problem wird im Zusammenhang mit der Klasse *knowledge* näher eingegangen (siehe S. 335). Ein Beispiel für eine Ergänzung zu einer Erkenntnis durch den Partner ist in Tabelle 7.31 zu finden.

9. **Begründung von Vorschlägen:** Das Phänomen, dass Begründungen von Vorschlägen, z.B. zum Design oder bez. des Prozesses Erkenntnisse im Sinne der Klasse *finding* darstellen können, wird erst im Rahmen der Erläuterungen zur Klasse *knowledge* diskutiert (siehe S. 319).

10. **Begründung von Erkenntnissen:** Erkenntnisäußerungen und ihre Begründungen sollten, wenn sie zeitnah direkt hintereinander geäußert werden, ohne dass ein als Unterbrechung wirksamer Einwurf des Partners stattfindet, zusammen mittels *eines explain_finding*s (oder *amend_finding*s oder *challenge_finding*s) kodiert werden (siehe z.B. das erste *amend_finding* in Tabelle 7.30). Hiermit wird der Tatsache Rechnung getragen, dass Begründungen von Einsichten oft Teil der Einsicht selbst sind. Es wird davon abstrahiert, ob im Rahmen der Verbalisierung Bestandswissen mit artikuliert wird – zumindest, wenn dieses nicht explizit z.B. durch „Meines Wissens nach...“ als solches hervorgehoben wird (siehe auch Matrix 7.49 (S. 336) bzw. Matrix 7.50 (S. 337)). Derartige explizite Hervorhebungen wurden allerdings bisher nicht beobachtet. Aus diesem Grund kann im Rahmen der hier vorgestellten Untersuchungen nicht näher auf diese Phänomene eingegangen werden. Weiterführende Analysen müssen sich im Detail darum kümmern, in welcher Weise (explizit hervorgehobenes) Bestandswissen im Kontext von Erkenntnisbegründungen markiert werden sollte.

11. **Widersprüche:** Erkenntnissen wird, wie auch anderen Äußerungen, auf zweierlei Weise widersprochen:

    (a) ***disagree_finding*:** Der Widerspruch erfolgt, ohne dass dabei über die pure Ablehnung hinausgehende kontroverse Informationen geäußert werden. So weist der Driver *P2* in Sitzung PR2.1 eine Erklärung des Partners – ein *explain_finding* – mit folgenden Worten zurück:

    > „(~Das◡hat◡damit◡nichts◡zu◡tun), dass die Taskleiste nicht geht.“

    Der Sprecher *P2* bezieht sich hier auf die Taskleiste des Betriebssystems Windows XP. Sein Widerspruch basiert allerdings auf einem Missverständnis. Der Observer *P1* hatte sich mit derjenigen Äußerung, auf die sich *P2* bezieht („Der muss 'n *Rebuild* machen – also der *buildet* gerad'“) gar nicht auf das zuvor von *P2* angesprochene Verhalten

HHI-Kode	Äußerung	Kontext/Kommentar
P1.explain_finding	„Das ist ja totaler Bullshit, dieser Name, seh' ich gerade“	Erkenntnistyp D_U: Nachdem *P1* einen Kommentar zu der bereits vorhandenen Funktion `registerFriendsLastChange` hinzugefügt hat, bewertet er ihren Namen.
P2.ask_knowledge[136]	„Wieso?“	Der Observer möchte die Aussage begründet haben. Seine Frage schließt sich der Äußerung des Partners unmittelbar an.
P1.propose_design	„Entweder `RegisterFriendsChange` (!...!)“	*P1* beantwortet die Frage des Partners nicht direkt, sondern äußert – dem Partner fast ins Wort fallend – einen Vorschlag, wie die Funktion besser zu benennen wäre. Hierbei scheint er zwei Alternativen anbieten zu wollen. Er stoppt aber nach der Nennung der ersten. Es ist möglich, dass die Äußerung in Zusammenhang mit einer Erkenntnis (vielleicht der obigen) steht. Dies ist aber im Rahmen einer BS-Kodierung unerheblich. Hier ist vorerst nur die Illokution „Vorschlagen“ zu konzeptualisieren.
P1.amend_finding	„Ah (.), weißt D[u], (~) `FriendsLastChange` is' ja totaler Blödsinn‿irgendwie‿so, weil wir registrieren ja irgendwie‿so, dass dieses Event stattgefunden hat.“	Nach ca. drei Sekunden Stille konkretisiert *P1* seine zuvor geäußerte Erkenntnisäußerung. Der Ausruf „Ah“ deutet darauf hin, dass *P1* eine weitere Erkenntnis hat. Da diese inhaltlich direkt mit der ursprünglichen zusammenhängt, ist die Äußerung mit *amend_finding* zu kodieren.[137]
P1.challenge_design	„Was hältst Du von (..) `UpdateFriendsLastChangeTime`?“	Ohne wirklich eine Sprechpause einzulegen, macht *P1* einen weiteren Vorschlag, wie die Funktion benannt werden sollte. Mit diesem spricht er sich implizit gegen seinen vorher geäußerten aus. Während er spricht, fängt er an, die Funktion entsprechend umzubenennen.
P2.agree_design	„Wunderbar.“	*P2* stimmt dem Vorschlag direkt zu.
		Fortsetzung auf der nächsten Seite.

[136] In der BS/BKM ist kein Konzept *ask_finding* vorgesehen (siehe S. 283).

[137] Die Episode erlaubt es allerdings nicht, zweifelsfrei auszuschließen, dass sich der Ausruf auf die Frage des Partners bezieht – im Sinne von „Ah, dass verstehst Du nicht“. Allerdings müsste auch in diesem Fall *amend_finding* annotiert werden.

		Fortsetzung der Tabelle der vorherigen Seite.
P1.amend_finding	„Das‿ist‿nämlich‿ das‿Schöne‿daran‿ ist (..), dass es sich hier (~) überall unterscheidet – hier haben wir ihn und hier updaten wir ihn. Und das ist doch eigentlich clean.“	Obwohl der Partner keine Rückfrage gestellt hat, erklärt der Driver seine Erkenntnis bzw. Gedanken, die im Zusammenhang mit der Erkenntnis stehen, näher. Dies geschieht unmittelbar. Hierbei zeigt er mit dem Mauszeiger auf unterschiedliche Stellen im Kode. Obwohl nicht zweifelsfrei nachweisbar ist, dass es sich bei der Äußerung nicht um eine neue, gerade gewonnene Erkenntnis handelt, sprechen zwei Aspekte für eine Kodierung mit *amend_finding*: 1. Es gibt einen inhaltlichen Bezug zur zuvor gemachten Äußerung; 2. Die Äußerung hat eher den Charakter einer nachgelagerten Erklärung als den einer spontanen Erkenntnis.
P2.agree_finding	„Gut.“	*P2* stimmt der Erkenntnis zu.

Tab. 7.30: Beispielepisode aus Sitzung PR1.1 (14:44:21–14:44:52), in der ein Entwickler eine von ihm selbst geäußerte Erkenntnis ergänzt bzw. näher ausführt. Die Episode verdeutlicht darüber hinaus, in welcher Weise zwischen *explain_finding*- und *propose_design*-Phänomenen zu unterscheiden ist.

HHI-Kode	Äußerung	Kontext/Kommentar
P1.propose_step	„Ähh, ich würd' sagen, wir kopieren mal alles knallhart hier."	Im nächsten Schritt will das Paar die Funktion `getFriendsLastChange` implementieren. Diese besitzt bisher einen leeren Funktionsrumpf. Als erste Aktion schlägt der Driver vor, den Rumpf der bereits vorhandenen Funktion `getFriendsIDs` zu übernehmen, das heißt, in den leeren Rumpf von `getFriendsLastChange` zu kopieren. Während er spricht, scrollt er zum Anfang der zu kopierenden Funktion.
P2.propose_design + *P2.explain_finding*	„Das SQL-(~) brauchen wir nicht mehr. Wir haben ja nur 'ne 'ne `Memcache`-Abfrage."	Der zu kopierende Kode beginnt mit SQL-Anweisungen. Der Observer bemerkt (ca. eine Sekunde nach dem Vorschlag von *P1*), dass man diese Anweisungen im Rahmen der zu implementierenden Methode nicht benötigt. Er begründet seine Aussage. Die Begründung bezieht sich direkt auf den gerade angezeigten Bildschirminhalt und stellt eine diesbezügliche Erkenntnis dar (Typ D_C). Während er spricht, beginnt *P1* den Rumpf zu kopieren.
P1.agree_design/ *P1.agree_finding*	„Genau"	*P1* bestätigt die Aussagen von *P2*, deren Inhalte für ihn allem Anschein nach keine neuen Erkenntnisse darstellen, unmittelbar.[138]
P1.propose_step	„Das lösch' ich auch gleich wieder raus."	Knapp drei Sekunden später – während er noch beim Kopieren ist – kündigt *P1* an, die SQL-Anweisungen gleich löschen zu wollen.
P1.ask_knowledge	„Was sind 'n das für Tabs[139] hier?"	Unmittelbar danach, aber immer noch während er damit beschäftigt ist, den Rumpf zu kopieren, stellt der Driver eine Frage zur Formatierung des vorhandenen Kodes.[140]
		Fortsetzung auf der nächsten Seite.

[138] Es kann nicht sicher festgemacht werden, ob sich *P1* auf den Vorschlag oder auf die zugehörige Erklärung des Partners bezieht. Es wird davon ausgegangen, dass der Sprecher beiden Teilen zusammen zustimmt.

[139] Die Bezeichnung Tab steht hier für Tabulator.

[140] Hier kann nicht eindeutig ermittelt werden, ob es dem Sprecher wirklich darum geht, den Partner zu befragen. Evtl. handelt es sich auch „nur" um eine Erkenntnisäußerung (siehe hierzu auch S. 339).

		Fortsetzung der Tabelle der vorherigen Seite.
P1.explain_finding	„If not Dollar *id*[141]!?“	Erkenntnistyp D_U: Nachdem *P1* den Kopiervorgang nach weiteren zehn Sekunden abgeschlossen hat, liest er die dritte Zeile des kopierten Skriptes vor (`if (!$id)`). Die Betonung lässt darauf schließen, dass er davon ausgeht, eine Auffälligkeit entdeckt zu haben. Allem Anschein nach handelt es sich nicht um eine Frage.
P2.mumble_sth	„Return Array, ne, (~~), (Null).“	Der Observer scheint sich auf den Schleifenrumpf zu beziehen (`return array();`). Seine Äußerung ist in wesentlichen Teilen unverständlich. Dies liegt vor allem daran, dass ihm der Driver ins Wort fällt.
P1.amend_finding	„(Schwachsinn).“	*P1* fällt seinem Partner ins Wort und beurteilt die `if`-Anweisung.
P2.amend_finding	„(~) Dollar *id* sowieso (~)“	Nach ca. drei Sekunden Stille ergänzt *P2* die Äußerungen des Partners. Diese Ergänzung ist aufgrund akustischer Probleme nur in Teilen zu verstehen. Die Reaktion des Drivers (siehe nachfolgenden Eintrag) deutet allerdings darauf hin, das dieser im Stande war, die Äußerung zu deuten. Diese Reaktion, sowie der erkennbare Bezug auf die Variable $id und die Verwendung des Adverbs *sowieso* sollen an dieser Stelle ausreichend sein, um von einem *amend_finding* zu sprechen.
P1.agree_finding	„Ja, eben.“	*P1* stimmt seinem Partner unmittelbar nachfolgend zu.

Tab. 7.31: Beispielepisode aus Sitzung PR1.1 (14:23:59–14:24:38), in der ein Entwickler eine vom Partner geäußerte Erkenntnis ergänzt bzw. näher ausführt. Das Beispiel verdeutlicht darüber hinaus, wie auch akustisch nicht vollständig zu verstehende Äußerungen im Rahmen einer Analyse gedeutet werden können.

[141] Hier wird die Variable `$id` referenziert (*id* steht für den englischen Begriff *identifier*). Der Bezeichner wird von *P1*, wie später auch von *P2*, in englischer Sprache buchstabiert (\ˈī-ˈdē\).

der Taskleiste bezogen. Vielmehr war *P1*, wie aus nachfolgenden Äußerungen zu erkennen ist, komplett entgangen, dass der Driver diesbezügliche Probleme formuliert hatte.

Ob es sich bei der Reaktion von *P2* um die Äußerung einer Erkenntnis oder die Formulierung von Bestandswissen handelt, kann nicht wirklich entschieden werden. Dies hat aber keine Auswirkung auf die Kodierung. Konform mit den zentralen Prinzipien (siehe Abschnitt 7.2) ist die Äußerung als *disagree_finding* zu konzeptualisieren.[142] Sie würde den Prinzipien folgend sogar dann nicht mit *explain_knowledge* annotiert werden (auch nicht zusätzlich), wenn nachgewiesen werden könnte, dass es sich tatsächlich um die Verbalisierung von Bestandswissen handelt.

Achtung: Genau genommen widerspricht der Driver nicht der Erkenntnis des Partners, sondern der von ihm vermuteten Interpretation der Beobachtung durch den Partner. Er geht davon aus, dass sich die Äußerung des Observers auf die Fehlfunktion der Taskleiste bezieht, also im Sinne von

> „Wegen des von mir gerade entdeckten *Rebuilds* geht die Taskleiste nicht“

intendiert ist. Die BS/BKM unterscheidet allerdings nicht zwischen der Interpretation eines Ereignisses und der reinen Verbalisierung des Erkennens eines solchen. Zumindest dann nicht, wenn die Interpretation nicht als reine Äußerung von Bestandswissen identifiziert werden kann.

Es bleibt zu bemerken, dass solche Widersprüche anders als der gerade betrachtete sehr kurz sein können (siehe auch die Erläuterungen im Zusammenhang mit der Klasse *design* auf S. 176).

(b) ***challenge_finding*:** Im Rahmen des Widerspruchs artikuliert der Sprecher weitere, widersprechende Informationen. Details hierzu werden im nächsten Unterpunkt erläutert.

12. **Kontroverse Erkenntnisse:** Es konnte beobachtet werden, dass Entwickler einer Erkenntnis des Partners widersprechen und ihr eine eigene entgegensetzen. Solche Äußerungen sind mit *challenge_finding* zu kodieren, wobei es sich bei der übermittelten Einsicht auch „nur“ um eine solche handeln kann, die darauf zielt, zu begründen, warum die attackierte Erkenntnis nicht richtig ist. Es muss sich also nicht um eine Äußerungen handeln, in der eine vermeintlich richtige oder passendere Einsicht erläutert wird. Ein Beispiel hierfür ist in Tabelle 7.32 zu finden. Auf derartige Phänomene wird auch noch einmal im Rahmen der Erörterung von *challenge_knowledge*

[142] Dass es sich um ein Missverständnis handelt, wird im Rahmen einer BS/BKM-Kodierung nicht sichtbar.

HHI-Kode	Äußerung	Kontext/Kommentar
P2.explain_finding	„Ne, mit der `lib` war nicht richtig – da musste schon noch ein ‚Punkt Punkt' hin."	Driver und Observer sind dabei die Datei *build.properties* bzw. die Fehlermeldungen des *Build*-Prozesses[a] in der Konsole von Eclipse durchzusehen.[b] Nach ca. 10 Sekunden äußert der Observer die Erkenntnis, dass in der Datei *build.properties* die Zeile `lib.dir=$basedir/lib` für das aktuelle Problem verantwortlich ist und zu `lib.dir=$basedir/../lib` geändert werden muss. Dabei zeigt er (wahrscheinlich[c]) auf die entsprechende Stelle in der Datei.
P1.challenge_finding	„Das kann (~doch nicht sein). Die `lib` liegt doch auch hier."	Der Driver widerspricht seinem Partner unmittelbar und äußert seine Ansicht zu `lib`. Obwohl er nicht glaubt, dass die Erkenntnis des Partners die Ursache des Fehlers ist, ändert er die entsprechende Zeile.[d]
P2.agree_finding	„Ja, stimmt. Dann war's nicht das Problem."	Der Observer stimmt der Ansicht des Drivers zu.

[a] Die Entwickler verwenden Apache Ant (http://ant.apache.org/ (Abruf: 26.01.2011)), um ihr Programm zu übersetzen, zu „deployen" etc.

[b] Es ist sowohl die Konsole mit der Fehlermeldung wie auch die Datei *build.properties* geöffnet bzw. sichtbar. Es kann nicht eindeutig ermittelt werden, welche Stellen der Observer inspiziert.

[c] Aus dem Blickwinkel der Kamera kann nicht zweifelsfrei ermittelt werden, wohin eine Person genau zeigt.

[d] Hier zeigt sich eine wichtige Beschränkung der BS. Implizites Zustimmen, z.B. durch direktes Umsetzen eines Vorschlags, wird in der Regel nicht erfasst.

Tab. 7.32: Episode aus Sitzung ST1.1 (16:15:39–16:15:48), in der ein Entwickler einer Einsicht des Partners widerspricht, indem er eine eigene kontroverse Meinung äußert. Inwieweit es sich bei diesem Widerspruch wirklich um die Verbalisierung einer neuen Erkenntnis oder „nur" um die von bereits länger vorhandenem Wissen handelt, kann an dieser Stelle nicht ermittelt werden. Dies spielt aber im Rahmen von BS/BKM-Kodierungen keine Rolle. Die geäußerte Kritik fällt in die Klasse *finding*. Hierauf wird im Zusammenhang mit Erläuterungen zur Klasse *knowledge* (ab S. 335) näher eingegangen.

auf S. 325 eingegangen. Detaillierte Analysen zu *challenge_finding* müssen nachfolgenden Untersuchungen überlassen werden, da in den hier betrachteten Sitzungen nur sehr wenige Phänomene dieses Typs identifiziert werden konnten.

13. **Impliziter Widerspruch:** In den hier diskutierten Sitzungen konnten nur wenige *disagree_finding*-Äußerungen identifiziert werden. Es war aber trotzdem möglich, zu beobachten, dass derartige Widersprüche nicht immer direkt, sondern manchmal auch nur implizit übermittelt werden. Ein derartiges Phänomen wird in Tabelle 7.33 erläutert. Es zeigt, in welcher

Weise Vorschläge als Ablehnung fungieren können. Inwieweit weiterführende Untersuchen solche impliziten Ablehnungen kodieren sollten, muss im Einzelfall festgelegt werden.

14. **Unterschiedliche Ursachen von Zustimmungen:** *agree_finding*-Äußerungen sind, wie auch die Zustimmungen zu Äußerungen anderer Klassen (siehe z.B. S. 175), in der Regel kurz. Sie bestehen oft nur aus einem oder zwei Worten bzw. einem Laut, wie z.B. „O.K.“, „Ja“, „Stimmt“, „Hast Recht“ oder „[A]hm“. Ungeachtet ihrer Form konnten unterschiedliche Arten von Affirmationen identifiziert werden:

    (a) ***agree_finding***$_{well-understood}$**:** Für den Sprecher war die vom Partner geäußerte Erkenntnis im Wesentlichen neu. Es war ihm aber möglich, diese nachzuvollziehen bzw. zu verstehen. Nun bestätigt er ihre Korrektheit. Beispielsweise reagiert *P1* in Sitzung PR1.1 auf die Äußerung „Du hast keine Parameter mit übergeben (~~)“ mit einem „Stimmt“, dessen Tonfall darauf hindeutet, dass die Äußerung des Partners ihm dabei geholfen hat, zu verstehen, warum der Aufruf eines PHP-Skriptes im Browser zur Ausgabe einer leeren Seite geführt hat.

    Achtung: Im weiteren Verlauf dieser Arbeit wird auch eine Affirmation vom Typ *agree_knowledge*$_{well-understood}$ erläutert. Sie umfasst unter anderem Zustimmungen, bei denen der Sprecher nur „glaubt“, dass es sich beim übermittelten Wissen um korrekte Informationen handelt. Aus Symmetriegründen ist es naheliegend, auch *agree_finding*$_{well-understood}$ diesbezüglich zu erweitern. Es muss aber festgehalten werden, dass derartige Phänomene bisher im Zusammenhang mit Erkenntnissen nicht zweifelsfrei identifiziert werden konnten.

    (b) ***agree_finding***$_{also\ understood}$**:** Der Sprecher signalisiert, dass er gerade zur selben Einsicht gelangt ist. Beispiel: In Sitzung PR1.1 äußert *P2* „Vermut' ich auch“, nachdem sein Partner festgestellt hatte: „Das Problem ist, wenn man das Skript hier abbricht (.) dann (.) wird der Request nicht beantwortet. (.) Oder,‿obwohl dann wird er seinen Standard-Header senden und auch (~nichts) zurückschicken.“

    Achtung: Im Rahmen der BS/BKM muss sich der Forscher bei derartigen Phänomenen nicht die Frage stellen, wie das Verb „vermuten“ intendiert ist, im Sinne von „der Meinung sein“ oder im Sinne von „die Hypothese haben“. In beiden Fällen ist *agree_finding* zu annotieren, da mittels einer BKM-Konzeptualisierung versucht werden sollte, sprachliche Bezüge sichtbar zu machen (siehe S. 161).

    (c) ***agree_finding***$_{well-known}$**:** Der Sprecher signalisiert, dass ihm die geäußerte Erkenntnis bereits bekannt war, diese für ihn also in gewisser Weise Bestandswissen darstellt. Siehe z.B. das erste *agree_finding* in Tabelle 7.31.

HHI-Kode	Äußerung	Kontext/Kommentar
P2.explain_finding	„Gut.“	Nachdem Änderungen am Kode vorgenommen wurden, ruft *P1* das in Entwicklung befindliche Programm auf, um die Auswirkungen der Modifikation zu testen. Ziel ist es, zu überprüfen, ob eine im Programm aufrufbare Attributtabelle[145] wie gewünscht funktioniert. Nachdem *P1* über Menüs des Programms auf die Tabelle zugegriffen hat, überprüft er, ob auch die Attributeigenschaften aufgerufen werden können. Kurz nachdem diese auf dem Display erschienen sind, schließt er das die Werte darstellende Fenster wieder und quittiert den Test mit „Gut“, was in diesem Kontext als „Ich habe erkannt, dass alles richtig funktioniert“ interpretiert werden kann.
P1.challenge_finding/ P1.propose_step	„Mach mal, (..) änder' mal die Werte (.) und geh' wieder rein“	*P1* reagiert unmittelbar auf die Einschätzung des Partners und weist darauf hin, dass auch die Modifikation der Attributeigenschaften getestet werden sollte. Er macht also einen Vorschlag bez. der nächsten Aktion. Mit diesem Vorschlag widerspricht er implizit der Erkenntnis/Bewertung des Partners. Beide Intentionen sollten markiert werden[146], wobei nicht auf der Hand liegt, ob *disagree_finding* oder *challenge_finding* zu annotieren ist.[147]
		Fortsetzung auf der nächsten Seite.

[145] Grob gesprochen handelt es sich um die Attribute von graphischen, in Landkarten einzufügenden Objekten. Außerdem handelt es sich genau genommen nicht um eine statische Tabelle, sondern um ein Eingabeformular, über das die Werte der Attribute bzw. Attributeigenschaften geändert werden können.

[146] Siehe auch Abschnitt 9.3.1.

[147] Sollte eine wie im vorliegenden Fall geschilderte Problematik auftreten und eine Lösung in Hinsicht auf den Untersuchungsblickwinkel notwendig erscheinen, müssen Eigenschaften oder weitere Konzepte (auch solche höherer Ordnung, z.B. solche, die größere Zusammenhänge beschreiben) entwickelt werden.

		Fortsetzung der Tabelle der vorherigen Seite.
P2.agree_finding/ P2.agree_step	„[A]hm. (.) Ja.“	*P2* fällt *P1* zustimmend ins Wort und ändert sofort eine Attributeigenschaft. Dann schließt er alle Attributtabellen, um sie nachfolgend erneut zu öffnen.
P1.explain_finding	„(O.K.)“	Nachdem die Attributeigenschaften erneut auf dem Display erscheinen, erkennt der Observer, dass die eben vorgenommene Änderung verschwunden ist. Er äußert seine Erkenntnis allerdings verhalten.
P2.explain_finding	„Okayyy, es tut nicht“	Unmittelbar anschließend gibt auch der Driver zu verstehen, dass er erkannt hat, dass das Ändern der Attributwerte nicht funktioniert. Da *P1* seine Erkenntnis zuvor sehr leise artikulierte und *P2* währenddessen gebannt auf den Bildschirm sah, ist davon auszugehen, dass es sich bei der Äußerung von *P2* um die Verbalisierung einer eigenen Erkenntnis und nicht um eine Zustimmung zu der von *P1* handelt.
P1.propose_step	„(~Zeig mal.)“	Ca. zwei Sekunden später macht *P1* den Vorschlag, den betroffenen Kode anzusehen. Allerdings artikuliert er auch diese Äußerung eher verhalten.
P2.propose_step	„Daaann sollten wir schauen wieso.“	Zwei weitere Sekunden später macht *P2* denselben Vorschlag noch einmal. Er scheint den Vorschlag des Partners nicht registriert zu haben.

Tab. 7.33: Episode aus Sitzung PR2.1 (12:39:27–12:39:52), in der ein Observer einer Einsicht des Partners dadurch implizit widerspricht, dass er einen Handlungsvorschlag unterbreitet. Der Widerspruch erweist sich als nützlich, da erst er dazu führt, dass ein Versagen des Programms identifiziert werden kann. Die Episode demonstriert des Weiteren die Grenzen der BS/BKM: Das Phänomen, dass der Driver allem Anschein nach mehrmals Äußerungen des Partners nicht mitbekommt oder sogar ignoriert, wird nur implizit erfasst. Das gleiche gilt für das Phänomen, dass erst ein Widerspruch des Observers zur Entdeckung eines Programmversagens geführt hat.

Streng genommen ist es nur im Fall $agree\_finding_{well-known}$ und $agree\_finding_{also\ understood}$ naheliegend, von Zustimmung zu übermittelten Erkenntnissen zu sprechen. Im Fall von $agree_{well-understood}$ handelt es sich hingegen in gewisser Weise auch um ein *explain_standard of knowledge*. Da aber in vielen Fällen kaum entscheidbar ist, um welche Form von Zustimmung es sich bei einer affirmativen Erkenntnisbewertung handelt[143], wurde für BS-Kodierungen festgelegt, dass grundsätzlich alle potentiell affirmativen Bewertungen von *explain_finding*-Äußerungen mit *agree_finding* zu annotieren sind. Die *agree*-Typen haben dabei die Funktion einer primär charakterisierenden Eigenschaft (siehe Abschnitt 6.2), die dabei helfen soll, im Auge zu behalten, welche Formen von Äußerungen im Rahmen der BS als affirmativ gelten. Die Typen stellen also keinen Bruch mit dem „Black-Box"-Prinzip (siehe S. 124) dar, da sie in BS-Kodierungen nicht vorkommen.[144]

Falls der Sprecher hervorhebt, was er oder in welcher Weise er die Erläuterungen des Partners verstanden hat, wird (zusätzlich) ein *explain_standard of knowledge* kodiert. Darüber hinausgehende Differenzierungen werden weiterführenden Untersuchungen überlassen.

Achtung: Es kann davon ausgegangen werden, dass sich Widersprüche in einer ähnlichen Art und Weise typisieren lassen. Im Rahmen der hier erläuterten Untersuchungen wurde aber darauf verzichtet.

15. **Zustimmungen/Aufmerksamkeit anzeigen:** Siehe S. 175.

16. **Eingeschränkte Zustimmung/Unterscheidung Zustimmung und Ablehnung**: Zustimmungen zu Erkenntnissen können mit (verbleibenden) Zweifeln behaftet sein. Hierauf wird in Abschnitt 9.3.8 eingegangen.

17. **Nach Erkenntnissen Fragen:** Im Rahmen der BS/BKM ist kein Konzept *ask_finding* vorgesehen. Es kann aber passieren, dass ein *ask_knowledge* mit einem *explain_finding* beantwortet wird, eine Frage also dazu führt, dass eine Erkenntnis entsteht. Es ist weiterführenden Untersuchungen überlassen, bei Bedarf ein *ask_finding* einzuführen. In Tabelle 7.34 wird anhand eines Beispiels erläutert, in welchen Situationen ein solches Konzept sinnvoll eingesetzt werden könnte.

Die in diesem Abschnitt formulierten Regeln standen – wie alle anderen auch – nicht von Anfang an zur Verfügung. Sie sind vielmehr Ergebnisse des Herleitungsprozesses. Für die Kodierung der hier beschriebenen Sitzungen bedeutete dies, dass sich der Kodierstil im Laufe der Untersuchungen aufgrund von neuen Erkenntnissen veränderte. Um eine gleichförmige Kodierung zu erhalten, wäre es

[143] Es kann z.B. nicht ausgeschlossen werden, dass es sich im zweiten Beispiel nicht um ein $agree\_finding_{also\ understood}$, sondern um ein $agree\_finding_{well-understood}$ handelt.

[144] Natürlich kann im Rahmen von weiterführenden Untersuchungen darüber nachgedacht werden, ob oder in welcher Weise die *agree_finding*-Typen übernommen werden sollten.

bei jeder Änderung des Kodierstils notwendig gewesen, bereits vorgenommene Annotationen, sofern sie von der Änderung betroffen sind, entsprechend zu modifizieren. Dies wurde aber aus Effizienzgründen nicht in vollem Umfang, sondern nur soweit, wie es zum tieferen Verständnis der neuen Einsichten notwendig war, durchgeführt.

Abgrenzungen von *finding*-Konzepten zu anderen Konzepten

Die wichtigsten Kriterien zur Abgrenzung zwischen *finding*- und *knowledge*-Konzepten wurden bereits auf S. 151 und im Abschnitt über den Fokus der *finding*-Konzepte (S. 250 f) erläutert. Weitere Details folgen im Rahmen der Erörterung der Klasse *knowledge* (siehe S. 331 f). Details zu Abgrenzungen von *finding*-Konzepten zu anderen *UK* werden erst ihm Rahmen der Erörterungen der entsprechenden Klassen ausgeführt. Bleiben die Abgrenzungen zu prozess- bzw. produktorientierten Konzepten. Grundlegend für diese Differenzierungen ist die Festlegung, dass eine Kodierung mit prozess- oder produktorientierten Konzepten in der Regel Vorrang vor einer Kodierung mit *UK* hat. Außerdem muss beachtet werden, dass das im Rahmen von Äußerungen, die mit prozess- und produktorientierten Konzepten zu annotieren sind evtl. mit übertragene Wissen nicht explizit betrachtet wird. Somit muss hier auch nicht zwischen Bestandswissen und Erkenntnissen unterschieden werden (siehe Abbildung 7.1). Detailliertere Erklärungen zur Abgrenzung zwischen *explain_finding* und *propose_design* sind auf S. 174 und S. 180 zu finden, solche zu *explain_finding* und *propose_step* auf S. 192. Im Weiteren seien hier noch einige Spezialfälle erläutert:

- ***explain_finding* vs. **_step*:** Sollte ein Sprecher im Rahmen der Formulierung eines **_step*s explizit eine zugehörige Erkenntnis als solche hervorheben, z.B. in Form eines einleitenden (Teil-) Satzes, muss diese Erkenntnis mit *explain_finding* markiert werden. Im Rahmen der Herleitung der BS/BKM wurden derartige Phänomene z.B. in Situationen beobachtet, in denen die jeweiligen Sprecher nach der Formulierung eines Vorschlags Ereignisse wahrnahmen, welche den Vorschlag als falsch oder zumindest ungünstig erscheinen ließen (siehe Beispiel in Tabelle 7.10). Ein analoges Vorgehen sollte ggf. auch im Zusammenhang mit anderen prozess- oder produktorientierten Konzepten (z.B. *propose_design*) praktiziert werden. Entsprechende Phänomene konnten aber bisher nicht beobachtet werden.

  Achtung: Im Rahmen der hier geschilderten Untersuchungen konnte keine einzige Äußerung identifiziert werden, die darauf abzielte, Anmerkungen in Hinsicht auf die korrekte Verwendung der Syntax der im Einsatz befindlichen Programmiersprache zu machen, wie z.B. „Da gehört ein Semikolon hin“. Trotzdem sei angemerkt, dass derartige Äußerungen mit *explain_finding* und nicht etwa mit *propose_step* oder gar *propose_design* zu annotieren wären. Schließlich steht das Erkennen und Deuten von gerade wahrgenommenen passiven Informationen (siehe S. 253) im Vordergrund.

HHI-Kode	Äußerung	Kontext/Kommentar
P1.explain_finding	„Der letzte Request hier, (, ,) `last` und hier, ähm `change` und dadurch (!...!) Ja, klar. Ja, wir kriegen jetzt 'ne Freundesliste.“	Der Driver denkt darüber nach, welches Ergebnis eine bestimmte `if`-Anweisung in einer festgelegten Situation zurückgeben wird. Hierbei macht er sich Notizen auf einem Zettel und spricht seine Gedanken in Teilen mit. Als Ergebnis formuliert er die Erkenntnis, dass das gerade in Bearbeitung befindliche Skript eine Liste von Freunden einer bestimmten Person liefern wird.
P2.challenge_finding	„Wir kriegen keine Freundesliste.“	*P2* widerspricht seinem Partner unmittelbar danach, indem er das Gegenteil behauptet.
P1.challenge_finding	„Doch wir kriegen eine.“	Nun widerspricht der Driver seinem Partner *P2*, indem er seine alte Behauptung wiederholt.
P2.ask_knowledge	„Wieso, wenn (!!...!!)“	*P2* fragt nach einer Begründung. Er kann nicht aussprechen, da ihm der Driver ins Wort fällt. Die Frage bezieht sich auf die vorher verbalisierte Erkenntnis des Partners. In gewisser Weise könnte man hier also auch von einem *ask_finding* sprechen.
P1.amend_finding	„Wir haben 100 Sekunden nachdem sie das letzte mal gefragt haben (.) – hundert Sekunden, nachdem sie das letzte mal gefragt haben – (.) was geändert. (..) Wir haben also was geändert, nachdem sie gefragt haben. Also kriegen sie jetzt 'ne neue Antwort.“	P1 erläutert seine Erkenntnis, indem er sie näher ausführt.

Tab. 7.34: Episode aus Sitzung PR1.1 (15:17:14–15:17:42), in der ein Phänomen beobachtet werden konnte, welches in gewisser Weise für die Einführung des Konzeptes *ask_finding* spricht. Da bisher nur ein solches Phänomen identifiziert werden konnte und bei diesem auch nicht weitgehend zweifelsfrei davon gesprochen werden kann, dass eine (neue) Erkenntnis „angefordert“ wird, wurde aber auf die Einführung eines solchen Konzeptes verzichtet. Die Episode liefert darüber hinaus ein Beispiel für Situationen, in denen Sprecher ihre eigenen Erkenntnisse näher ausführen (*amend_finding*).

Außerdem handelt es sich bei derartigen Äußerungen nicht um die Verbalisierung einer Gestaltungsmöglichkeit, denn hier wird etwas Zwingendes adressiert. Es sei betont, dass diese Festlegung nicht im Zusammenhang mit konkreten Beobachtungen auf Nützlichkeit oder Trennschärfe verifiziert werden konnte. Dementsprechend kann es sich bei weiterführenden Untersuchungen derartiger Phänomene als sinnvoll herausstellen, zumindest in bestimmten Situationen zusätzlich oder sogar ausschließlich *propose_step* zu annotieren.

- ***explain_finding* vs. *explain_completion*:** Das Konzept *explain_completion* kann als Spezialfall des Konzeptes *explain_finding* betrachtet werden, der explizit aus *explain_finding* „herausgenommen“ wurde. Schließlich handelt es sich bei Fortschrittsbewertungen in der Regel um gerade erlangte Einsichten.
- ***explain_finding* vs. *explain_state*:** Auch das Konzept *explain_state* kann als Spezialfall des Konzeptes *explain_finding* betrachtet werden, der explizit aus *explain_finding* „herausgenommen“ wurde.

Weitere Anmerkungen zu *finding*-Konzepten

Abschließend soll noch einmal festgehalten werden, dass mit der Einführung der Klasse *finding* nicht mit dem behaviouristischen Vorgehen bei der Herleitung der BS/BKM (siehe S. 124) gebrochen wurde, da die Annotation von Erkenntnissen auf dem Identifizieren von Reizen und (verbalen) Reaktionen basiert (der Typ T bildet hier sicherlich einen Grenzfall). Weiterführende Untersuchungen müssen entscheiden, welche theoriebildenden Schlussfolgerungen sich aus diesem Ansatz ziehen lassen. Es liegt aber auf der Hand, dass ein Verzicht auf wie auch immer geartete Erkenntniskonzepte kaum zu einer brauchbaren Theorie der PP führen kann. Schließlich würde man sich durch einen solchen z.B. der Möglichkeit berauben, das Erkennen eines „Fehltuns“ zu analysieren.

7.6.2 *hypothesis*-Konzepte

Fokus der *hypothesis*-Konzepte

Eine *propose_hypothesis*-Äußerung ist eine nicht unmittelbar als Handlungs- oder Gestaltungsvorschlag intendierte Formulierung einer Hypothese oder Vermutung (siehe Exkurs 11), wobei von dem Konzept *propose_hypothesis* sowohl formal wie auch empirisch „beweisbare“ Hypothesen/Vermutungen adressiert werden. Hierbei gilt: Um mit dem Konzept *propose_hypothesis* bezeichnet werden zu können, müssen Äußerungen weder derart präzise formuliert sein, dass sie nachfolgend ohne weiteres überprüft werden können, noch müssen die durch sie übermittelten Sachverhalte allgemein als noch nicht überprüft bzw. bez. ihres Wahrheitsgehaltes unbekannt gelten. Die Hauptcharakteristik einer *propose_hypothesis*-Äußerung besteht vielmehr darin, dass der Sprecher nicht hundertprozentig für

***Exkurs 11:** Hypothesen und Vermutungen*

Die Begriffe Hypothese und Vermutung werden in der Literatur nicht immer synonym verwendet. So wird unter einer Hypothese oft eine überlegte, explizit formulierte, prüfbare Aussage verstanden, der Gültigkeit unterstellt wird, die aber nicht bewiesen oder verifiziert ist (siehe z.B. [126, S. 76]), während eine Vermutung eine unsicherere Erkenntnis darstellt, die evtl. nur vage formuliert ist. Außerdem wird der Begriff Hypothese auch als Abkürzung für den Terminus „wissenschaftliche Hypothese“ verwendet. Hierbei handelt es sich um „Annahmen über reale Sachverhalte (empirischer Gehalt, empirische Untersuchbarkeit) in Form von Konditionalsätzen. Sie weisen über den Einzelfall hinaus (Generalisierbarkeit, Allgemeinheitsgrad) und sind durch Erfahrungsdaten widerlegbar (Falsifizierbarkeit)“ [28, S. 4].

Da mit der Klasse *hypothesis* alle Formen von Vermutungen und Hypothesen adressiert werden sollen, insbesondere auch solche, die nicht über den Einzelfall hinaus gehen, nicht in Form von Konditionalsätzen formuliert sind oder sich (praktisch) nicht falsifizieren lassen, wird im Zusammenhang mit dieser Klasse oft von Hypothesen *und* Vermutungen gesprochen – abkürzend manchmal aber auch nur von Hypothesen oder nur von Vermutungen.

Achtung: Die Klasse *hypothesis* ist zwar dahingehend konzipiert, auch wissenschaftliche Hypothesen zu adressieren, diesbezügliche Äußerungen konnten aber bisher nicht beobachtet werden.

die Richtigkeit seiner Aussage bürgt und mehr oder weniger explizit betont, dass es sich um einen nur in einem gewissen, nicht unbedingt quantifizierbaren Maß wahrscheinlichen Sachverhalt handelt. Dieser Vermutungscharakter, der die Klasse *hypothesis* als Ganzes bestimmt, bildet grob gesprochen das Abgrenzungskriterium zu dem, was im Rahmen der BS/BKM *finding* oder *knowledge* genannt wird. Ein Indiz für eine *hypothesis*-Äußerung, insbesondere für ein *propose_hypothesis*, kann die Verwendung von Wörtern und Phrasen wie z.B. „vermutlich“, „wahrscheinlich“, „ich glaube“ oder „das scheint“ sein. Eine *hypothesis* kann zum Bestandswissen des Sprechers gehören, also eine Konjektur sein, die der Sprecher schon länger hat, oder von ihm gerade erst aufgestellt worden sein.

Hypothesen und Vermutungen können sich auf unterschiedliche Typen von Entitäten beziehen. In Tabelle 7.35 sind diejenigen aufgelistet, die im Rahmen der hier betrachteten Sitzungen identifiziert werden konnten. Die Beispiele in der Tabelle verdeutlichen darüber hinaus, welche Formen von Hypothesen bzw. Vermutungen durch *hypothesis*-Konzepte adressiert werden. Im Einzelnen handelt sich hierbei um die folgenden:[148]

- **Nicht sicheres Bestandswissen (*uncertain knowledge* (H_{uck}))**: Vermutungen die eigentlich keine solchen sein müssten, da sie Sachverhalte thematisieren, die dem Sprecher bzw. dem Paar eigentlich bekannt sein sollten

[148] Achtung: Die Liste von *hypothesis*-Typen erhebt keinen Anspruch auf Vollständigkeit. Sie führt lediglich diejenigen Typen auf, die im Rahmen der hier untersuchten Sitzungen beobachtet werden konnten.

oder könnten und die im Allgemeinen nicht von der aktuellen Situation bzw. dem aktuellen Kontext beeinflusst werden. Zur Verdeutlichung betrachte man das Beispiel zur Entität „Konventionen“ in Tabelle 7.35. Hier bezieht sich die Vermutung auf eine vom Sessionverlauf unabhängige Festlegung (Syntax eines SVN-Kommentars), die beiden Entwicklern eigentlich bekannt sein müsste.

– **Bestenfalls mittel- oder langfristig verifizierbare Annahmen (*long-run-verifiable assumptions* bzw. *hard-to-verify assumptions* (H_{lva}))**: Vermutungen die von den Entwicklern allem Anschein nach zur Zeit oder kurzfristig[149] gar nicht oder nur mit großem, evtl. nicht abschätzbaren Aufwand überprüft werden können. Als Beispiel betrachte man Äußerung 2 zur Entität „Prozessverlauf“ in Tabelle 7.35. Hier ist es den Entwicklern ohne größeren Aufwand bzw. kurzfristig kaum möglich, die aufgeführte Vermutung zu verifizieren.

  Zu dieser Klasse von Vermutungen gehören auch solche, bei denen den Sprechern nicht klar zu sein scheint, auf welche Weise sie überprüft werden könnten. Zur Verdeutlichung sei hier auf das zweite Beispiel zur Entität „Eigenschaften bzw. Verhalten der Umgebung“ in Tabelle 7.35 verwiesen.

– **Kurzfristig verifizierbare Annahmen (*short-run-verifiable assumptions* (H_{sva}))**: Vermutungen bez. allem Anschein nach kurzfristig von den Entwicklern nachprüfbaren Sachverhalten. So lässt sich die im ersten Beispiel zur Entität „Prozessverlauf“ in Tabelle 7.35 aufgeführte Vermutung kurzfristig verifizieren. Denn um festzustellen, ob ein „Undeploy“ das Problem lösen würde, müsste nur ein solches durchgeführt werden (Details zur Reaktion des Partners sind auf S. 288 beschrieben).

[149] Bez. der Begrifflichkeiten kurz, mittel- und langfristig siehe Fußnote 61 auf S. 187.

Entitäten	Beschreibung	Beispiele
Eigenschaften bzw. Verhalten des Programms	Hypothesen bez. des Verhaltens des in Ausführung befindlichen Programms – sowohl zu Ursachen gerade beobachteten wie auch in Hinsicht auf zukünftiges Verhalten.	1. ST1.1: Nachdem *P1* das in Entwicklung befindliche Programm gestartet und eine Operation über dessen Webfrontend aufgerufen hat, wird eine `NullPointerException`[150] geworfen. *P1* kommentiert dies wenige Sekunden später. Hierbei markiert er im Quelltext einen Aufruf der Methode `receive`[151]: „weißt⌣Du,⌣was⌣jetzt⌣passiert⌣ist, (...) ich vermute, (~nein) ich⌣weiß⌣es⌣nicht,⌣aber⌣ich vermute, (..) dass das `receive` nach 10 Sekunden nichts bekommen hat.“ 2. PR1.1: Nachdem der Driver einen Übergabewert an ein PHP-Skript geändert hat, spekuliert der Observer darüber, was das Skript nun zurückgeben wird: „Is' größer - (~also wir) sollten eigentlich ein `exit` kriegen und keine (~Freundesliste), oder?“[152]
Eigenschaften bzw. Verhalten der Umgebung	Vermutungen in Hinsicht auf die Entwicklungs- und Laufzeitumgebung, z.B. bez. der Eigenschaften von Entwicklungswerkzeugen – wie IDEs oder *Build*-Systemen – oder der zur Verfügung stehenden Netzwerkverbindungen.	1. ST1.1: Die Entwickler sind dabei, sich Konfigurationsproblemen zu widmen, die ihrer Meinung nach mit der Art der Verwendung von MBeans[153] zusammenhängen. Hierbei stellt der Observer *P2* eine Vermutung auf: „Macht da äh (!...!).⌣Da (~hunzt) äh XDoclet[154] uns rein, (..) vielleicht. (..) Oder?“ 2. PR1.1: Der Observer vermutete, dass Einstellungen der Firewall dafür verantwortlich sind, dass die Ausführung des Programms Wget[155] fehlschlägt. Er formuliert seine Hypothese in einem Satzfragment: „Firewall?“
		Fortsetzung auf der nächsten Seite.

[150] `java.lang.NullPointerException`

[151] Es handelt sich um den Aufruf der Methode `receive` auf einem Objekt vom Typ `javax.jms.MessageConsumer` (Teil der JMS-API, http://download.oracle.com/javaee/1.4/api/javax/jms/MessageConsumer.html (Abruf: 26.01.2011)).

[152] Man beachte folgende Punkte: 1. Genau genommen handelt es sich um ein *explain_finding*, welches von einem *propose_hypothesis* gefolgt wird. 2. Erst auf S. 338 wird erläutert, warum derartige Fragen als *propose_hypothesis* und nicht als *ask_knowledge* behandelt werden sollten. 3. Der Sprecher spricht verkürzt von `exit` und bezieht sich dabei auf die Kodezeile `exit('NOCHANGE')`.

[153] MBeans gehören zum Funktionsumfang der „Java Management Extensions“-Spezifikation (JMX, http://java.sun.com/javase/technologies/core/mntr-mgmt/javamanagement/ (Abruf: 26.01.2011)).

[154] Siehe Fußnote 23 auf S. 23.

[155] *Wget* ist ein Kommandozeilenprogramm zum Herunterladen von Dateien aus dem Internet (http://www.gnu.org/software/wget/ (Abruf: 08.08.2011)).

		Fortsetzung der Tabelle der vorherigen Seite.
Eigenschaften der Programmartefakte	Hypothesen, die sich direkt auf Eigenschaften des Programmkodes oder andere Programmartefakte beziehen.	1. PR1.1: Nachdem das in Entwicklung befindliche Programm in einem Testlauf nicht das gewünschte Ergebnis geliefert hat, vermutet der Observer *P2*, dass in der Bedingung einer bestimmten `if`-Anweisung die Klammern falsch gesetzt sind: „(#`Safe::id_to_code`#). Wahrscheinlich (~irgendwelche) Klammerfehler.“[156] 2. PR2.1: Bevor *P1* eine Klasse in ein anderes Paket verschiebt, stellt er eine Vermutung über deren Verwendung an: „(So, die wird jetzt wahrscheinlich auch gar nicht mehr hier verwendet.)“
Prozessverlauf	Hypothesen bez. Eigenschaften des bisherigen oder zukünftigen Sitzungsverlaufs.[157]	1. ST1.1: Nachdem *P1* zum wiederholten Mal den *Build*-Prozess gestartet hat, stellt er eine neue Hypothese bezüglich der Ursache für das Versagen, das sich bei den vorhergehenden, den *Build*-Prozess-Aufrufen folgenden Tests der in Entwicklung befindlichen Webapplikation ereignete: „Oder wir hätten doch ein echtes ‚Undeploy‘ vielleicht machen sollen.“[158] 2. PR2.1: Nachdem *P2* einen strategischen Vorschlag gemacht hat, merkt er an: „Ich fürcht’ halt, dass bei dem Arbeitspunkt, dass wir sowieso die Zeit überschreiten werden.“[159]
Konventionen	Hypothesen bez. jeglicher Formen von allgemeinen Formatfestlegungen, seien es solche zum Programmierstil oder zu sonstigen Formaten. Auch Hypothesen zu allgemeinen Vorgehensvorschriften fallen unter diese Klasse.	PR2.1: *P1* ist dabei, ein SVN-*Commit* durchzuführen, und hat bereits damit begonnen, das Kommentarfeld[160] auszufüllen. Im Unternehmen ist festgelegt, dass ein solcher Kommentar durch einen wohldefinierten Aufgabenbezeichner einzuleiten ist. Diesbezüglich hat *P1* ‚cad-507:‘ notiert. Daraufhin wirft *P2* ein: „Ich glaub’, das ‚cad‘ groß oder so?“

Tab. 7.35: Beispiele für diejenigen Entitäten, auf die sich Hypothesen beziehen können. Es sind nur solche Entitäten gelistet, für die beobachtet werden konnte, dass sie tatsächlich im Rahmen von *propose_hypothesis*-Äußerungen referenziert werden.

[156] Mit `Safe::id_to_code` beginnt der Teil der Bedingung, in dem *P2* den Defekt vermutet.
[157] Prozessfortschrittsbewertungen werden mit *completion*- bzw. *state*-Konzepten annotiert.
[158] Der Sprecher spricht in Richtung Monitor. Nichts deutet darauf hin, dass es sich um eine direkt an den Partner gerichtete Frage handelt. Vielmehr scheint P1 mit der Äußerung eine Hypothese zu formulieren, die sich folgendermaßen paraphrasieren lässt: „Es kann sein, dass ein echtes ‚Undeploy‘ hätte durchgeführt werden müssen.“
[159] Weitere Details zum Kontext dieser Äußerung sind in Tabelle 7.22 dargestellt.
[160] Für den Zugriff auf das SVN-Repository wird ein Eclipse-Plugin verwendet. Dieses fragt im Rahmen eines Wizards einen *Commit*-Kommentar ab.

Identifizierte *hypothesis*-Konzepte und ihre speziellen Eigenschaften

Folgende *hypothesis*-Konzepte konnten aus den Daten extrahiert werden: *propose_hypothesis*, *agree_hypothesis*, *disagree_hypothesis* und *amend_hypothesis*. Sie wurden bereits grob in Abbildung 7.3 beschrieben. Details werden nachfolgend erörtert.

1. **Vorschlagscharakter einer Hypothesenäußerung:** Mit *propose_hypothesis* werden Äußerungen adressiert, in denen dargelegt wird, auf welche Weise sich etwas erklären lassen oder welche Eigenschaften etwas besitzen *könnte*. Somit war es im Rahmen der Herleitung der BS/BKM angemessener das Verb *propose* als das Verb *explain* bei der Benennung des Konzepts zu verwenden.

2. **Zustimmung und Ablehnung von Hypothesen**: Bei einem *agree_hypothesis*-Phänomen handelt es sich um eine Äußerung, in der der Sprecher der Ausrichtung einer zuvor artikulierten Hypothese oder Vermutung im Grundsatz zustimmt und nicht etwa um eine solche, mit der er versucht, zu belegen (oder erklären), dass es sich (wirklich) um eine richtige Vermutung handelt. Bei einem *disagree_hypothesis* verneint er die potentielle Gültigkeit im Grundsatz. Beispiele sind in Tabelle 7.36 zu finden.

   Achtung: Falls ein Entwickler die Richtigkeit einer Hypothese seines Partners begründen kann, sollten die entsprechenden Äußerungen mit *agree_hypothesis* sowie *explain_knowledge* (ggf. *explain_finding*) annotiert werden. Analoges gilt für die Zurückweisung von Hypothesen. Konzepte der Art *support_hypothesis* oder *falsify_hypothesis* sind im Rahmen der BS/BKM nicht vorgesehen. Ihre Einführung wird weiterführenden Untersuchungen überlassen. Aus einer BS-Kodierung kann also nicht direkt entnommen werden, ob ein Entwickler erklärt, warum er eine Hypothese für plausibel (siehe z.B. Zeile 3 in Tabelle 7.36) oder unplausibel hält oder ob er zeigen kann, dass sie richtig oder falsch ist.

3. **Bedingte Zustimmungen:** Die Zustimmung zu einer Hypothese kann von einer Bedingung abhängig gemacht sein. So stimmt *P2* der Hypothese seines Partners („Oder wir hätten doch ein echtes ‚Undeploy' vielleicht machen sollen." (siehe Tabelle 7.35)) mit folgenden Worten zu:

   > „Wenn's jetzt immer noch nicht funktioniert, dann (~können wir es machen)." (*agree_hypothesis*/*propose_step*)

   Derartige die Hypothese selbst nicht modifizierende Bedingungen werden im Rahmen der BS/BKM als zu *agree*-Äußerungen gehörig gewertet.[161]

[161] Die Zustimmung ist hier in einen Vorschlag „verpackt". Somit ist eine Doppelkodierung mit *agree_hypothesis* und *propose_step* angebracht.

propose_hypothesis	***agree_-/ disagree_hypothesis***	**Kontext/Kommentar**
P1: „Also ich denk mal, dass (.) <**Entwicklername**> die auf 'n Opera hat.“	*P2*: „(O.K.)“	PR2.1: Die Entwickler haben vor Kode in ein SVN-Repository zu überspielen (*Commit*). Hierbei haben sie einen *Commit*-Kommentar zu schreiben[a], der die genaue Bezeichnung (ID) der von ihnen bearbeiteten Aufgabe enthalten muss. Diese ist beiden Entwicklern nicht bekannt. *P1* vermutet, dass sie die Bezeichnung über den Rechner eines Kollegen (<**Entwicklername**>), auf den sie einen Remote-Desktop-Zugriff[b] haben – genau genommen über den dort installierten Browser (Opera[c]) – ermitteln können. Während des Sprechens beginnt er seiner Hypothese nachzugehen. Hierbei verhält er sich desorientiert, was die Annahme untermauert, dass es sich bei seiner Äußerung um eine Vermutung handelt. *P2* stimmt mit einem leisen „(O.K.)“ zu.
P1: „Also ich glaub' nicht, dass wir am `FeatureProxy` was ändern müssen.“	*P2*: „Hmhm, gut.“	PR2.1: Im Rahmen einer Diskussion über die vor ihnen liegenden Arbeitsschritte äußert *P1* die Vermutung, dass es nicht nötig sein wird, die Klasse `FeatureProxy` zu editieren. *P2* stimmt zu (siehe auch S. 226).
P1: „Weißt⌣Du⌣was⌣jetzt⌣passiert⌣ist, (...) ich vermute, (~nein) ich⌣weiß⌣es⌣nicht,⌣aber⌣ich vermute, (..) dass das `receive` nach 10 Sekunden nichts bekommen hat.“	*P2*: „Ja, es hat ja auch eine Weile gedauert – stimmt.“	ST1.1: Siehe Tabelle 7.35 unter Punkt „Eigenschaften bzw. Verhalten des Programms“. P2 ergänzt seine Zustimmung um eine Erklärung von Typ *explain_finding*.
P2: „Ja, äh, man muss irgendwie um das (.) zum Laufen zu bringen muss man gewisse Sachen machen und da fehlt irgendwas noch.“[d]	*P1*: „Aber das haben wir doch alles gemacht.“	ST1.1: Der Versuch, sich in der weiterzuentwickelnden J2EE-Applikation (Webshop) anzumelden, führt dazu, dass eine `Exception` geworfen wird. *P2* vermutet, dass nicht alle hierfür notwendigen Einstellungen durchgeführt wurden. Hierbei sieht er in die den Entwicklern vorliegende Kurzbeschreibung der Konfiguration. Sein Partner *P1* weist diese Hypothese in Form einer Ablehnungsbegründung (*explain_finding*) zurück.

[a] Siehe auch Tabelle 7.35.

[b] Es kommt ein *Windows Remote Desktop* (http://www.microsoft.com/windowsxp/downloads/tools/rdclientdl.mspx (Abruf: 03.03.2011)) unter Windows XP zum Einsatz.

[c] http://de.opera.com/ (Abruf: 03.03.2011)

[d] Eine Kodierung als *explain_knowledge* oder *explain_finding* wurde nicht vorgenommen, da in der Äußerung nur eine sehr vage Aussage formuliert wird.

Tab. 7.36: Beispiele für Zustimmungen zu oder Ablehnungen von Hypothesen.

4. **Eigene Hypothesen widerrufen:** Ähnlich wie bei *explain_finding* (siehe S. 269) konnten auch bei *propose_hypothesis* Äußerungen identifiziert werden, die kurze Zeit später vom Sprecher selbst widerrufen wurden. So formuliert der Driver in Sitzung PR2.1 eine Hypothese, um sie unmittelbar danach in Frage zu stellen:

   > „Es könnte (~nämlich) sein, dass es ausgerechnet diese eine (~) `AttributeConfiguration` ist, in der wir es noch nicht drin haben.“ (*P2.propose_hypothesis*)

   > „Wobei, ich glaub's nicht.“ (*P2.disagree_hypothesis*)

5. **Eigene Hypothesen ersetzen:** Im Rahmen der hier analysierten Sitzungen konnte auch das Phänomen beobachtet werden, dass Entwickler ihre eigenen Hypothesen vollständig modifizieren, dass heißt durch neue Hypothesen ersetzen (z.B. dann, wenn sich die ursprüngliche als nicht haltbar herausgestellt hat). So äußert *P1* in Sitzung PR2.1 die Hypothese

   > „Das muss, du musst – die Demo hier unten <*zeigt auf die im *Package Explorer* von Eclipse sichtbare Klasse `FeatureLayer-AttributeDemo`*>, (.) da müsste es drin sein, dieses `Edit-ColumnAttribute`“ (*P1.propose_hypothesis*),

   um sie nach einer kurzen Diskussion mit dem Partner (von ca. 12 Sekunden), in deren Verlauf die Klasse `FeatureLayerAttributeDemo` *nicht* geöffnet wird, durch folgende zu ersetzen:

   > „Moment, ne, dann muss es doch hier oben sein <*zeigt auf eine andere Stelle im *Package Explorer*>, n[e]?“[162] (*P1.challenge_hypothesis*)

6. **Widersprüche mit Hypothesencharakter:** Es fällt auf, dass der oben erläuterte Hypothesenwiderruf („Wobei, ich glaub's nicht.“) wenig strikt formuliert ist. Er hat vielmehr selbst die Form einer Vermutung (genau genommen die einer der ursprünglichen Hypothese entgegengesetzten). Dass die Äußerung nicht als *challenge_hypothesis* konzeptualisiert wird, liegt darin begründet, dass alle sonstigen Indizien darauf hinweisen, dass die Illokution der Äußerung nicht darin besteht, etwas Neues vorzuschlagen (nämlich eine andere Hypothese), sondern darin klarzustellen, dass die ursprünglich geäußerte Hypothese wohl doch nicht richtig ist und somit auch nicht überprüft werden sollte. So ist zu erkennen, dass *P2* keine unmittelbare Überprüfung oder weitere Diskussion seiner ursprünglichen Hypothese anstrebt. Stattdessen fügt er hinzu, dass er sich bezüglich dieses Sachverhaltes nicht auskennt:

[162] Die nachfolgenden Äußerungen von *P1* bekräftigen die These, dass es sich bei dieser Äußerung um eine Vermutung handelt.

„Ich weiß es nicht.“ (*P2.explain_standard of knowledge*)

Es ist also naheliegender, die Äußerung mit *disagree_hypothesis* als mit *challenge_hypothesis* zu kodieren.

Hinweis: Das Beispiel demonstriert noch einmal die Anwendung der in Abschnitt 7.2 erläuterten zentralen Prinzipien.

7. **Hypothesenvorschläge gegeneinander abgrenzen:** Eine Entscheidung darüber, ob mit einer Äußerung eine neue Hypothese/Vermutung formuliert wird oder es sich um die Erweiterung einer bestehenden handelt, hängt davon ab, was unter einer Erweiterung einer *hypothesis*-Äußerung verstanden werden soll. Für Hypothesen in Form von „Wenn-dann-Sätzen“ ist es relativ einfach möglich, eine diesbezügliche Festlegung vorzunehmen. So ist es naheliegend, dann von einer Erweiterung bzw. Modifikation ohne prinzipielle Ablehnung zu sprechen, wenn eine Erweiterung des „Wenn-Teils“ (der so genannten Bedingung) in Form von Disjunktionen oder Konjunktionen (siehe [28, S. 6]) vorgenommen wird (wenn auch eine Hypothese in der Regel bei Erweiterungen der Bedingung um Konjunktionen an Informationsgehalt verliert [28, S. 6]). Ebenso kann man bei derartigen Erweiterungen des „Dann-Teils“ (der so genannten Folge) von einer Erweiterung der Hypothese sprechen.

   Achtung: Es wird an dieser Stelle davon abstrahiert, dass ein Sprecher die Bedingung einer Hypothese derart um eine Konjunktion erweitern kann, dass die Hypothese nicht mehr verifizierbar ist, er die Erweiterung also nur als rhetorisches Mittel verwendet, um die Hypothese abzulehnen.

   Bei Hypothesen/Vermutungen, die nicht derart formuliert sind, ist es schwieriger, nachfolgende Äußerungen als Erweiterung dieser zu klassifizieren. In solchen Fällen sollte versucht werden, die Hypothese in eine „Wenn-dann-Form“ zu paraphrasieren und dann zu prüfen, inwieweit die neu zu klassifizierende Äußerung eine Erweiterung im obigen Sinne darstellt oder doch „nur“ eine weitere Hypothese, Plausibilisierung in Hypothesenform etc. ist.

   Hinweis: Im Rahmen der hier erläuterten Analysen konnten keine ohne jeden Zweifel als *amend_hypothesis* zu konzeptualisierende Phänomene identifiziert werden. Bestenfalls das in Tabelle 7.37 erläuterte Beispiel könnte als ein solches angesehen werden. Denn hier wäre es möglich die Hypothesenäußerungen H_{1_1} und H_3 zusammenzuführen und folgendermaßen zu paraphrasieren:

   > „Da ich es für möglich halte, dass das `basedir` das Eclipse-Projekt ist, könnte es sein, dass diese ‚Punkt Punkt‘ falsch sind.“

Bzw. könnte man die Äußerung wie folgt umformulieren:

> „Wenn das `basedir` das Eclipse-Projekt ist, sind diese ‚Punkt Punkt' falsch."

Somit wäre Äußerung H_3 als *amend_hypothesis* zu klassifizieren, da sie die Hypothese H_{1_1} zu einer „Wenn-dann-Hypothese" macht. Aufgrund dieser nicht gänzlich unplausiblen Interpretation wurde das Konzept *amend_hypothesis* in die BKM aufgenommen. Nähere Eigenschaften von *amend_hypothesis* müssen aber im Rahmen zukünftiger Untersuchungen eruiert werden. Die hier dargelegten Überlegungen können dabei als Ausgangspunkt verwendet werden – insbesondere auch als eine solcher, der in Frage zu stellen ist. Schließlich fußen sie kaum auf beobachteten Phänomenen.

8. **Begründung von Hypothesen:** Es konnte beobachtet werden, dass Hypothesen nicht nur einfach „unbegründet in den Raum gestellt" sondern in vielen Fällen explizit motiviert oder begründet werden. Solche Begründungen sollten entsprechend ihrem Typ separat kodiert werden, z.B. als *explain_knowledge* oder auch als weiteres *propose_hypothesis* (siehe Beispielepisode in Tabelle 7.37). Hierbei muss allerdings zwischen Begründungen von Hypothesen und initial zu Hypothesen gehörenden Bedingungen, im Sinne von „Wenn-Teilen" in „Wenn-dann-Sätzen", unterschieden werden. Nur im ersten Fall ist eine solche separate Kodierung angebracht (siehe z.B. Tabelle 7.22). Im anderen Fall ist eine Kodierung mit *propose_hypothesis* ausreichend (siehe z.B. die Paraphrasierung weiter oben).

9. **Hypothesen als Begründung:** Wie gerade schon angesprochen formulieren Entwickler Hypothesen und Vermutungen auch mit dem Ziel, andere meist zuvor gemachte Äußerungen zu begründen bzw. zu plausibilisieren (siehe Beispiel in Tabelle 7.37). In solchen Fällen ist eine separate Kodierung angebracht.

Abgrenzungen von *hypothesis*-Konzepten zu anderen Konzepten

Es interessieren besonders Abgrenzungen zu den Konzepten der Klassen *finding* und *knowledge*. Abgrenzungen zu *finding* werden nachfolgend anhand von *propose_hypothesis* vs. *explain_finding* näher beleuchtet, Abgrenzungen zu *knowledge*, vor allem solche zwischen Hypothesen vom Typ H_{uck} und *ask_knowledge*, auf S. 338f.

- ***propose_hypothesis* vs. *explain_finding*:** Wie schon erläutert, kann es sich auch bei Hypothesen um Erkenntnisse handeln, präzise formuliert, um am Ende eines Erkenntnisprozesses stehende Vermutungen. Werden derartige Hypothesen identifiziert, ist festgelegt, dass nur Konzepte der Klasse

HHI-Kode	Äußerung	Kontext/Kommentar
P1.propose_hypothesis	„Ich glaube, diese ‚Punkt Punkt' da sind falsch. (..) Und‿es‿könnte‿sein, dass ich die gesetzt habe, auf meinem Rechner."	Da ein zuvor gestarteter *Build*-Prozess nicht die gewünschten Ergebnissen geliefert hatte und nachfolgende Versuche, die Ursachen zu verstehen, erfolglos waren, sehen sich die Entwickler nun die Konfigurationsdatei `build.properties` an.[163] Der Driver äußert in diesem Zusammenhang die Vermutung (H_{1_1}), dass bestimmte Pfadangaben in der Datei falsch sind. Hierbei bezieht er sich mit „Punkt Punkt" auf Dateipfade, die einen Wechsel in ein Elternverzeichnis enthalten (wie z.B. `conf.dir = ${basedir}/../conf`). Die Konfigurationsdatei wurde vor der Sitzung vom Rechner von *P1* auf den in der Sitzung verwendeten Rechner kopiert. *P1* vermutet nun (H_{1_2}), dass er die Verzeichniswechsel auf seinem Rechner eingefügt hatte (diese aber auf dem Sitzungsrechner zum beobachteten Fehlverhalten führen). Genau genommen handelt es sich also um zwei Hypothesen, wobei der Sprecher die erste mit der zweiten zu plausibilisieren versucht.[164]
P1.mumble_sth	„Ich, äh (!...!)"	Der Driver löscht an drei Stellen in der Datei den Wechsel in Elternverzeichnisse.
P2.ask_knowledge	„Sind das alle Dateien so, oder?"	Unmittelbar nachdem *P1* mit seinen Änderungen fertig ist, fragt der Observer, ob auch an anderen Stellen des Projektes Pfade zu Dateien geändert werden müssen.
P1.propose_hypothesis	„Hm, hmmm. Nee, dass ist nur an der (.) Stelle, dürfte das proble(.)matisch sein."	*P1* scheint sich nicht wirklich sicher zu sein. Seine Antwort hat die Form einer Vermutung (H_2).
		Fortsetzung auf der nächsten Seite.

[163] Siehe Fußnote *a* in Tabelle 7.32.

[164] Die erste Hypothese ist vom Typ H_{KVB}, die zweite vom Typ H_{uck}.

		Fortsetzung der Tabelle der vorherigen Seite.
P1.explain_knowledge + *P1.propose_hypothesis*	„Weil◡ich◡ja◡vorher◡das◡Projekt◡anders◡hatte.◡Das◡`basedir`◡ist◡glaube◡ich◡das (..) Eclipse-Projekt, (n[e]) was alles (!...!)“	*P1* begründet, wie er zu seiner zuvor geäußerten Hypothese gekommen ist. Diese Begründung besteht zum einen aus einer Äußerung, an deren Richtigkeit er keinen Zweifel lässt, und zum anderen aus einer Vermutung (H_3) über die Bedeutung des Bezeichners `basedir`. Während er spricht, startet er den *Build*-Prozess erneut.
P2.explain_standard of knowledge + *P2.agree_hypothesis*	„Ah,◡jetzt◡versteh'◡ich◡dich.◡Aber, du hast schon Recht.“	Nachdem der Driver seine Begründung fertig formuliert hat, ist auch der *Build*-Prozesses abgeschlossen – allerdings mit mehreren Fehlermeldungen. In diesem Moment gibt der Observer zum Ausdruck, dass er jetzt verstanden hat, was sein Partner sagen wollte. Mit dem zweiten Teil seiner Äußerung stimmt er dann der Ausrichtung der letzten Hypothese des Partners zu.

Tab. 7.37: Episode aus Sitzung ST1.1 (16:14:52–16:15:27), in welcher der Driver Hypothesen plausibilisiert/begründet. Die Begründungen äußert er zum einen in Form einer Hypothese (H_{1_2}) und zum anderen derart, dass man davon ausgehen muss, dass es sich um ein von ihm schon länger als richtig klassifiziertes Faktum handelt.

hypothesis zu annotieren sind (siehe Abbildung 7.10). Es stellt sich in diesem Zusammenhang allerdings die Frage, inwieweit es überhaupt möglich ist zu entscheiden, ob es sich bei einer Äußerung um die Formulierung einer Vermutung oder die einer als wahr eingeschätzten, neu erlangten Einsicht handelt. Schließlich benutzen die Entwickler eher selten Formulierungen wie „Ich habe die Vermutung, dass ...“, „Ich vermute, dass folgende Hypothese stimmt“ oder auch „Jetzt ist mir klar, dass...“. Die Äußerungen ähneln in der Regel eher der folgenden:

> „Ahhh, ich glaube, er <*Eclipse*> hat uns dieses (!...!), er erkennt das hier gar nicht als PHP-Projekt (~an).“[165] (PR1.1)

Hier stellt sich die Frage, in welchem Sinne der Sprecher das Verb „glauben“ verwendet, im Sinne von „vermuten“ oder im Sinne von „der Meinung sein“. Ersteres würde für ein *propose_hypothesis*, zweiteres für ein *explain_finding* sprechen. Hier kann nur die Analyse des Kontextes (evtl. unter Zuhilfenahme der Kodierung im Paar (siehe Abschnitt 5.2.4)), insbesondere auch der nachfolgenden Aktivitäten helfen, um zu einer nachvollziehbaren Kodierung zu kommen. Im vorliegenden Fall führte eine solche Analyse dazu, dass das Phänomen mit *explain_finding* kodiert wurde.

Es lässt sich aber nicht immer hinreichend zweifelsfrei ermitteln, welche Zwecke ein Sprecher mit einer Äußerung primär verfolgt hat. So ist beispielsweise eine Äußerung wie

> „Da müsste XPet wieder Emails verschicken können, definitiv“[166] (ST1.1)

nur schwierig zu deuten, da die Verwendung des Konjunktivs und das angehängte „definitiv“ im Widerspruch zueinander stehen.[167] Nur durch eine direkt Nachfrage könnte ermittelt werden, wie sicher sich der Sprecher bez. seiner Aussage wirklich ist. Da dies in der Regel nicht möglich ist, sollte in weiterführenden Untersuchungen darüber nachgedacht werden, ob und in welcher Weise derartige Äußerungen berücksichtigt werden können. Ein weiteres Beispiel hierzu ist in Tabelle 7.38 erläutert.

[165] Der Sprecher bezieht sich auf die zur Zeit nicht funktionierende Syntaxhervorhebung von Eclipse.

[166] Der Sprecher bezieht sich auf die Wirkung eines gerade durchgeführten Editierschritts (Modifikation einer Funktionalität der in Entwicklung befindlichen Webapplikation XPetstore (XPet)).

[167] Eine Möglichkeit, den vermeintlichen Widerspruch aufzulösen, besteht darin, anzunehmen, dass der Sprecher seine Einschätzung im Laufe seiner Äußerung ändert. Ob eine solche Annahme angemessen ist, muss in weiterführenden Untersuchungen entschieden werden.

#	HHI-Kode	Äußerung	Kontext/Kommentar
1	*P2.explain_finding*	„[A]hhh, er kennt die Klasse nicht.“	Der Observer kommentiert Fehlermeldungen im aktuell geöffneten „Editor-Tab“ von Eclipse (siehe auch Screenshot 7.12).[a]
2	*P1.explain_finding*	„((#`VirtualColumnAttribute`#) (~))“	Direkt nach dem Vorschlag des Partners liest der Driver einen kleinen Teil des Bildschirminhaltes mit sehr leiser Stimme vor.
3	*P2.propose_step*	„Is’ wahrscheinlich so (!...!) (.) (!!Mach‿(~mal)‿ein‿*Organize*‿*Imports*!!).‿Mach‿einfach‿ein‿*Organize Imports*.“[b]	Der Observer unterbreitet eine Vorschlag zur Problemlösung. Obwohl sein Partner ihm dabei ins Wort fällt, spricht er weiter. Der Einwurf von *P1* ist unverständlich (*P1.mumble_sth*).
4	*P1.ask_knowledge*	„(!!Was?!!)“	Da nicht eindeutig ermittelt werden kann, worauf sich die Frage des Drivers bezieht, ist sie mit *ask_knowledge* und nicht mit einem spezialisierten Konzept wie z.B. *ask_step* kodiert. Der Observer spricht simultan zum Driver. Sie fallen sich gegenseitig ins Wort (siehe nächste Zeile).
5	*P1.explain_finding*	„(!!Dann‿ist’s!!)‿erledigt. *Organize Imports*,‿dann‿it’s‿erledigt. (.) Behaupt’ ich mal.“	*P2* erklärt, warum *Organize Imports* aufgerufen werden sollte. An dieser Stelle ist es nicht möglich, zweifelsfrei zu ermitteln, ob *P2* wirklich davon überzeugt ist, dass seine Aussage stimmt. Die Wendung „Behaupt’ ich mal“ könnte auch darauf hindeuten, dass es sich um eine Vermutung von *P2* handelt (siehe hierzu auch S. 292).

[a] In der IDE Eclipse können mehrere Dateien gleichzeitig zum Editieren geöffnet sein. Auf diese kann über so genannte Tabs oder auch Registrierkarten zugegriffen werden.

[b] Bei dem angesprochenen *Organize Imports* handelt es sich um eine Funktionalität der IDE Eclipse (http://help.eclipse.org/help33/index.jsp?topic=/org.eclipse.jdt.doc.user/reference/ref-22.htm (Abruf: 02.05.2011)).

Tab. 7.38: Episode aus Sitzung PR2.1 (12:18:36–12:18:52). Im Editor werden Fehler (genau genommen Defekte) angezeigt. Das Paar versucht diese zu verstehen und zu beheben.

Weitere Anmerkungen zu *hypothesis*-Konzepten

Genau genommen unterscheidet die BS/BKM Äußerungen in Hinsicht auf die Klassen *finding* und *hypothesis* oft nur anhand der Darstellungsform, die der

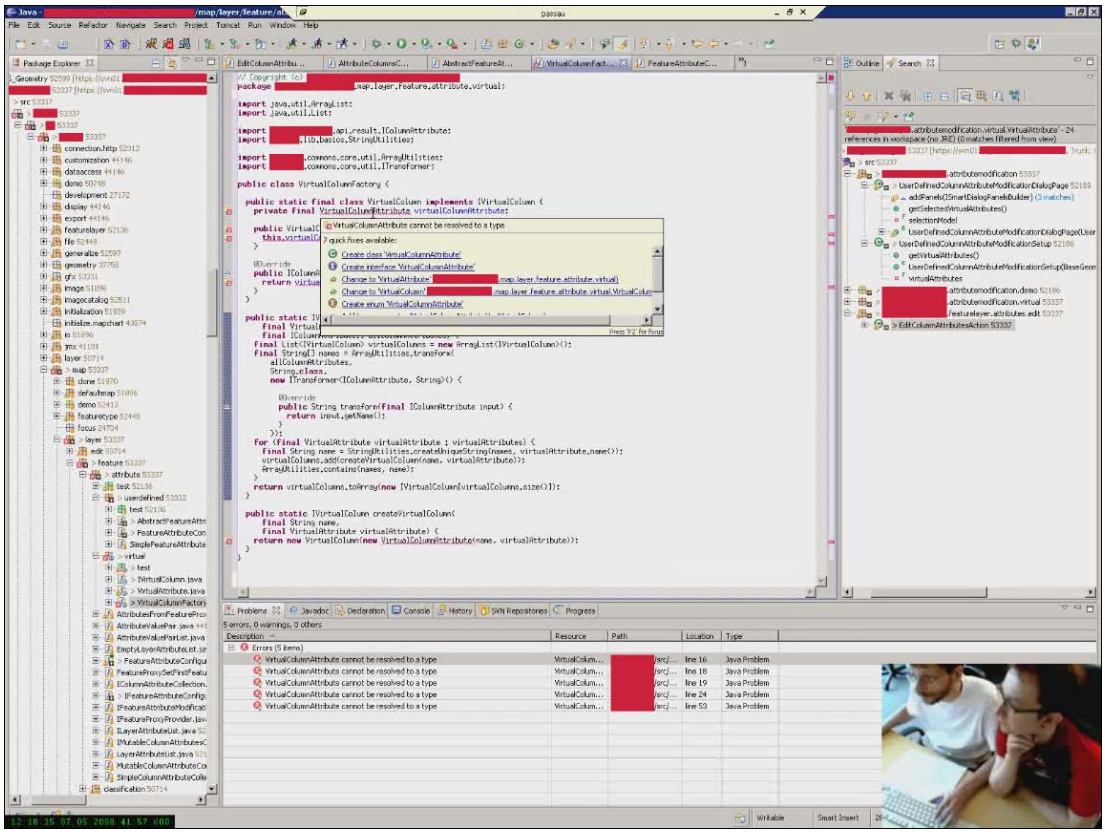


Abb. 7.12: Screenshot aus Sitzung PR2.1 (12:18:33). Die Sitzung läuft seit 41 Minuten und 57 Sekunden. Der Observer *P2* (rechts) kommentiert die Fehlermeldung (siehe auch erste Zeile in Tabelle 7.38). Teile des Kodes wurden entsprechend der Vertraulichkeitsvereinbarungen durch rote Balken unlesbar gemacht.

Sprecher verwendet, und nicht anhand von Fakten, die zeigen, ob es sich überhaupt um einen in der aktuellen Situation als wahre Information klassifizierbaren Äußerungsinhalt halten kann oder ob eigentlich nur eine Hypothese möglich ist. Anders ausgedrückt: Selbst wenn aus dem Kontext klar sein sollte, dass der Sprecher nicht wissen kann, ob seine Äußerung richtig ist, wird sie, falls er sie als richtig darstellt, nicht mit *hypothesis*, sondern mit *finding* annotiert. Dieser Umstand muss bei weiterführenden Untersuchungen berücksichtigt werden.

7.6.3 *standard of knowledge*-Konzepte

Fokus der *standard of knowledge*-Konzepte

Wie bereits auf S. 151 erläutert, werden mit *standard of knowledge*-Konzepten solche Äußerungen adressiert, in denen der Wissensstand einer Person – in der Regel bez. eines bestimmten Themas – angefragt (*ask*) oder von der Person selbst dargestellt (*explain*) wird. Es wird also das individuelle Wissen eines einzelnen Entwicklers verhandelt, nicht das des Paares als Ganzes. Von der BS/BKM

wird somit zwischen Wissen und Wissensstand unterschieden, wobei der Wissensstand in gewisser Weise eine Teilmenge des Wissens einer Person (im Sinne von W_{BS}^{-}) darstellt. Er umfasst die Einschätzungen einer Person darüber, was sie bzw. in welcher Weise sie etwas zu wissen, nur teilweise zu wissen oder auch gar nicht zu wissen meint. Somit handelt es sich bei Verbalisierungen vom Typ *explain_standard of knowledge* um wahrhaftige Äußerungen[168], in denen es nicht primär darum geht, das vom Wissensstand adressierte Wissen bzw. die vom Wissensstand adressierte nicht vorhandene oder nur teilweise vorhandene Information selbst zu transferieren.

Das Wissenstandsäußerungen durch die BS/BKM explizit sichtbar gemacht werden können, liegt an der Beobachtung, dass sie für eine effektive Zusammenarbeit von besonderer Bedeutung sind. Sie dienen in der Regel einem der folgenden Zwecke:

1. **Vorbereitung eines potentiellen Wissenstransfers (*preparation knowledge transfer* (PK))**: Ein Entwickler hält fest, welche in der Regel gerade relevanten Informationen ihm derzeit zumindest in Teilen fehlen und in einem nachfolgenden Schritt evtl. von seinem Partner zu liefern wären.[169] Für eine Klassifizierung als PK ist es nicht entscheidend, ob wirklich ein Transfer von Wissen anvisiert oder möglich ist bzw. letztendlich durchgeführt wird. Hierbei gilt: Direkt an den Partner adressierte Fragen nach solchem Wissen, was nicht als Wissen über das Vorhandensein oder Fehlen von Wissen betrachtet werden kann, sind nicht mittels *standard of knowledge* zu konzeptualisieren, sondern über Elemente der Verbklasse *ask*, im Kontext der UK über *ask_knowledge*. Details hierzu werden auf S. 340 erläutert.

   Es konnten folgende Untertypen beobachtet werden:

   (a) **Planungsgrundlagen klären (*clarifying planning criteria* (PK_{CP}))**: Ein Entwickler erklärt, inwieweit er ausreichend Wissen besitzt, um anstehende Entscheidungen fällen zu können.

   (b) **Durchführungsgrundlagen klären (*clarifying execution criteria* (PK_{CE}))**: Ein Entwickler erklärt, inwieweit er ausreichend Wissen besitzt, um eine anstehende Tätigkeit (in der Regel im Sinne eines *step*s) durchzuführen. In Folge einer solchen Erklärung kann es dazu kommen, dass gar nicht erst versucht wird, fehlendes Wissen zu transferieren, sondern stattdessen ein Rollenwechsel eingeleitet wird. Mit diesem wird das Ziel verfolgt, dem allem Anschein nach besser informierten Entwickler die Rolle des Drivers zukommen zu lassen.

[168] Im Rahmen der BS wird nicht berücksichtigt, dass es denkbar ist, dass ein Entwickler seinen Partner absichtlich in Hinsicht auf seinen Wissensstand fehlinformieren will.

[169] Hierzu soll auch der „entartete“ Fall zählen, dass ein Entwickler artikuliert, dass er alles zu einem bestimmten Thema/Aspekt weiß.

   (c) **Anderweitig fehlendes/vorhandenes Wissen klären (*other* (PK_o))**: Nicht jede Äußerung bez. des Wissensstands einer Person wird in Hinblick auf gerade anstehende Entscheidungen oder als nächstes durchzuführende Aktivitäten gemacht. PK_o fasst alle anderweitigen „Wissenstransfervorbereitungen“ zusammen.

2. **Wissenstransfer zurückweisen (*rejecting knowledge transfer* (RK))**: Eine Frage nach Informationen (in der Regel ein *ask_knowledge*) wird nicht beantwortet. Stattdessen wird klargestellt, dass man nicht genug Wissen besitzt, um irgendeine Art von Antwort geben zu können.

3. **Erfolg eines Wissenstransfers bestätigen (*acknowledging knowledge transfer* (AK))**: Ein Wissensadressat hält fest, inwieweit ein vorher durchgeführter Wissenstransfer erfolgreich oder auch nicht erfolgreich war. Auch Äußerungen, über die klar gemacht wird, dass ein Wissenstransfer gar nicht notwendig gewesen wäre, z.B. weil die übermittelten Informationen schon bekannt waren, werden AK zugerechnet.

Beispiele für die unterschiedlichen Typen von *explain_standard of knowledge*-Äußerungen sind in den Tabellen 7.39 und 7.40 gelistet. In Tabelle 7.41 werden dann diejenigen ursächlichen Bedingungen (siehe axiales Kodieren auf S. 64) detaillierter erläutert, für die beobachtet werden konnte, dass sie zu *explain_standard of knowledge*-Äußerungen führen können.

#	Zweck	Äußerung	Kontext/Kommentar
1	PK_{CP}	*P2*: „Wie die Fassade aussieht, weiß ich aber noch nicht. D[i]ss is' mir noch nicht, noch nicht (.) klar.“	Nachdem *P2* erläutert hat, dass er davon ausgeht, dass es das Ziel sein müsste, durch die Einführung einer Fassade[a] bestimmte Aspekte nach außen unsichtbar zu machen, macht *P2* klar, dass er im Augenblick nicht weiß, welche Gestalt diese Fassade idealerweise haben sollte.
2	PK_{CE}	*P2*: „(~)‿(~magst)‿du‿kurz‿nachschauen? Ich‿hab'‿keine‿Ahnung‿wie‿ ich‿da (!...!) <*ran kommen kann*>. Ich weiß es nicht auswendig.“	*P2* ist dabei, ‚online' nachzusehen, welche Kennung (ID) der gerade bearbeitete Task besitzt.[b] Unvermittelt weist er darauf hin, dass er gar nicht weiß, wie er zu dieser Kennung kommen kann. Er bietet an, die Driver-Rolle abzugeben. Bei diesem Teil der Äußerung handelt es sich um die Formulierung einer Strategie vom Typ DPR (siehe S. 217).
3	PK_o	*P2*: „Was ich irgendwo noch nicht ganz verstanden hab, (!...!)'“	*P2* ist dabei, sich einen Überblick über eine bestehende Kasse zu verschaffen, wobei er wenige Sekunden, nachdem er sie geöffnet hat, zum Ausdruck bringt, dass er etwas nicht versteht.
4	*RK*	*P2*:„Keine Ahnung, was das Ding macht.“	P2 kann die Frage danach, was einer bestimmten Funktion übergeben werden muss (*P1*: „Der braucht dann `All`, (..) vermutlich?“), nicht beantworten. Stattdessen macht er klar, dass er diesbezüglich über kein Wissen verfügt.
5	*AK*	*P2*: „In [...] <**Teil eines Paketnamens**>`.pro` ist das `VirtualAttribute`? O.K.“	Nachdem *P1* erklärt hat, in welchem Paket eine bestimmte Klasse zu finden ist („Das `VirtualAttribute` ist hier in `pro` <*zeigt auf den Bildschirm*>.“), wiederholt *P2* die Information in eigenen Worten. Er macht damit klar, was er verstanden hat. Dem Tonfall nach scheint ihn die Information des Partners zu erstaunen. Das „O.K.“ ist allem Anschein nach als „O.K., konnte Dir soweit folgen“ intendiert.
6	*AK*	*P2*: „(O.K., versteh' ich jetzt gar nicht, aber gut.) Ich hab's nicht verstanden, aber gut.“	Nachdem *P1* dargelegt hat, warum er bestimmte Änderungen vorgenommen[c] hat (*explain_knowledge*), gibt sein Partner zum Ausdruck, dass er diese Erklärung nicht versteht. Allem Anschein nach scheint er aber auch nicht an weiteren Erklärungsversuchen interessiert (Indiz hierfür: Nachsatz „aber gut“).

[a] Hier wird das Entwurfsmuster Fassade [70] referenziert.
[b] Siehe hierzu auch das Beispiel zur Entität „Konventionen“ in Tabelle 7.35.
[c] Siehe Zeile 6 in Tabelle 7.45.

Tab. 7.39: Beispiele aus Sitzung PR2.1 für die Verwendung des Konzeptes *explain_standard of knowledge*.

#	Zweck	Äußerung	Kontext/Kommentar
1	PK_{CP}	PR2.1 *P2*: „(~Das◡ ist◡das), was ich noch nicht seh'. Weiß ich nicht. Ich weiß es einfach nicht.“	*P2* ist dabei zu erklären, wie er bestimmte Teile des bestehenden Kodes ändern/refaktorisieren würde (hier nicht dargestelltes *propose_design* + *explain_finding*). Dabei kommt er zu einem Aspekt, der ihm selbst noch nicht klar ist.
2	PK_{CE}	PR1.1 *P1*: „Vor allem die Frage ist (!...!) (.) Dafür müssen wir erst mal alle Unit-Tests laufen lassen die (~die da) geschrieben haben. (.) Ich weiß ehrlich gesagt noch nicht mal (.) wie die Ukrainer ihre Unit-Tests (~).“	In Bezug auf einen zuvor vom Partner geäußerten Vorschlag zur Refaktorisierung verweist *P1* auf einen in diesem Fall durchzuführenden Arbeitsschritt (Ausführen von Unit-Tests; *propose_step*[a]). Er betont, dass ihm bezüglich dieses Tasks Informationen fehlen.
3	PK_o	PR1.1 *P1*: „((#translate◡ each◡friend's◡id◡to◡ ids#)).◡Da war'n wir doch schon mal.◡Ich hab' hier echt keinen Überblick gerade wo wir sind.◡(~Ach da) (!!...!!)“	Der Observer liest eine gerade im Quellcode sichtbare Kommentarzeile vor (erste Teiläußerung) und gibt in diesem Zusammenhang zu verstehen, dass er (kurzzeitig) den Überblick darüber verloren hat, ob bzw. wo im Kode weitere Änderungen vorgenommen werden sollten. Das schwer verständliche „Ach da“ am Ende deutet darauf hin, dass er den Überblick im Laufe der Äußerung zurückgewonnen haben könnte.
3	PK_o	PR1.1 *P1*: „Ich find' das alles auch sehr verwirrend.◡Ich mach mir hier jedes mal 'ne Zeichnung, damit ich's überhaupt (~) (~verstehen kann).“	*P1* macht ein Anmerkung in Hinsicht auf seine Probleme, die Funktionsweise einer bestimmten `if`-Anweisung zu verstehen.
4	AK	PR2.1 *P2*: „Hm, ich weiß. D[a]s ist furchtbar.“	Nachdem *P1* Eigenschaften einer Klasse erläutert hat, gibt *P2* zu verstehen, dass ihm diese schon vorher bewusst waren. Er ergänzt diese Äußerung um eine Beurteilung (siehe S. 319).

[a] Es handelt sich um einen Arbeitsschritt mit strategischem Charakter (siehe S. 230).

Tab. 7.40: Weitere Beispiele für die Verwendung des Konzeptes *explain_standard of knowledge*. Achtung: Die in der Spalte „Äußerung“ gelisteten Beispiele sind zum Teil in unterschiedlich zu kodierende Teiläußerungen aufzubrechen. In der Spalte „Kontext/Kommentar“ wird ggf. darauf hingewiesen.

standard of knowledge-Konzepte und ihre speziellen Eigenschaften

Es konnten zwei *standard of knowledge*-Konzepte aus den Daten extrahiert werden: *explain_standard of knowledge* und *ask_standard of knowledge*. Eine grobe Beschreibung ist in Abbildung 7.3 zu finden. Details werden nachfolgend erörtert.

1. **Wissensstand abfragen:** Es konnte beobachtet werden, dass Entwickler ihre Partner danach befragen, wie viel diese bez. eines bestimmten Aspektes oder in Hinsicht auf ein bestimmtes Thema wissen, wobei es den Entwicklern im Rahmen ihrer Fragen erst einmal nicht unmittelbar darum geht, mehr Wissen zu dem angesprochenen Aspekt/Thema zu erlangen. Das Erlangen von Wissen ist, falls überhaupt erreichbar, eher ein nachgelagertes Ziel, genauso wie das Vermitteln von Wissen. Fragen dieser Art sind mit *ask_standard of knowledge* zu kodieren. Zur Verdeutlichung betrachte man die nachfolgende Episode aus Sitzung PR1.1. Hier möchte der Driver *P1* herausbekommen, ob sein Partner das als nächstes zu überarbeitende PHP-Skript kennt. Er fragt:

   > „Du kennst das Skript überhaupt gar nicht, oder?“

   P2 antwortet:

   > „Ja, ich hab’s mir vorhin angeguckt und es ist eigentlich nicht weiter kompliziert.“[170]

   Die Hauptintention des Observers besteht allem Anschein nach darin, seinem Partner klar zu machen, dass er über ausreichend Informationen bez. des Skriptes verfügt. Die Beurteilung der Komplexität ist dabei Mittel zum Zweck. Im Rahmen der hier vorgestellten Analysen wurde der erste Teil der Äußerung mit *explain_standard of knowledge* und der zweite, da es sich um eine Bewertung handelt – hierauf wird im Rahmen der Erläuterungen zur Klasse *knowledge* auf S. 319 näher eingegangen – mit *explain_knowledge* kodiert.

2. **Paraphrasieren von übermitteltem Wissen:** Um deutlich zu machen, dass bzw. inwieweit gerade übermitteltes Wissen verstanden wurde, kommt es vor, dass dieses (in eigenen Worten) wiederholt wird (*explain_standard of knowledge*-Äußerung vom Typ *AK*). Hierbei kann es dazu kommen, dass Teilaspekte verbalisiert werden, die vorher nicht explizit ausgesprochen worden waren. Im Rahmen der BS/BKM sollte auch hier *explain_standard of knowledge* zum Einsatz kommen (Details hierzu werden im Zusammenhang

[170] Wie in vielen anderen Situationen ist die Äußerung auch hier widersprüchlich formuliert. Erst stimmt der Sprecher der Aussage des Partners zu, um nachfolgend klar zu machen, dass er dies eben nicht tut. In der Regel ist es aber trotzdem ohne Probleme möglich festzustellen, was der Sprecher gemeint hat. Warum es zu solchen Phänomen kommt, liegt außerhalb des Fokus’ der vorliegenden Arbeit.

Ursächliche Bedingung	**Beschreibung einer Beispielsituation**	**Reaktion (Zweck)**
Übermittlung von Bestandswissen	Der Observer *P2* erklärt seinem Partner, wie er aus einer bestimmten Applikation heraus mit der Maus Text in die Zwischenablage kopieren kann (*explain_knowledge*).	*P1*: „(Ach so), also die (~Mitte).“ (AK)[172]
Übermittlung von Erkenntnissen	Der Observer *P2* erläutert seine gerade gewonnene Erkenntnis zum bestehenden Kode: „Ahh hier bei `set` wird es da gemacht. Aber da wird's auf beiden Seiten gemacht.“	*P1*: „Verstehe.“ (AK)[173]
Wissensanfrage	Siehe Beispiel 4 in Tabelle 7.39.	
Wissensstandsanfrage	Nachdem *P1* eine Erkenntnis erläutert hat, fragt er seinen Partner: „Weißt Du, was ich meine?“	*P2*: „Nee“ (VW_o)
Handlungsvorschlag	Nachdem der Driver *P1* einige Kodeänderungen vorgenommen hat, macht er noch einen Handlungsvorschlag: „Dann kommentieren wir das noch ein bisschen.“	*P1*: „Das hier bedeutet ähmm (!...!) (...). Ja was soll das denn eigentlich?“[174] (PK_{CE})
Nächste Tätigkeit wird begonnen	Das Paar ist dabei, Kodestellen zu analysieren, an denen eine bestimmte Funktion aufgerufen wird. Hierbei kommt es dazu, dass sie sich auch die Funktion selbst ansehen. Danach will *P1* zur nächsten (im Vorfeld via Volltextsuche ermittelten) Aufrufstelle wechseln. Er verbalisiert dieses Vorhaben allerdings nicht explizit, sondern gibt stattdessen zu verstehen, dass er sich nicht erinnern kann, welche Stelle als nächstes zu inspizieren ist (siehe rechte Spalte).[175]	„Jetzt müssen wir mal eben (!...!).‿Jetzt hab' ich nämlich schon wieder vergessen, wo eigentlich unsere Suche angeschlagen hatte.“ (PK_{CE})
		Fortsetzung auf der nächsten Seite.

[172] Die Abgrenzung zu *agree_knowledge* wird auf S. 341 erläutert.

[173] Die Abgrenzung zu *agree_finding* wird auf S. 309 erläutert.

[174] In diesem Beispiel sind derjenige, der einen Handlungsvorschlag macht, und derjenige, der Verständnisprobleme in Bezug auf den Vorschlag äußert, ein und dieselbe Person. Das muss im Allgemeinen nicht so sein. Achtung: Bei der Äußerung handelt es sich nicht um eine an den Partner gerichtete Frage, sondern um eine fast parallel zum Denkprozess verbalisierte Erkenntnis dahingehend, dass etwas vermeintlich Verstandenes doch noch unklar ist (*P1* wendet sich hierbei in keiner Weise seinem Partner zu). Gleichwohl versucht sich *P1* nachfolgend darin, eine Antwort zu geben.

[175] Während er spricht, beginnt er damit, die Suche erneut aufzurufen.

		Fortsetzung der Tabelle der vorherigen Seite.
Artefakt wird durchgesehen	Der Driver *P1* öffnet eine bestimmte (nicht vom Paar implementierte) PHP-Datei an einer bestimmten Stelle, damit sich das Paar den Programmkode um die Stelle herum genauer ansehen kann. Kurz nachdem der Kode im Editor erschienen ist, weist der Observer darauf hin, dass er nicht versteht bzw. nicht weiß, warum es dort eine Funktion mit dem Namen `updateUserData` gibt (siehe rechte Spalte).	„`updateUserData`? (~Was‿ist‿denn‿das) für'n Mist? (..) Sagt mir nichts, aber gut (~~).“ (VW_o)

Tab. 7.41: Typen von Situationen (Spalte 1), die zu *explain_standard of knowledge*-Reaktionen führen können. In der zweiten Spalte wird jeweils ein diesbezügliches Beispiel erläutert, in der dritten Spalte das zugehörige *explain_standard of knowledge* wiedergegeben. Die Beispiele stammen aus Sitzung PR1.1. Die hier erörterten Situationen sind im Allgemeinen weder hinreichend noch notwendig dafür, dass es zu einem *explain_standard of knowledge* kommt. So ist beispielsweise denkbar, dass Entwickler auf ein *explain_standard of knowledge* des Partners mit einem eigenen *explain_standard of knowledge* reagieren oder dass Entwickler zum Ausdruck bringen, dass sie (einen bestimmten Teil) einer Hypothese des Partners nicht verstehen.

mit der Klasse *knowledge* auf S. 340 erläutert). Achtung: Paraphrasierungen konnten nicht nur im Zusammenhang mit Äußerungen vom Typ *explain_knowledge*[171] beobachtet werden, sondern auch im Zusammenhang mit Vorschlägen wie z.B. *propose_stategy* (siehe Tabelle 7.21 bzw. Abbildung 7.6).

Einen Spezialfall bilden solche Äußerungen, in denen ein Entwickler eine Wissensäußerung zu Ende führt, die vom Partner nicht vollständig formuliert wurde – sei es, weil dieser es nicht für nötig erachtete oder weil er unterbrochen wurde. Es konnte beobachtet werden, dass solche Vervollständigungen allem Anschein nach auch deshalb gemacht werden, um zu signalisieren, was verstanden wurde. In diesem Fall sind sie mit *explain_standard of knowledge* zu kodieren (siehe Beispiel in Tabelle 7.42).

Ob bzw. inwieweit es sich bei den in diesem Abschnitt beschriebenen Phänomenen evtl. nicht eher um solche vom Typ *agree_**, insbesondere *agree_knowledge* oder *agree_finding* handelt, wird auf S. 309 und auf S. 341 diskutiert.

3. **„Live“ verfolgbare Identifikation von Wissenslücken:** Im Zusammenhang mit der Klasse *finding* ist bereits erörtert worden, dass es vorkommt, dass Entwickler nicht nur das „endgültige“ Ergebnis einer kognitiven Leistung verbalisieren, sondern Teile des Denkprozesses mitsprechen

[171] Gleiches gilt potentiell auch für Äußerungen vom Typ *explain_finding*. Solche Bezüge konnten im hier diskutierten Zusammenhang allerdings bisher nicht beobachtet werden.

HHI-Kode	Äußerung	Kontext/Kommentar
P2.ask_knowledge	„Kann‿es‿zu `timestamp_last_change` größer Null kommen? (.) Ä‿ä, kleiner Null – oder gleich Null von mir aus. (.) Unter der Bedingung <*zeigt auf den Bildschirm*>.“	*P2* stellt eine Frage bez. des Wertes einer bestimmten Variable.
P1.explain_knowledge	„Hmmmm. Das ist quasi unsere Anforderung, die wir doch an diese Funktion stellen (!!...!!)“	*P1* beginnt, die Frage zu beantworten, wird aber vom Partner unterbrochen. Die von ihm angesprochene Funktion soll den zuzuweisenden Wert der Variablen bestimmen. Während des Sprechens markiert er einen noch auskommentierten Aufruf dieser Funktion (siehe Abbildung 7.13).
P2.mumble_sth	„Sollten wir das da nicht schon, damit nicht schon, also (!...!)“	*P2* unterbricht den Partner. Er formuliert seine Bemerkung aber nicht aus. Es ist unklar, was er mitteilen möchte.
P1.explain_knowledge	„Dass sie Null zurück gibt, ähm.“	*P1* geht auf die Unterbrechung nicht ein, sondern fährt nach dieser unmittelbar mit seiner Erklärung fort.
P2.agree_knowledge	„O.K., hmhm.“	*P2* stimmt unmittelbar zu.
P2.explain_standard of knowledge	„Wenn's nich', wenn's fehl schlägt, irgendwie.“	*P2* fährt fort und ergänzt die von *P1* geäußerte Erklärung um die Bedingung für die angesprochene Rückgabe von Null. Diese hatte *P1* nicht ausformuliert, obwohl sie ihm, wenn man den Kontext berücksichtigt, bewusst sein müsste. *P2* gibt somit zu verstehen, in welcher Weise er die Erklärung von *P1* verstanden hat bzw. was er nun bezüglich des diskutierten Aspektes zu wissen meint.[a]
P1.agree_knowledge[b]	„Genau“	

[a] An dieser Stelle wurde darauf verzichtet, das Konzept *amend_knowledge* einzuführen und anzuwenden, da es sich allem Anschein nach nicht um eine als Ergänzung intendierte Äußerung handelt.

[b] Die BKM enthält kein Konzept, mit dem Phänomene der folgenden Art sichtbar gemacht werden können: Ein Entwickler bestätigt seinem Partner, dass dieser das richtige Verständnis erlangt hat. Es wurde also auf die Einführung eines Konzeptes wie z.B. *attest_standard of knowledge* verzichtet. Somit bleibt im vorliegenden Fall nur die Möglichkeit, ein *agree_knowledge* zu annotieren – obwohl das passende Gegenstück, z.B. in Form eines *explain_knowledge*, fehlt. Weiterführende Untersuchungen sollten die BKM an dieser Stelle bei Bedarf ergänzen.

Tab. 7.42: Beispielepisode aus PR1.1 (14:14:31–14:14:51): Wissensstandsäußerung vom Typ *AK*, in der zusätzliche, nicht explizit formulierte, in diesem Fall aber schon im Vorfeld angesprochene Aspekte einer Erklärung referenziert werden. Der Zweck der Äußerung besteht darin, darzulegen, in welcher Weise die Erklärung verstanden wurde.

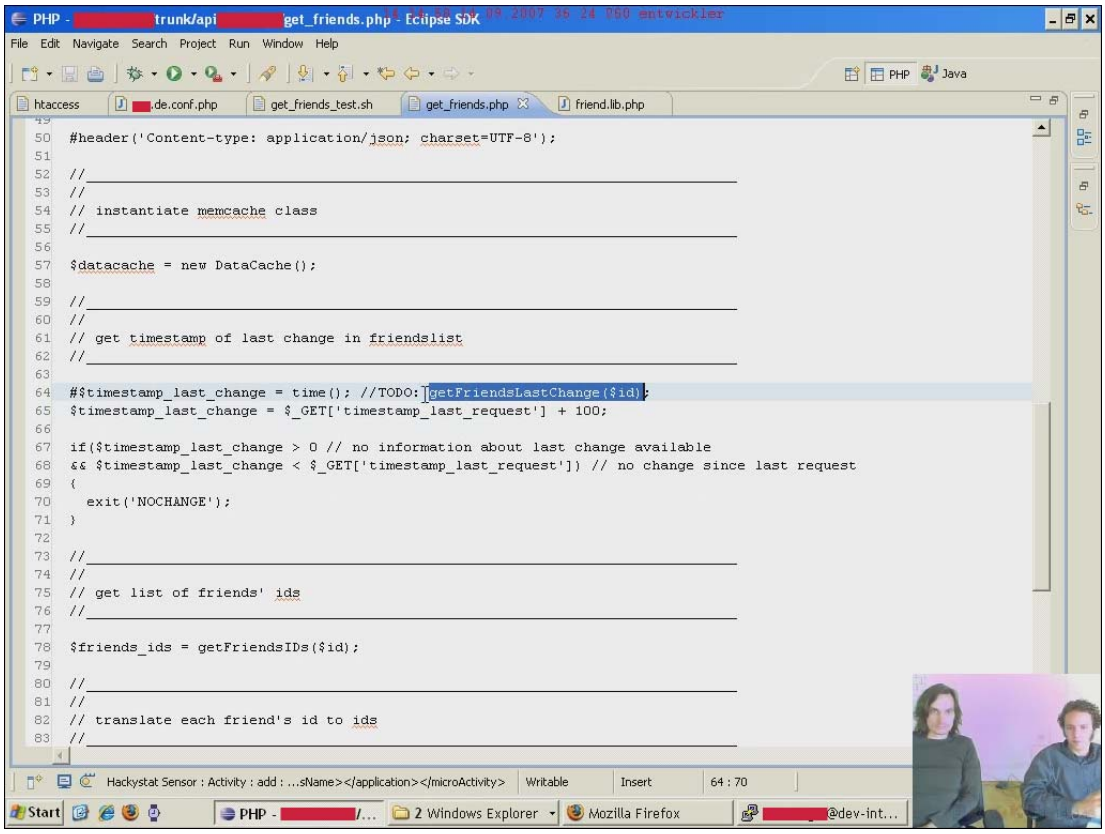


Abb. 7.13: Screenshot aus Sitzung PR1.1 (14:14:50). Der Driver (links im Bild) ist dabei, Wissen an den Observer weiterzugeben (*explain_knowledge*). Details sind in Tabelle 7.42 erläutert.

(siehe S. 268). In ähnlicher Weise konnte beobachtet werden, dass es Situationen gibt, in denen es möglich ist mitzuverfolgen, wie Entwickler zu der Einsicht kommen, dass sie bestimmte Dinge zur Zeit nicht oder nur in Teilen wissen oder verstehen. So äußert sich *P1* in Sitzung PR1.1 folgendermaßen:

> „Richtig schick wäre es natürlich, wenn wir irgendwas zurückgeben könnten, was immer (!...!) (..) hm (...) (Augenblick (~) mal)‿Was immer (!...!) (.....) Jetzt hab' ich (~irgendwo) den Faden verloren."

Solche Phänomene sollten im Rahmen der BS/BKM mit *explain_standard of knowledge* kodiert werden.

Achtung: Derartige Beobachtungen konnten im Rahmen der hier diskutierten Sitzungen nur im Zusammenhang mit Äußerungen einer bestimmten Person gemacht werden – hier allerdings mehrere Male.

4. **Erkenntnisse:** Wie im letzten Beispiel gesehen und bereits in Abbildung 7.1 dargestellt kann es sich bei Äußerungen vom Typ *ex-*

plain_standard of knowledge auch um solche handeln, in denen eine gerade gewonnene Erkenntnis formuliert wird. Sie werden im Rahmen einer BS/BKM-Kodierung gemäß den in Abbildung 7.10 dargestellten Vorfahrtsregeln trotzdem nur mit *explain_standard of knowledge* kodiert.

5. **Nachgeschobene Vorschlags- oder Entscheidungsrelativierungen**: Wie bereits erörtert werden Vorschläge oder Entscheidungen auch dann mit Elementen aus *P&P* konzeptualisiert, wenn es sich um Äußerungen handelt, die der Sprecher nicht unbedingt als vollständig richtig oder einzig angemessen deklariert (siehe z.B. die *propose_step*-Typen in Tabelle 7.5). Somit könnte man konsequenterweise festlegen, dass auch einzelne nachgeschobene Relativierungen immer als Bestandteil eines *P&P*-Vorschlags bzw. einer *P&P*-Entscheidung gesehen und kodiert werden sollten und nicht als *standard of knowledge*-Äußerung. Die BS verfährt allerdings nicht so, sondern kodiert derartige Äußerungen anhand ihrer primären Intention.[176] Besteht diese allem Anschein nach darin, dem Partner klar zu machen, dass man den vorher geäußerten Vorschlag bzw. die vorher gefällte Entscheidung auf Basis eingeschränkten Wissens (oder umgekehrt auf Basis soliden Wissens) artikuliert hat, wird sie als *explain_standard of knowledge* kodiert (siehe Beispiel in Tabelle 7.22).

6. **Indirekte Wissensstandsäußerungen:** Es konnte beobachtet werden, dass Äußerungen bez. des eigenen Wissensstands auch derart formuliert werden, dass Interesse an bestimmten Informationen kundgetan wird, ohne geradeheraus eine Wissenslücke zu artikulieren oder eine Frage an den Partner zu richten. So äußert beispielsweise *P2* in Sitzung PR2.1 in Hinsicht auf bestimmte Funktionalitäten von Subversion[177]

   „Würd' mich selber interessieren"[178]

   und der Driver *P1* in Sitzung PR1.1 merkt an

   „Ich würd's gern verstehen!"

   Auch solche Äußerungen werden im Rahmen der BS als *explain_standard of knowledge* kodiert. Die zweite Äußerung – eine Entgegnung auf ein *explain_standard of knowledge* des Partners (siehe „Artefakt wird durchgesehen" in Tabelle 7.41) – darüber hinaus auch noch mit *propose_step*, da *P1* explizit Zeit einfordert, damit er das gewünschte Verständnis erlangen

[176] Hier handelt es sich um eine der Regeln, die in nachfolgenden Untersuchungen in besonderer Weise hinterfragt werden sollten. Welches Vorgehen angemessen ist, lässt sich ohne Berücksichtigung des Blickwinkels kaum sagen.

[177] Siehe Fußnote 66 auf S. 191.

[178] In diesem Fall wurde die Wissenslücke allerdings schon einmal kurz zuvor explizit artikuliert (siehe nachfolgenden Unterpunkt „Wiederholungen von Wissensstandsäußerungen").

kann (siehe auch S. 196). Ein Indiz für diese Intention liefert die Beobachtung, dass er unmittelbar nach der Äußerung damit beginnt, sich den unverständlichen Programmkode genauer anzusehen (siehe auch Screenshot in Abbildung 7.14).

7. **Wiederholungen von Wissensstandsäußerungen:** Zeitnahe (evtl. paraphrasierte) Wiederholungen von Wissensstandsäußerungen werden im Rahmen der BS erneut mit *explain_standard of knowledge* kodiert. So ist die im vorletzten Beispiel wiedergegebene Äußerung eine Wiederholung des folgenden *explain_standard of knowledge*:

   „Wie's allerdings beim SVN ist, weiß ich nicht."

   Die hier als Wiederholung kategorisierte Äußerung ist allerdings keine reine Paraphrasierung, denn durch sie wird betont, dass es sich um eine für den Sprecher interessante Information handelt. Dieser Aspekt geht aus der ersten Äußerung nicht hervor.[179] Man sollte sich vergegenwärtigen, dass der Unterschied in der Ausrichtung der beiden Äußerungen im Rahmen der BS bzw. im Rahmen des Blickwinkels der BS verschwindet. Bei spezifischeren Fragestellungen muss er aber evtl. sichtbar gemacht werden.

   Auf Regeln im Umgang mit Wiederholungen wird noch einmal in Abschnitt 9.3.7 eingegangen.

8. **Zurückblickende Wissensstandsäußerungen:** *explain_standard of knowledge* adressiert auch solche Äußerungen, mit denen klar gemacht wird, dass bestimmte Verständnisprobleme schon länger existierten. Auch dann, wenn die Wissenslücke zum Zeitpunkt der Formulierung der Äußerung schon geschlossen ist oder vom Sprecher als nicht mehr relevant betrachtet wird. So gibt der Driver *P2* in Sitzung PR2.1 Folgendes zu verstehen:

   „Hab' ich eh schon vorher nicht verstanden."

   Hiermit kommentiert er die Durchführung des Löschens einer Kodezeile. Dass diese Kodezeile entfernt werden muss, hatten die beiden Entwickler unmittelbar zuvor einstimmig erkannt.

9. **Nachdenken signalisieren:** Auch in dem Fall, in dem ein Entwickler verbal signalisiert, dass er gerade dabei ist, über eine bestimmte Fragestellung oder einen bestimmten Aspekt nachzudenken, sollte das Konzept *explain_standard of knowledge* zum Einsatz kommen. So reagiert *P2* in Sitzung PR1.1 auf die Erkenntnisäußerung

[179] Die Einführung eines Konzeptes *amend_standard of knowledge* ist in diesem Zusammenhang nicht angebracht. Schließlich liefert die Äußerung keine neuen Informationen in Hinsicht auf den Wissensstand über Subversion.

„Ähm, damit würden wir ihn optional machen.“

mit

„Das überlege ich gerade.“

Der Driver *P2* erklärt hier, dass eine von ihm anvisierte Kodeänderung den Übergabeparameter `timestamp_last_request` optional machen würde. P1 signalisiert, dass er genau darüber gerade nachdenkt.[180]

Achtung: Derartige Phänomene sind von denen unter dem Punkt ‚„Live“ geäußerte Denkprozesse‘ auf S. 268 und unter dem Punkt ‚„Live“ verfolgbare Identifikation von Wissenslücken‘ auf S. 304 beschriebenen zu unterscheiden.

Abgrenzungen von *standard of knowledge*-Konzepten zu anderen

In diesem Abschnitt wird auf Abgrenzungen zu den Klassen *finding* und *hypothesis* sowie auf Abgrenzungen zu bestimmten *P&P*-Konzepten eingegangen. Abgrenzungen zu *knowledge*-Konzepten werden erst in Zusammenhang mit Erläuterungen der *knowledge*-Klasse ab S. 331 erörtert.

- ***explain_standard of knowledge* vs. *agree_finding*:** Jedes *agree_finding* kann, wenn man davon ausgeht, dass Entwickler nur solchen Ideen zustimmen, die sie nachvollziehen können bzw. verstehen[181], zusätzlich auch als Wissensstandsäußerung, also als *explain_standard of knowledge* interpretiert werden. Die BS/BKM wertet in diesem Zusammenhang das Aufzeigen von sprachlichen Bezügen (siehe Abschnitt 7.2 über die zentrale Prinzipien im Umgang mit HHI-Konzepten) stärker als die Einhaltung der Vorfahrtsregeln (siehe Abbildung 7.10), wobei letztere sowieso vor allem in Hinblick auf initiative Verben definiert wurden. Äußerungen, die als Ganzes primär in die Subklasse *agree_finding* fallen, werden also auch nur mit diesem Konzept annotiert (siehe z.B. Tabelle 7.32). Weiterführende Untersuchungen müssen, falls es notwendig erscheinen sollte, andere Regeln festlegen.

  Achtung: Trotz dieser Festlegung kann es natürlich im Rahmen einer BS-Kodierung sinnvoll sein, die Reaktion eines Entwicklers auf ein *explain_finding* mit *explain_standard of knowledge* zu kodieren. Dies ist

[180] Genau genommen baut der ursprüngliche Designvorschlag von *P1* gerade darauf auf, dass der Parameter in bestimmten Situationen weggelassen werden muss/kann (*P1* expliziert dies). *P2* stimmt diesem Vorschlag sofort zu, schiebt aber trotzdem wenige Sekunden später die Frage „`time` (!...!).‿(~Ist)‿`timestamp_last_request` nicht optional?“ hinterher. Diese beantwortet *P1* wie oben angegeben. Vor diesem Hintergrund ist unklar, worüber *P2* genau nachdenkt.

[181] Diese Annahme ist sicherlich nicht allgemeingültig. Die Berücksichtigung von entgegengesetzten Phänomenen liegt allerdings nicht im Fokus der BS/BKM.

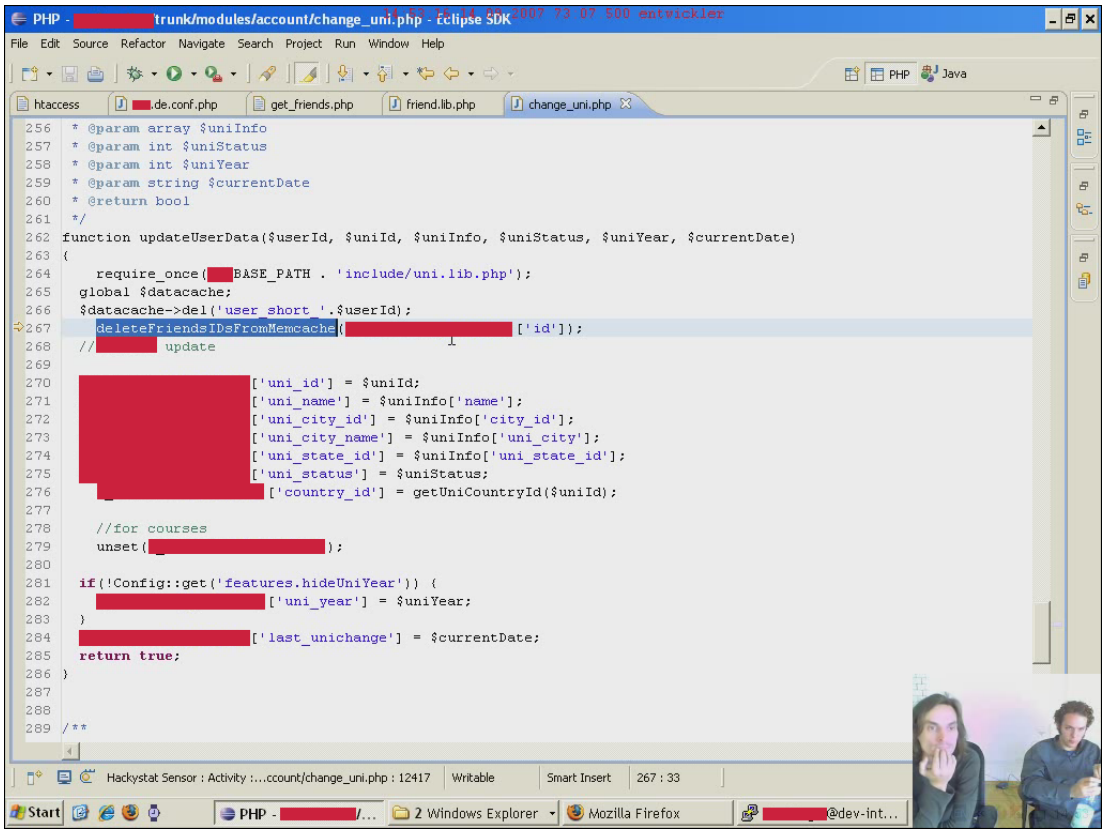


Abb. 7.14: Screenshot aus Sitzung PR1.1 (14:53:16). Der Driver *P1* (links) ist dabei, sich eine bestimmte Funktion genauer anzusehen. Diese Funktion ist innerhalb der Sitzung nicht verändert worden. Das Ziel von *P1* besteht darin, die Bedeutung dieses Kodes zu verstehen. Details sind auf S. 307 erläutert.

z.B. dann der Fall, wenn ein Entwickler mit seiner Äußerung einzig oder zumindest mit nachvollziehbar gleicher Dringlichkeit (explizit) zum Ausdruck bringt, inwieweit er dem Partner in Hinsicht auf seine Erkenntnis folgen konnte (siehe Beispiel zu „Übermittlung von Erkenntnissen“ in Tabelle 7.41). In diesem Zusammenhang kann es auch angebracht sein, eine Äußerung in zwei Teiläußerungen zu zerlegen, ein *agree_finding* und ein *explain_standard of knowledge*.

– ***explain_standard of knowledge* vs. *disagree_finding*:** Diese Abgrenzung verläuft analog zu der von *agree_finding*.

– ***explain_standard of knowledge* vs. *explain_finding*:** Wie bereits erläutert kann es sich bei Wissensstandsäußerungen um Verbalisierungen von gerade gewonnenen Erkenntnissen handeln. Trotzdem werden sie im Rahmen der BS nicht mit *explain_finding*, sondern ausschließlich mit *explain_standard of knowledge* annotiert. Dies gilt insbesondere für Äußerungen wie „Ich hab’ ’ne Idee“ (PR2.1), mit denen ein Entwickler klar macht,

dass er eine Erkenntnis in Hinsicht auf einen bestimmten (aber vom Partner nicht unbedingt genau zu identifizierenden) Aspekt[182] erlangt hat, ohne dass er aber die eigentliche Erkenntnis selbst verbalisiert – also für Äußerungen, mit denen ein Entwickler einzig und allein zu verstehen gibt, dass sich seine Wissensbasis (in Hinsicht auf einen bestimmten Aspekt) erweitert hat.

– ***explain_standard of knowledge* vs. *(dis)agree_*$O_{P\&P}$:** Genauso wie $P\&P$-Vorschläge können auch Zustimmungen zu diesen bzw. Ablehnungen von diesen derart formuliert sein, dass ein Zuhörer erkennen kann, dass der Sprecher nicht uneingeschränkt hinter seiner Bewertung steht. Beispielsweise konnte beobachtet werden, dass Bewertungen unter Zuhilfenahme von Wissensstandsäußerungen abgegeben und somit gleichzeitig auch relativiert werden (siehe Beispiel auf S. 176). In Hinsicht auf solche Phänomene stellt sich die Frage, inwieweit (auch) das Konzept *explain_standard of knowledge* zum Einsatz kommen sollte. Die BS/BKM stellt hier zwei, zumindest in Teilen bereits erläuterte Leitlinien zur Verfügung:

  1. Da im Rahmen von BS-Kodierungen das Ziel verfolgt werden soll, sprachliche Bezüge kenntlich zumachen (siehe Abschnitt 7.2) und darüber hinaus $P\&P$-Kodierungen Vorrang vor UK-Kodierungen haben (siehe S. 163), sollte im Allgemeinen nur ein *(dis)agree_*$O_{P\&P}$ annotiert werden.
  2. Da aber auch Situationen ähnlich der unter dem Punkt „*explain_standard of knowledge* vs. *agree_finding*“ beschriebenen vorstellbar sind, kann es angemessen sein, zusätzlich oder sogar ausschließlich *explain_standard of knowledge* zu annotieren. Hierbei sollte anlog zu „*explain_standard of knowledge* vs. *agree_finding*“ vorgegangen werden.

– ***explain_standard of knowledge* vs. *propose_hypothesis*:** Den Vorfahrtsregeln folgend, muss eine Äußerung, falls beide hier angesprochenen Konzepte zur Konzeptualisierung geeignet scheinen, mit *propose_hypothesis* kodiert werden. So handelt es sich beispielsweise bei der Äußerung

  „Vielleicht‿habe‿ich‿(~sogar‿selber‿welche‿geschrieben),
  weiß ich gar nicht.“ (PR2.1),

  in der sich der Sprecher auf im Vorfeld implementierte, im Laufe der Sitzung noch nicht betrachtete, aber aktuell evtl. zu modifizierende Unit-Tests bezieht, nicht um ein *explain_standard of knowledge*, sondern um ein *propose_hypothesis*, genauer gesagt um eines vom Typ H_{uck}. Der Zusatz „weiß

[182] Bei einem Aspekt kann es sich z.B. um ein im Vorfeld identifiziertes Problem handeln.

ich gar nicht“ wurde hierbei nicht als eigenständige Wissensstandsäußerung, sondern als eine Betonung des Vermutungscharakters des ersten Teils interpretiert. Das Beispiel macht aber deutlich, dass es mittels der BS nicht möglich ist, immer eine über jeden Zweifel erhabene Kodierung der Daten zu erhalten. Schließlich wäre es im vorliegenden Beispiel auch nicht völlig unplausibel, den zweiten Teil der Äußerung abzutrennen und mit *explain_standard of knowledge* zu kodieren. Deshalb sei hier noch einmal darauf hingewiesen: Derartige „Uneindeutigkeiten“ offenbaren nicht eine grundsätzliche Schwäche der BS/BKM. Die BS/BKM befördert in solchen Situationen vielmehr eine differenzierte Betrachtung von Phänomenen. Die Ergebnisse dieser Untersuchungen sollten dann mittels geeigneter Mittel (Eigenschaften von Konzepten, neuen Konzepten, Memos etc.) festgehalten werden. Außerdem: Die „Ränder“ eines komplexeren Konzepts (Kategorie) sind in der Regel sowieso kaum genau zu fassen. Beispielsweise spricht Ludwig Wittgenstein im Rahmen von Erläuterungen zur Bedeutung des Begriffs Spiel von einem „Begriff mit verschwommenen Rändern“ [223, S. 615]. Auf diese Punkte wird noch einmal in Abschnitt 9.3.1 eingegangen.

7.6.4 *gap in knowledge*-Konzepte

Fokus der *explain_gap in knowledge*-Konzepte

Im Laufe der hier dargestellten Untersuchungen wurde die These aufgestellt, dass diejenigen Situationen, in denen von einem Entwickler festgestellt oder festgehalten wird, dass er und sein Partner ein und dieselbe den weiteren Arbeitsfortschritt behindernde Wissenslücke haben, für das Verständnis der PP von besonderem Interesse sein könnten. Es wurde entschieden, dass solche Äußerungen durch eine eigene Objektklasse, nämlich *gap in knowledge* behandelt werden sollen. Dieser Beschluss wurde gefasst, obwohl derartige Situationen nur wenige Male beobachtet werden konnten.

Folgende kurze Episode liefert ein Beispiel für die Verwendung der *gap in knowledge*-Konzepte: In Sitzung ST1.1 reagiert der Observer *P2* auf die Situationszusammenfassung

> „Wir wissen nicht, ob das, was wir in der Datei ändern (..), bis zum JBoss[183] durchdringt“ (*explain_gap in knowledge*)

mit

> „Genau“ (*agree_gap in knowledge*).

[183] Siehe Fußnote 15 auf S. 87.

gap in knowledge-Konzepte und ihre speziellen Eigenschaften

Bisher konnten nur zwei Typen von Phänomenen identifiziert werden, die in die Klasse *gap in knowledge* fallen: *explain_gap in knowledge* und *agree_gap in knowledge*. Wie alle anderen HHI-Konzepte wurden sie bereits in Abbildung 7.3 eingeführt. Über die dort gemachten Darstellungen hinaus ist festzuhalten, dass man zumindest zwischen zwei Typen von *explain_gap in knowledge*-Äußerungen unterscheiden kann: Zum einen Situationszusammenfassungen und zum anderen eine Wissenslücke deutlich machende Eigeneinschätzungen. So kann es sich bei Äußerungen vom Typ *explain_gap in knowledge*, wie bereits oben gesehen, um Aggregationen bereits ausgetauschter Informationen handeln – zum Teil auch derart, dass der Informationsaustausch nicht nur zusammenfassend wiedergegeben, sondern in Hinsicht auf das Ergebnis desselben analysiert wird. Dies passiert z.B. dann, wenn Wissenslücken im Vorfeld nur implizit zugegeben wurden. Außerdem konnten *explain_gap in knowledge*-Äußerungen beobachtet werden, bei denen die Verbalisierung einer *gap in knowledge*-Erkenntnis darauf aufbaut, dass der Partner zuvor eine auf einen bestimmten Aspekt zielende (evtl. auch nur rhetorisch gemeinte) Frage formuliert oder eine *explain_standard of knowledge*-Äußerung angebracht hat, mit der er eine bestimmte Wissenslücke offenbarte. So beantwortet der Observer *P2* nach einer kurzem Pause von ca. drei Sekunden, in der sich die Entwickler fragend ansehen, die Äußerung

> „Die Frage ist (.): Wer stellt mir 'n `Topic`[184] zur Verfügung? (.) <*P2 wirft ein: '[Ä], JBoss'*> JBoss, ja! JBoss stellt ihn zur Verfügung. Warum macht er das? In welcher (.) Datei steht das? Und wird diese Datei von XDoclet[185] generiert oder nicht?" (*P1.ask_knowledge*[186])

mit

> „Sehr gute Frage. Alles wieder vergessen" (*P2.explain_gap in knowledge*).

P2 stellt mit dieser Äußerung nicht nur fest, dass er die Frage seines Partners nicht beantworten kann, sondern betont darüber hinaus, dass das Paar diese Information benötigt, um voran kommen zu können.[187] Derartige Äußerungen werden nicht mit *explain_standard of knowledge*, sondern mit *explain_gap in knowledge* annotiert.

[184] `javax.jms.Topic` (siehe Tabelle 7.4)
[185] Siehe Fußnote 23 auf S. 23.
[186] Die Äußerung sollte in mehrere Teile, zumindest zwei *ask_knowledge*-Phänomene, ein *explain_knowledge*-Phänomen und ein *agree_knowledge*-Phänomen aufgeteilt werden. Aus Platzgründen wurde hier auf eine solche Aufteilung verzichtet.
[187] Diese Deutung wird vor dem Hintergrund der eingesetzten Techniken und deren hier angestrebten Verwendungsszenarien vollständig nachvollziehbar.

Abgrenzungen von *gap in knowledge*-Konzepten zu anderen

In diesem Abschnitt wird auf die Abgrenzung zur Klasse *standard of knowledge* eingegangen. Abgrenzungen zu *knowledge*-Konzepten werden erst in Zusammenhang mit Erläuterungen der *knowledge*-Klasse ab S. 331 erörtert.

- ***explain_gap in knowledge* vs. *explain_standard of knowledge*:** Oben wurde bereits diskutiert, dass es Äußerungen gibt, die mit *explain_standard of knowledge* kodiert werden könnten, bei denen aber im Rahmen der BS/BKM *explain_gap in knowledge* zum Einsatz kommen muss. Auf diese Weise sollte dann verfahren werden, wenn das Zurückweisen einer Frage (siehe Beschreibung zum Typ *RK* auf S. 299) folgende Eigenschaften hat:

  - Es wird zumindest implizit erwähnt, dass beiden Partnern das entsprechende Wissen fehlt.
  - Es lässt sich erkennen, dass dem Sprecher bewusst ist, dass es sich um eine für das Vorankommen erhebliche Fragestellung handelt.

  Achtung: In weiterführenden Untersuchungen kann es evtl. sinnvoll sein, die zweite Eigenschaft zu modifizieren und zu fordern, dass vom Forschenden zu erkennen ist, dass es sich um eine für das Vorankommen erhebliche Fragestellung handelt. Eine solche Änderung kann z.B. dann interessant werden, wenn untersucht werden soll, in welcher Weise mit fehlendem Wissen umgegangen wird.

- ***agree_gap in knowledge* vs. *agree_standard of knowledge*:** Dem zweiten zentralen Prinzip im Umgang mit HHI-Konzepten folgend ist immer *agree_gap in knowledge* zu kodieren (siehe Abschnitt 7.2).

- ***explain_gap in knowledge* vs. *propose_step*:** Mit einer Äußerung wie

  „Jetzt müssen wir uns überlegen: (.) Wo kommt der her?“ (PR1.1)

  wird ein Vorschlag zum nächsten Arbeitsschritt gemacht. In diesem Arbeitsschritt soll herausgefunden werden, woher ein bestimmter Wert (der in einer `if`-Anweisung verwendet wird bzw. werden soll) kommt. Wenn man davon ausgeht, dass die Aussage mit „Wir sollten uns jetzt Zeit zum Überlegen nehmen. Schließlich wissen wir beide nicht, woher der Wert kommt“ gleichzusetzen ist, muss sie sowohl mit *propose_step* wie auch mit *explain_gap in knowledge* kodiert werden. Das Beispiel zeigt, wie Paraphrasierungen bei der Konzeptualisierung helfen können.

Weitere Anmerkungen zu *gap in knowledge*-Konzepten

Mittels der Klassen *gap in knowledge* können nicht alle erkennbaren Wissenslücken markiert werden, sondern nur solche, die vom Paar verbal expliziert werden. Untersuchungen, in denen auch andere Phänomene dieser Art von Bedeutung sind, müssen konzeptuelle Erweiterungen vornehmen, sei es durch neue, evtl. auch übergeordnete Konzepte oder durch die Erweiterung der Einsatzszenarien der *gap in knowledge*-Kodes.

7.6.5 *knowledge*-Konzepte

Fokus der *knowledge*-Konzepte

Wie bereits in Abschnitt 7.1.1 ausgeführt adressieren Konzepte vom Typ *knowledge* erst einmal Äußerungen, die nicht von anderen Elementen der BKM (außer solchen der Klasse *activity*) adressiert werden. Die Klasse *knowledge* fungiert also als eine Art „Sammelbecken“ für alle diejenigen im Kontext der zur erledigenden Entwicklungsaufgabe stehenden Äußerungsinhalte, die nicht in Äußerungen „verpackt sind“, die von anderen Klassen adressiert werden – zumindest insoweit, wie für eine Äußerung nicht ersichtlich ist, dass die Einführung einer ganz neuen (Objekt-)Klasse notwendig wird bzw. angemessen erscheint. Hierbei steht vor allem dasjenige (verbalisierte) Bestandswissen im Fokus, welches nicht von anderen Elementen der BKM (mit Ausnahme der *activity*-Konzepte) mit adressiert wird. Beispielsweise kann es sich hierbei um Wissen über die Funktionsweise bestimmter Entwurfsmuster [70] oder Wissen über im Vorfeld der Sitzung durchgeführte Tätigkeiten handeln, welches zum Zweck seiner Erläuterung geäußert wird. Dieser Teil des Bestandswissens wird im Rahmen der vorliegenden Arbeit mit W_k bezeichnet (siehe Abbildung 7.1). Da es sich bei W_k um Wissen im Sinne von $W_{\overline{BS}}$ handelt, hat es die auf S. 239 ff beschriebenen Eigenschaften. Beispielsweise umfasst es nur tatsächlich verbalisiertes Wissen. Genau genommen muss deshalb bei W_k auch von dem von einer Person bis zu einem bestimmten Zeitpunkt geäußerten Wissen gesprochen werden. Außerdem beinhaltet es nicht ausschließlich wahre, gerechtfertigte Meinungen. Es reicht vielmehr aus, dass eine geäußerte Meinung wahrhaftig ist.

Wie bereits erläutert (siehe z.B. Abbildung 7.1) kann es passieren, dass durch *knowledge*-Konzepte (speziell *explain_knowledge*) auch solche Äußerungen adressiert werden, die eigentlich primär darauf zielen, Erkenntnisse zu verbalisieren – unter anderem dann, wenn für diese Erkenntnisse (bisher) keine Identifikationsmerkmale festgelegt wurden. Bei der Verwendung von *explain_knowledge* sollte also im Hinterkopf behalten werden, dass es sich bei einer gerade zu klassifizierenden Äußerung evtl. um die Verbalisierung einer Erkenntnis handeln könnte, welche die Festlegung eines neuen Identifikationskriteriums für Erkenntnisse notwendig macht. Allerdings ist leicht ersichtlich, dass solche Kriterien in bestimmten Fällen nicht ausgemacht werden können, obwohl es sich um die Verbalisierung einer Erkenntnis handelt – zumindest dann nicht, wenn der Sprecher nicht direkt

diesbezüglich befragt werden kann. Anders ausgedrückt: Manchmal ist es unmöglich zu entscheiden, ob einer Äußerung eine Erkenntnis zugrunde liegt oder „nur“ bereits im Vorfeld erlangtes Wissen verbalisiert wird. In solchen Fällen muss *explain_knowledge* eingesetzt werden (siehe auch Abbildung 7.1).

Ein *explain_knowledge* kann auf Anforderung (*ask_knowledge*) oder aus eigenem Antrieb des Sprechers heraus geäußert werden. Beispiele sind in Tabelle 7.43 sowie in den Tabellen 7.3, 7.28 und 7.42 zu finden.

Achtung: Die Konstruktion der BS/BKM suggeriert in gewisser Weise, dass es in Paarprogrammierungssitzungen nur drei Formen von initiativen Äußerungen gibt – solche, in denen es darum geht, Vorschläge zu machen, solche, in denen es darum geht, Erkenntnisse mitzuteilen oder solche, die darauf abzielen, Bestandswissen zu verbalisieren/übermitteln. Dieser Eindruck kommt vor allem daher, dass die Klasse *knowledge* einerseits als „Sammelbecken“ und anderseits als die Klasse beschrieben wird, mit der vor allem Äußerungen adressiert werden, in denen es primär darum geht, Bestandswissen zu vermitteln. Es ist aber – offensichtlich – keineswegs so, dass in Paarprogrammierungssitzungen nur diese drei Formen von initiativen Äußerungen vorkommen. Vielmehr gilt, dass durch die BS/BKM bei bestimmten Äußerungen darauf verzichtet wird, die primäre lllokution zu markieren und stattdessen nur das mit überlieferte Bestandswissen explizit adressiert wird. Ein Beispiel folgt im ersten Punkt des nachfolgenden Abschnitts. Initiative Äußerungen, die auf Grund der hier festgelegten „Sammelbeckenfunktion“ in die Klasse *knowledge* fallen müssten, in denen aber keinerlei Bestandswissen mit übermittelt wird (und für die auch offensichtlich ist, dass keine andere Klasse der BKM „zuständig“ sein könnte), wurden bisher nicht beobachtet. Dies muss aber keineswegs bedeuten, dass es solche nicht gibt. Die hier gemachten Überlegungen gelten in analoger Weise auch für reaktive Äußerungen.

Identifizierte *knowledge*-Konzepte und ihre speziellen Eigenschaften

In Bezug auf die Verbalisierung des oben näher beschriebenen Teils des Bestandswissens einer Person wurden die HHI-Aktivitäten *explain_knowledge*, *agree_knowledge*, *disagree_knowledge*, *challenge_knowledge* und *ask_knowledge* beobachtet bzw. Phänomene als derartige subsumiert und beschrieben. Sie sind in Abbildung 7.3 grob dargestellt. Weitergehende Charakterisierungen werden im Folgenden beschrieben.

1. **Zwecke von Wissensäußerungen:** Durch die oben beschriebene „Sammelbeckenfunktion“ der Klasse *knowledge* lässt sich der Zweck, den Entwickler mit *knowledge*-Äußerungen verfolgen, weniger präzise charakterisieren als bei anderen Klassen wie z.B. *standard_of_knowledge*. So kann es im Rahmen einer BS-Kodierung bei bestimmten Äußerungen – zumindest dann, wenn nicht neue Konzepte oder Klassen eingeführt werden (sollen) – unumgänglich sein, auf die Kodierung des primären illokutionären Aktes zu verzichten und stattdessen *explain_knowledge* (oder auch *chal-*

#	HHI-Kode	Äußerung	Kontext/Kommentar
1	*P2. ask_knowledge*	„Aber cURL ist schon noch drauf? [...]“	Der Observer fragt, ob das Programm cURL[a] auf dem Entwicklungsrechner installiert ist.
2	*P1. explain_knowledge*	„[Ja]“	Der Driver beantwortet eine Frage von *P2* (siehe Äußerung 1).
3	*P1. explain_knowledge*	„Die rufen wir eben nicht auf.◡Weil die so einen riesen Overhead hat, hab’ ich die raus gelassen.◡Die ist nämlich hier oben auskommentiert.“	Der Driver beantwortet ein Frage des Partners in Hinsicht auf den Aufruf bestimmter existierender Funktionalitäten, indem er erklärt, was er im Vorfeld gemacht hat. Während er spricht, scrollt er zur angesprochenen Kodestelle. Interessanterweise stellt sich unmittelbar danach heraus, dass seine Aussage nicht vollständig richtig ist.
4	*P2. ask_knowledge*	„Was machst Du hier?◡Hier prüfst Du äh, ob äh `id_code` ’n echter (~Key), echter Dingens ist?“	*P2* stellt eine Frage zum bereits bestehenden Kode. Hierbei greift er zur Maus und scrollt zur infrage stehenden Stelle. Er wird also kurzzeitig zum Driver.
5	*P1. explain_knowledge*	„Ahm., genau. (.) Also einfach Hex mit 16 (.) Stellen.“	*P1* beantwortet eine Frage seines Partners (siehe Äußerung 4).
6	*P1. explain_knowledge*	„(#`friends_ids`#). (.) Da◡haben◡wir◡nämlich - deswegen hab’ ich das gemacht.◡Mich nervt das, Kommentare zu schreiben. `i d s` und ähm (..) und `ids` kannste ja voll leicht verwechseln.◡`id_codes` kannste nicht verwechseln. Und die Funktion hier <*markiert einen Funktionsaufruf*> heißt auch `id_to_code`. [...]“	Ohne dazu aufgefordert worden zu sein, erläutert der Driver *P1* von ihm im Vorfeld der Sitzung vergebene Namen für Variablen und Funktionen (Hinweis: *P1* spricht alle Bezeichner englisch aus.)
7	*P2. explain_knowledge*	„Du hast doch schon einmal ein Beispiel gemacht.“	*P2* weist darauf hin, dass sein Partner bez. einer Wissenslücke, die dieser kurz zuvor formuliert hatte, schon ein Beispiel in Form eines Kommentars notiert hatte. Während er spricht, greift er sich die Tastatur und scrollt zu der diesbezüglichen Stelle im Kode.[b]

[a] cURL ist eine Abkürzung für „Client for URLs“, ein Kommandozeilenprogramm zum Transfer von Dateien aus dem Internet (http://curl.haxx.se/ (Abruf: 08.08.2011)).

[b] Das Beispiel ist in Teilen schon vorher sichtbar. Es ist also nicht auszuschließen, dass *P2* dieses erst in diesem Moment entdeckt hat, es sich also um ein *explain_finding* und nicht um ein *explain_knowledge* handelt.

Tab. 7.43: Beispiele aus Session PR1.1 für die Verwendung der Konzepte *ask_knowledge* und *explain_knowledge*.

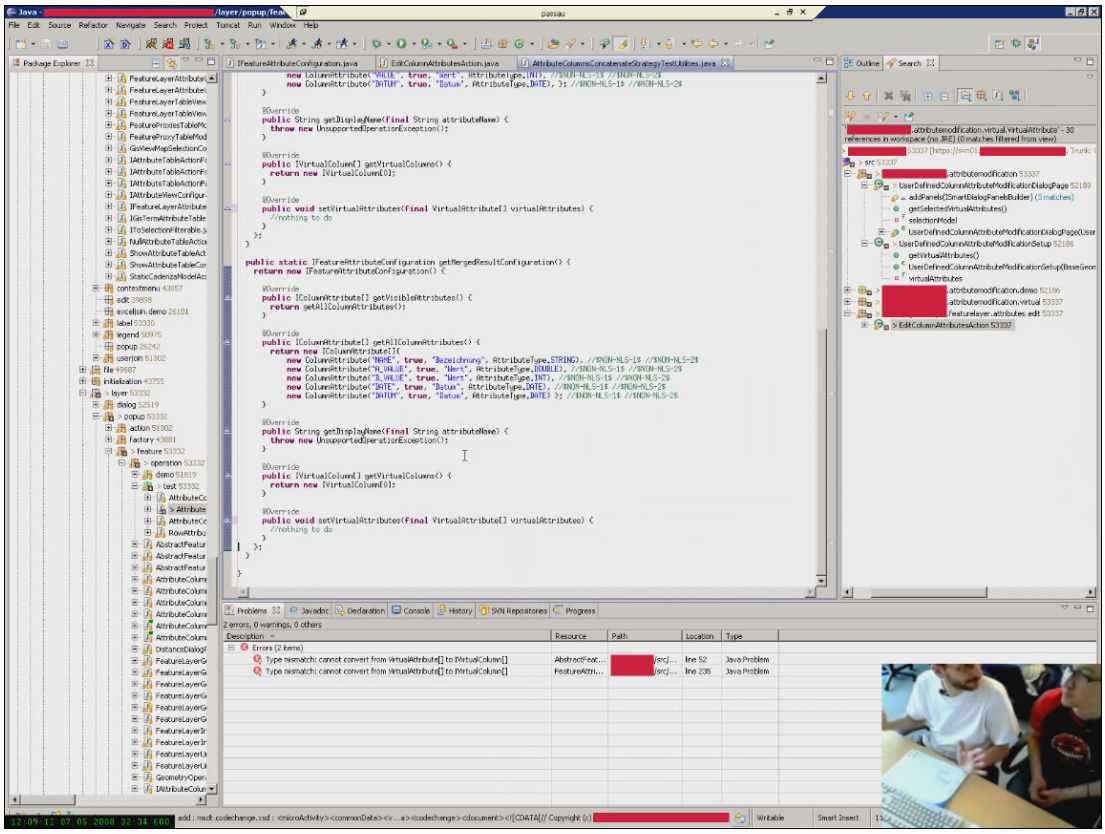


Abb. 7.15: Screenshot aus Sitzung PR2.1 (12:09:13). Die Sitzung läuft seit 32 Minuten und 34 Sekunden. Der Driver *P1* (links) gibt gerade zu verstehen, dass er den vom Partner zuvor kritisierten Kode nicht geschrieben hat.

lenge_knowledge) einzusetzen.[188] Beispielsweise antwortet der Driver *P1* in Sitzung PR2.1 auf eine kritische, sich auf vorhandenen Kode beziehende Frage des Partners mit:

> „Das hab' ich nicht geschrieben."

Hierbei reißt er die Unterarme hoch und nimmt eine abwehrende Haltung ein (siehe Screenshot in Abbildung 7.15). Bei der primären Illokution handelt es sich also offensichtlich nicht um ein „Erklären" oder „Erläutern", sondern eher um ein „Rechtfertigen". Eine solche wird aber im Rahmen der BKM nicht adressiert.[189] Die Äußerung muss somit als *explain_knowledge* kodiert werden.

[188] Inwieweit es ein Problem darstellen kann, dass die Konzepte der BKM nicht derart angelegt sind, dass sie ausschließlich den primären illokutionären Akt erfassen, muss in weiterführenden Studien analysiert werden.

[189] Dass eine solche Intention von der BS/BKM nicht direkt adressiert wird, hat sich natürlich nicht kanonisch ergeben, sondern ist eine der vielen Entscheidungen, die im Rahmen der Herleitung gefällt wurden.

2. **Begründung von Vorschlägen:** Wie bereits mehrfach erläutert, sind Begründungen von *P&P*-Vorschlägen in der Regel mit *explain_knowledge* (siehe z.B. Abbildung 7.5 und Tabelle 7.22), *explain_finding* (siehe z.B. Tabelle 7.3) bzw. evtl. auch *propose_hypothesis* zu annotieren. Bei Begründungen (zu *P&P*-Vorschlägen) muss der Forschende also herausfinden, ob sie auf Bestandswissen oder gerade gewonnenen Erkenntnisse fußen und ob sie vom Sprecher als uneingeschränkt richtig angenommen werden. Hierbei kann es nötig werden, Begründungen in Teiläußerungen zu zerlegen. Weitere Details zu Begründungen werden in Abschnitt 9.3.10 erläutert.

3. **Beurteilungen:** In den meisten Fällen handelt es sich bei der Bewertung oder Beurteilung von Artefakten, Teilen von Artefakten oder anderer Gegebenheiten wie z.B. Vorschriften um Erkenntnisäußerungen, die mit *finding*-Konzepten zu annotieren sind. Allerdings konnten auch Beurteilungen identifiziert werden, bei denen die zugrunde liegende Erkenntnis offensichtlich schon im Vorfeld erlangt wurde. In solchen Fällen sollten *knowledge*-Konzepte, z.B. *explain_knowledge*, eingesetzt werden. Ein diesbezügliches Beispiel wird in Zeile 4 in Tabelle 7.40 erläutert. Hier gibt der Sprecher zu verstehen, dass ihm bestimmte Eigenschaften einer Klasse schon eine geraume Zeit nicht gefallen. Auch bei der Äußerung in Zeile 4 von Tabelle 7.48 handelt es sich um eine derartige Beurteilung. Dieses Beispiel macht darüber hinaus deutlich, dass es im Rahmen der BS/BKM notwendig werden kann, Zubilligungen als *explain_knowledge* zu annotieren. Die Äußerung wird hierbei als „Hiermit teile ich Dir mit, dass ich damit einverstanden bin, dass diejenigen, die es für hilfreich erachten, *Alt-Shift-C* umzudefinieren, dies auch tun“ interpretiert – also als eine Äußerung, die für den Partner Wissen darstellt.

4. **Unaufgeforderter Wissenstransfer:** Es konnte beobachtet werden, dass Entwickler Wissen weitergeben, obwohl keine diesbezügliche Anfrage des Partners vorliegt (siehe Beispiel 6 in Tabelle 7.43). Auch solche Äußerungen werden im Rahmen der BS/BKM mit *explain_knowledge* kodiert. Auf eine (zusätzliche) Kodierung, z.B. mit *explain_finding*, wird hingegen verzichtet, auch wenn davon ausgegangen werden kann, dass der Äußerung die Einsicht zugrunde liegt, dass der Partner dieses Wissen (in der aktuellen Situation) benötigt. Von dieser Regel ausgenommen werden sollten explizite Verbalisierungen derartiger Erkenntnisse, wie z.B. „Du weißt ja wohl nichts über [...]“. Solche Äußerungen konnten aber im Rahmen der hier untersuchten Sitzungen nicht beobachtet werden.

5. **Rhetorische Fragen**: Fragen, die allem Anschein nach rhetorischer Natur sind, werden in der Regel nicht mit *ask_knowledge* kodiert. Stattdessen sollte versucht werden, deren primäre Illokution zu ermitteln. Dem Ergebnis dieser Betrachtungen entsprechend kann es dann beispielsweise notwendig werden, diese Äußerungen mit *explain_finding* (siehe Eintrag zu Erkenntni-

styp D_O in Tabelle 7.27) oder *propose_step* (siehe das zweite *propose_step* in Tabelle 7.28) zu kodieren.

6. **Zusammenfassung von Äußerungen**: Hier gilt das gleiche wie bei Erkenntnisäußerungen (siehe S. 267). Werden mehrere Informationen direkt hintereinander bzw. innerhalb einer Äußerung artikuliert, sollten sie auch zusammen mit nur einem *explain_knowledge* (bzw. *challenge_knowledge*) annotiert werden (siehe die Beispiele 3 und 6 in Tabelle 7.43). Von dieser Vorgabe sollte nur dann abgerückt werden, wenn zwei (oder mehrere) offensichtlich in keiner Weise direkt inhaltlich miteinander in Beziehung stehende Informationen zusammen formuliert werden. Sollte dies der Fall sein, muss die Äußerung geteilt und jeder Teil separat konzeptualisiert werden.

   Achtung: Analog zu Erkenntnisäußerungen (siehe S. 267) kann es in weiterführenden Untersuchungen wichtig werden, von solchen Zusammenfassungen abzusehen – z.B. dann, wenn Analysen anstehen, in denen weitere Arten von Wissen auseinandergehalten werden müssen.

7. **Wissensaussagen ergänzen**: Obwohl es umgangssprachlich nicht ungewöhnlich ist, davon zu sprechen, dass Wissen ergänzt wird, wurde im Rahmen der Herleitung der BS/BKM auf die Einführung des Konzeptes *amend_knowledge* verzichtet. Dies liegt daran, dass – vor allem aufgrund der „Sammelbecken"-Funktionalität von *knowledge* – bisher keine halbwegs allgemein anwendbaren Kriterien identifiziert werden konnten, mit deren Hilfe in nützlicher Weise leicht und nachvollziehbar zwischen Äußerungen unterschieden werden kann, in denen im Vorfeld formuliertes Wissen vom Typ W_k erweitert/ergänzt wird und solchen, in denen vom vorher geäußerten Wissen vom Typ W_k unabhängiges Wissen verbalisiert wird. Somit müssen alle Äußerungen, in denen W_k adressiert wird, sofern es sich nicht um Widersprüche (*disagree_knowledge* bzw. *challenge_knowledge*) oder Zustimmungen (*agree_knowledge*) handelt, mit *explain_knowledge* kodiert werden. Im Rahmen von weiterführenden Untersuchungen sollte aber überprüft werden, ob nicht doch Bedarf besteht, die BS/BKM um ein *amend_knowledge* zu erweitern, und in welcher Weise sich eine solche Erweiterung operationalisieren lassen kann.

8. **Uneigentliche Antworten**: Nicht jede Antwort auf eine Frage wird im Sinne des Fragenden gegeben. So kann es beispielsweise vorkommen, dass in der Antwort lediglich dargelegt wird, warum es gar nicht nötig ist, die Frage im Sinne des Fragenden zu beantworten (siehe z.B. Tabelle 7.44). Solche Phänomene werden von der BS/BKM nicht explizit adressiert, sondern schlicht mit *explain_knowledge* kodiert.

9. **Unterschiedliche Ursachen von Zustimmungen**: Betrachtet man Dialoge, in denen Bestandswissen vom Typ W_k übermittelt wird, so kann man

#	HHI-Kode	Äußerung	Kontext/Kommentar
1	*P1.ask_knowledge/ P1.propose_step*	„Weißt Du, wie die (.), wie ich an die Funktion ran komme, 'ne Methode zu verändern?“	Der Driver fragt nach einer bestimmten Funktionalität der IDE Eclipse. Dabei öffnet er das Kontextmenü auf der Methode `setVirtualAttributes`. Details, insbesondere zur Doppelkodierung, sind auf S. 204 erläutert.
2	*P2.explain_knowledge/ P2.disagree_step*	„Das bringt dir doch gar nichts.“	Der Observer *P2* antwortet seinem Partner. Allerdings nicht in der von diesem erhofften Art und Weise. Er gibt eine Einschätzung ab, die gleichzeitig als Ablehnung des Vorschlags fungiert.[a] Für Details siehe S. 205.
3	*P1.challenge_knowledge+ P1.propose_design*	„Doch ich kann in der Methode, wenn‿ich‿die‿aufgemacht‿hab', das `IVirtualColumn`[b] in 'n `I`(!!...!!)“	Der Driver widerspricht dem Observer unmittelbar, indem er betont, dass ihm der Zugriff auf die Methode natürlich etwas bringen würde. Er motiviert diese Aussage, indem er einen Designvorschlag unterbreitet. Bei dessen Formulierung wird er allerdings unterbrochen.
4	*P2.challenge_knowledge/ P2.disagree_step/ P2.disagree_design+ P2.explain_knowledge*	„Das‿bringt (!...!)‿Das‿wird‿dir‿nicht‿viel‿bringen.‿Aber *Alt-Shift-C*, mach, mach's mit *Alt-Shift-C*.“	Der Observer unterbricht seinen Partner und widerspricht noch einmal. Es kann allerdings nicht entschieden werden, ob er sich auf die ursprünglich vorgeschlagene Tätigkeit oder den Designvorschlag bezieht. Anschließend beantwortet er dann aber doch die ursprünglich von *P1* gestellte Frage.
5	*P1.challenge_knowledge*	„*Alt-Shift-C* dürften Konstanten sein.“	Der Driver widerspricht dem Observer unmittelbar und erklärt, dass mit der Tastenkombination *Alt-Shift-C* eine andere als die von ihm gewünschte Funktionalität aufgerufen wird.

[a] Bei der Einschätzung handelt es sich allem Anschein nach nicht um eine Erkenntnis im Sinne von *finding*. Vielmehr scheint es sich um eine auf einer grundlegenden Einstellung fußenden Meinung zu handeln. So antwortet P2 unmittelbar und in einem sehr gelassenen Tonfall.

[b] Er bezieht sich hier auf den Parameter `IVirtualColumn` der Methode `setVirtualAttributes`.

Tab. 7.44: Episode aus Sitzung PR2.1 (12:04:53–12:05:13), die folgende Phänomene enthält: (a) „Verpackter“ Handlungsvorschlag (Zeile 1): Details hierzu sind auf S. 204 erläutert; (b) Uneigentliche Antwort (Zeile 2): Details hierzu sind auf S. 320 erläutert; (c) Nicht zweifelsfrei zu ermittelnder Bezug einer Äußerung (Zeile 4): Es resultiert eine Mehrfachkodierung; (d) Widerspruch zu geäußertem Wissen – inkl. Begründung, warum dieses falsch sein muss (Zeile 5): Details hierzu sind auf S. 325 erläutert.

erkennen, dass es zumindest drei Typen von zustimmenden Äußerungen gibt:

(a) ***agree_knowledge***$_{well-known}$: Der Empfänger bestätigt die Korrektheit. Er kann dies tun, da er bereits im Vorfeld über dieses Wissen bzw. diese Informationen verfügt hat. Ein Beispiel hierfür ist in Tabelle 7.28 zu finden.

(b) ***agree_knowledge***$_{well-understood}$: Der Empfänger gibt zu verstehen, dass er die Erläuterungen bzw. die Informationen des Partners verstanden hat und als richtig beurteilt. Diesem Typ werden auch Zustimmungen zugerechnet, bei denen der Empfänger die Erklärung zwar nicht (vollständig) versteht oder ihm keine Mittel zur Verfügung stehen, diese (vollständig) zu verifizieren, er es aber für wahrscheinlich hält, dass die Erklärung richtig ist. Beispiele sind in Tabelle 7.42 (Rückmeldung auf die Antwort zu einer Frage; Indiz: nachfolgendes *explain_standard of knowledge*) und Tabelle 7.45 (Aha-Erlebnis, welches allerdings nur von sehr kurzer Dauer ist) erläutert.

(c) ***agree_knowledge***$_{accepted}$: Der Empfänger gibt zu verstehen, dass er die Korrektheit der Erläuterungen bzw. der Informationen des Partners akzeptiert, auch wenn er sich über diese kein Bild gemacht hat oder machen will. In Zeile 1 in Tabelle 7.48 ist ein diesbezügliches Beispiel zu finden. Außerdem macht auch die in Zeile 6 in Tabelle 7.45 dargestellte Äußerung deutlich, dass das Auftreten derartiger Zustimmungen nicht unplausibel ist – auch wenn es sich hier bei dem „O.K." weniger um eine Zustimmung als um eine prophylaktische Zurückweisung möglicher weiterer Erklärungen handelt.

Genau genommen ist es nur im Fall $agree_{well-known}$ vorbehaltlos angebracht, von Zustimmung zu übermitteltem Wissen zu sprechen. Im Fall von $agree_{well-understood}$ scheint es hingegen nicht unplausibel zu sein, stattdessen von einem *explain_standard of knowledge* zu sprechen (hierauf wird auf S. 341 noch einmal eingegangen), während ein $agree_{accepted}$ auf ein Zurückweisen weiterer Information oder Delegieren von Verantwortung hinweist. An dieser Stelle sei auf die Erläuterungen zu *agree_finding* auf S. 277 verwiesen. Hier wird auch ausgeführt, warum es im Rahmen einer BS-Kodierung weitgehend unerheblich ist, dass nicht immer eindeutig zwischen unterschiedlichen *agree*-Typen unterschieden werden kann und warum die hier vorgenommenen Erörterungen keinen Bruch mit dem „Black Box"-Prinzip (siehe S. 124) darstellen.

10. **Zustimmung vs. Aufmerksamkeit anzeigen**: Im Zusammenhang mit dem Konzept *agree_design* wurde bereits erörtert, dass die BS/BKM nicht zwischen Zustimmungen zu Inhalten von Äußerungen und so genannten „Back-channels" unterscheidet (siehe S. 175). Hier soll gelten: Äußerungen

#	HHI-Kode	Äußerung	Kontext/Kommentar
1	*P1.propose_design*	„So wie das momentan (,) is', können wir den hier vernachlässigen.“	Der Driver schlägt vor, eine bestimmte Kodezeile[a] zu löschen. Er markiert diese, während er spricht.
2	*P2.explain_knowledge/* + *P2.agree_design*	„Das war eh nicht richtig.“	Der Observer wirft ein, dass diese Zeile sowieso nie richtig war.
3	*P1.challenge_knowledge*	„Das war schon richtig. Das war nach deiner Änderung nicht mehr richtig. Davor war es richtig.“	Der Driver widerspricht der Aussage des Partners und erklärt, bis zu welchem Zeitpunkt die Zeile richtig war. Die Änderung, die er dabei anspricht, ist vom Partner im Vorfeld der Sitzung durchgeführt worden.
4	*P2.agree_knowledge+ P2.disagree_knowledge*	„Aha,‿ok. (..) Eigentlich nicht.“	Zuerst stimmt der Observer zu, dann widerspricht er.
5	*P1.explain_knowledge*	„Das hat da (!...!) Also die, die‿die Strategie vorhin war die, dass man über (.) äh `AllColumnAttributes` auch die virtuellen mitkriegt. Deshalb musst' ich die zurücksetzen und dass wenn sich nichts ändert, die wieder setzen.“	*P1* gibt unmittelbar eine weitere Erklärung ab.
6	*P1.explain_standard of knowledge*	„(O.K., versteh' ich jetzt gar nicht, aber gut.) Ich hab's nicht verstanden, aber gut.“	Siehe Zeile 6 in Tabelle 7.39.

[a] Es handelt sich um das Setzen eines Attributes: `attributeConfiguration.setVirtualAttributes(virtualColumns)`

Tab. 7.45: Episode aus Sitzung PR2.1 (12:06:38–12:07:14), in der geäußertes Wissen vom Partner richtiggestellt wird.

wie z.B. ein während einer Erklärung eingestreutes „[Â]hm“ des Partners werden immer dann mit *agree*-Konzepten, also z.B. *agree_knowledge*, annotiert, wenn sie potentiell affirmativ sind. Weiterführende Untersuchungen müssen entscheiden, ob und in welcher Weise sie von dieser Regelung abweichen wollen.

Es bleibt zu bemerken, dass die Zustimmungen zu *explain_knowledge*-Äußerungen in der Regel sehr kurz sind – z.B. „Ja“ (siehe Beispielepisode in Tabelle 7.46), „Aha,‿O.K.“ (siehe Beispielepisode in Tabelle 7.45) oder „Aja, O.K.“ (PR2.1).

#	HHI-Kode	Äußerung	Kontext/Kommentar
1	*P1.explain_knowledge*	„Ich hab' gesagt, ich hab' [...] Ich (.) ähhhm arbeite ein bisschen anders als du. Ich [...] –◡mit verwenden. Ich hab' schon da in der Berücksichtigung mit drin, dass die evtl. aus diesem `getColumnAttributes`, dass die da auch drin sind. Deshalb werden die da jetzt zurückgesetzt."	Auf Nachfrage erklärt der Observer seinen im Vorfeld erstellten Kode. Er zeigt dabei auf bestimmte auf dem Monitor sichtbare Stellen.
2	*P2.agree_knowledge+ P2.explain_standard of knowledge*	„[Â]hm. Dann kann man's leicht, dann kann man's leicht ändern.◡[!!Das hat mich etwas verwirrt.!!]"	*P2* stimmt zu und erklärt darüber hinaus, in welcher Weise er die Ausführungen des Partners verstanden hat[a] und dass ihn die entsprechenden Kodezeilen ursprünglich „verwirrt" hatten. Bei der letzten Teiläußerung fällt ihm *P1* ins Wort (siehe nächste Zeile).
3	*P1.explain_knowledge*	„[!!Ja!!]"	*P1* gibt zu verstehen, dass *P2* ihn richtig verstanden hat. Die BS/BKM stellt für eine solche Form von Zustimmung kein spezialisiertes Konzept zur Verfügung.

[a] Die Äußerung wurde folgendermaßen interpretiert: „Du willst sagen, dass Du das deshalb so gemacht hast, weil man es auf diese Art leicht ändern kann".

Tab. 7.46: Episode aus Sitzung PR2.1 (11:53:01–11:53:23). *P2* stimmt einer Erklärung des Partners zu. Er ergänzt seine Zustimmung durch ein *explain_standard of knowledge*. In dieser Ergänzung konzentriert er sich auf den Kern der Äußerung von *P1*. Die Teiläußerung befindet sich auf der Grenze zwischen *explain_standard of knowledge* und *explain_knowledge*.

11. **Widerspruch**: Übermitteltem Wissen kann auf zweierlei Weise widersprochen werden:

    (a) ***disagree_knowledge***: Der Widerspruch erfolgt, ohne dass dabei mit Wissen argumentiert wird, das über die pure Ablehnung hinausgeht. In Zeile 4 in Tabelle 7.45 ist ein Beispiel für einen solchen Widerspruch erläutert. Solche Widersprüche können sehr kurz sein (siehe die Erläuterungen im Zusammenhang mit der Klasse *design* auf S. 176).

    (b) ***challenge_knowledge***: Im Rahmen der Äußerung wird über den puren Widerspruch hinausgehendes, entgegengesetztes oder zumindest nicht mit dem ursprünglich geäußerten vollständig verträgliches Wissen verbalisiert (Details werden im nachfolgenden Punkt erläutert).

12. **Kontroverses Wissen**: In ähnlicher Weise wie beobachtet werden konnte, dass Entwickler den Erkenntnissen ihres Partners widersprechen, indem sie diesen eigene Einsichten entgegensetzen (siehe S. 275), konnten auch Äußerungen identifiziert werden, in denen sich Entwickler mit Wissensäußerungen vom Typ *knowledge* nicht einverstanden erklären, indem sie eigenes, zumindest in Teilen nicht mit dem des Partners übereinstimmendes Wissen artikulieren. Hierbei kommt es vor, dass das geäußerte Wissen nicht direkt korrigiert wird, sondern stattdessen solches Wissen geäußert wird, welches zeigt, dass das ursprünglich geäußerte falsch sein muss (siehe Beispiel in Tabelle 7.44). Im Rahmen der Herleitung der BS/BKM wurde entschieden, in solchen Fällen nicht *disagree_knowledge* und ein weiteres *explain_knowledge*, sondern *challenge_knowledge* zu annotieren. Diese Festlegung fußt auf dem auf S. 161 erläuterten zentralen Prinzip im Umgang mit HHI-Konzepten. In weiterführenden Untersuchungen sollte aber darüber nachgedacht werden, ob es in Hinsicht auf das Untersuchungsziel hilfreich sein könnte, doch *disagree_knowledge* und *explain_knowledge* zu annotieren. Gleiches gilt für *challenge_finding*. Eine kurze Episode, in der geäußertes Wissen vom Typ W_k direkt korrigiert wird, ist in Tabelle 7.45 zu finden.

13. **Widerspruch = Zuspruch zum Gegenteil**: Handelt es sich bei einer Wissensäußerung (z.B. einem *explain_knowledge*) um die Entscheidung einer zweiwertigen Fragestellung (z.B. „X hat nicht die Eigenschaft Y.“), kann ein einfacher Widerspruch zu dieser (z.B. ein „Doch.“) in der Regel mit dem Äußern eines Gegenvorschlags gleichgesetzt werden. Eine Kodierung mit *challenge_knowledge* ist in solchen Fällen angebracht.[190]

14. **Überzeugungen/Standpunkte**: Da W_k als Untermengen von W^+_{BS} bzw. W^-_{BS} unter anderem (als wahr angenommene) Meinungen enthält (siehe S. 240), sind sowohl Äußerungen, in denen subjektive Ansichten wie z.B. Überzeugungen oder Standpunkte geäußert werden wie auch Erwiderungen auf diese, in denen entgegengesetzte Überzeugungen bzw. Standpunkte übermittelt werden, mit *explain_knowledge* bzw. *challenge_knowledge* zu annotieren – zumindest insoweit, wie sie nicht in andere Klassen wie z.B. *finding* fallen. Ein Beispiel ist in Tabelle 7.47 zu finden.

    Achtung: Aus Perspektive der Herleitung der BS ist die hier vorgenommene Form der Darstellung nicht korrekt. Schließlich sind die Wissensbegriffe der BS zu nicht unerheblichen Teilen in einem evolutionären Prozess aus Anpassungen von *explain_knowledge* sowie anderer Konzepte hervorgegangen, mit denen das Ziel verfolgt wurde, die Elemente der BKM einsatzfähig zu machen. Somit stimmt es genau genommen nicht, dass sich die Einsatzgebiete von *explain_knowledge* oder *challenge_knowledge*

[190] Achtung: Das dem Beispiel zugrunde liegende Phänomen stammt aus Sitzung PR2.2. Die Mengen der Teilnehmer in Sitzung PR2.1 und PR2.2 sind disjunkt.

aus der *Definition* von W_k bzw. W_{BS}^{-} ergeben. In Hinsicht auf die Anwendung der BS/BKM ist es aber hilfreich, derart zu argumentieren.

15. **Eingeschränkte Zustimmung bzw. Ablehnung/Unterscheidung Zustimmung und Ablehnung**: Es kommt vor, dass in zustimmenden Äußerungen verbleibende Zweifel mit artikuliert werden – auch ohne dass diese ausformuliert werden (siehe z.B. letzte Äußerung in Tabelle 7.22). Umgekehrt werden auch Ablehnungen nicht immer als absolute Aussagen formuliert. So kommt es vor, dass sie in Frageform geäußert werden und ein „in Zweifel ziehen“ darstellen (in Sitzung PR1.1 z.B. in Form eines „Echt?“). Zustimmungen und Ablehnungen sind also nicht immer klar voneinander zu trennen. In weiterführenden Untersuchungen muss nötigenfalls näher auf diese Problematik eingegangen werden.

16. **Nach Wissen fragen**: Ein Wissenstransfers wird in vielen Fällen dadurch eingeleitet, dass Fragen an den Partner gerichtet werden (siehe z.B. Tabelle 7.28). Solche Fragen sind nur dann mit *ask_knowledge* zu kodieren, wenn sie nicht in *P&P*-Klassen fallen.

17. **Fragen, die Antwortmöglichkeiten enthalten**: Es konnte beobachtet werden, dass Entwickler beim Stellen einer Frage Hypothesen über die (richtige) Antwort aufstellen (siehe Beispiel 4 in Tabelle 7.43). Auch in diesen Fällen wird in der Regel nur *ask_knowledge* kodiert. Auf eine zusätzliche Annotation, z.B. mit einem *propose_hypothesis*, wird verzichtet. Nur wenn nachvollziehbar dargelegt werden kann, dass es sich um eine Hypothese und nicht um eine Frage handelt, wird entsprechend kodiert (siehe auch S. 338).

18. **Aussagen vs. Fragen**: Es ist – auch wenn man Kontext und Intonation einbezieht – nicht immer ersichtlich, ob ein Sprecher mit einer Äußerung Bestandswissen vermitteln will oder die Korrektheit von Informationen verifiziert haben möchte – also eine Frage stellt. Evtl. kann noch nicht einmal ausgeschlossen werden, dass keines von beiden der Fall ist und der Sprecher stattdessen versucht, eine Erkenntnis zu kommunizieren. So äußert der Observer *P2* in Sitzung PR2.1, nachdem er nach einem erfolgreichen Test feststellt, dass man nun „weiter machen“ könne, ungefragt „Die Verarbeitung der, der (~Thread-Ids) bleibt ja gleich, da ist ja nichts (!...!)“. Sein Partner antwortet hierauf mit „Genau“. Bei diesem Phänomen macht besonders die Dechiffrierung der Bedeutung des Modalpartikels „ja“ im Zusammenhang mit dem Satzabbruch Schwierigkeiten.[191] Mit diesem könnte der Sprecher darauf hinweisen wollen, dass die geäußerte Tatsache bekannt ist. Es könnte sich aber auch um eine Abkürzung für ein „ja wohl“ handeln, welches

[191] Eine Diskussion darüber, ob Modalpartikel wie „ja“ als illokutive Indikatoren dienen, es also ermöglichen (sollen), Äußerungen richtig zu interpretieren, bzw. inwieweit sie illokutionsmodifizierend bzw. illokutionspräzisierend sind, sprengt den Rahmen der vorliegenden Arbeit. Siehe hierzu z.B. Helbig [94].

#	HHI-Kode	Äußerung	Kontext/Kommentar
1	*P2.explain_finding*	„Ja richtig, das hat' ich (!!gerad' drin!!), dass (~wir's nirgends) anders gebraucht haben. (!!Brauchen wir!!) aber natürlich. In jedem, in jedem Fall brauchen wir das (.) äh auch (..) woanders noch."	Nach der Durchsicht von `VirtualAttribute.java` gibt *P2* zum Ausdruck, dass ihm nun klar geworden ist, dass der in der Datei definierte Aufzählungstyp `VirtualAttribute`[192] auch an anderen Stellen gebraucht wird. Während er spricht, macht sein Partner zwei potentiell affirmative Einwürfe („[A]hm"), die mit *agree_finding* zu annotieren sind.
2	*P2.explain_knowledge*	„⌣Also⌣(!!für⌣mich!!)⌣fänd' (~ich's) jetzt zumindest nicht schlimm.⌣Weil das ja kein, (.) keine `pro`-Funktionalität[193] ist, das `VirtualAttribute` selbst. Für mich nicht. (!!Sondern!!) nur die GUI dafür[194] ist (!!(~`pro`-Funktionalität)!!)."	Ohne eine Sprechpause zu machen, erläutert *P2* seinen Standpunkt bezüglich des Aufzählungstyps `VirtualAttribute`. Achtung: An dieser Stelle kann nicht wirklich ausgeschlossen werden, dass es sich um eine neue Erkenntnis handelt (siehe hierzu z.B. Abbildung 7.1 bzw. die Erörterungen auf S. 331). *P1* streut potentiell affirmative Äußerungen ein. Am Ende fällt er *P2* ins Wort.
			Fortsetzung auf der nächsten Seite.

[192] `public enum VirtualAttribute` – für Details zu Aufzählungstypen in Java siehe http://download.oracle.com/javase/tutorial/java/javaOO/enum.html (Abruf: 17.05.2011). `VirtualAttribute` gehört derzeit zum Paket `de.<**Teil eines Paketnamens 0**>.<**Teil eines Paketnamens 2**>.pro.attributemodification.virtual`. Das Paket, in das der Aufzählungstyp verschoben werden soll, ist `de.<**Teil eines Paketnamens 0**>.<**Teil eines Paketnamens 1**>.<**Teil eines Paketnamens 2**>.map.layer.feature.attribute.virtual`. Um dieses (in Teilen noch anzulegende) Paket scheint es in der weiteren Diskussion zu gehen. Die Entwickler nennen diesen Pfad aber zu keinem Zeitpunkt. Für den Forschenden ist er nur deshalb ermittelbar, weil das Verschieben im weiteren Verlauf tatsächlich durchgeführt wird.

[193] *P2* verwendet den Begriff `pro`-Funktionalität, ohne näher zu erläutern, was er damit meint. Im Rahmen der hier beschriebenen Untersuchungen wurde davon ausgegangen, dass er mit diesem Begriff solche Funktionalitäten adressiert, die in das Paket `pro` gehören. Als Schreibweise bei der Transkription wurde dementsprechend `pro`-Funktionalität gewählt.

[194] Gemeint ist wohl die grafische Benutzeroberfläche für das Darstellen und Verändern von `VirtualAttributes`.

			Fortsetzung der Tabelle der vorherigen Seite.
3	*P1.explain_standard of knowledge*	„Da war ich (.) mir noch nicht so sicher.‿Weil ich bin mir noch nicht klar darüber ob das eher 'n Typ ist (!!(.)!!) oder ob das (!!(.)!!) wirklich noch mächtiger (!...!)“	*P1* fällt dem Partner ins Wort und erläutert, wie weit er sich über die Bedeutung von `VirtualAttribute` im klaren war bzw. ist. *P2* nutzt eine erste kurze Sprechpause für ein „Hmmmm“. Dies scheint reinen „Back-channel“-Charakter zu besitzen und wurde nicht weiter kodiert. In einer zweiten sehr kurzen Pause setzt er an etwas zu sagen, bricht diesen Versuch aber sofort wieder ab („Also (!...!)!“).
4	*P1.explain_knowledge*	„‿Also, (!!vom Namen!!) her wär's mächtiger‿–‿wie es momentan heißt, als nur 'n Typ.‿Aber von den Sachen die wir (!!hier drin‿drin!!) haben momentan, ist es momentan eher so ein Initialisierungstyp, der (!...!)“	*P1* fährt unmittelbar fort und erläutert seine Einschätzungen zu `VirtualAttribute`. *P2* setzt weitere zweimal an etwas zu sagen („Also mein (...)“ und „Das ist, das (...)“).
5	*P2.challenge_knowledge*	„Für mich ist's kein,‿nicht unbedingt (!!'n Typ!!) sondern Attribut.“	P2 setzt seine Einschätzung – einen Standpunkt – entgegen. Sein Partner wirft ein „Hmm“ ein. Der Intonation folgend scheint es sich um eine Zustimmung zu handeln.
6	*P2.explain_standard of knowledge*	„‿Aber das wird sich noch raus stellen.‿Weiß ich nicht.‿ Weiß ich noch nicht.“	*P2* erläutert, dass seine Einschätzung vorläufig ist bzw. dass es evtl. notwendig werden könnte, diese zu revidieren.[195]
			Fortsetzung auf der nächsten Seite.

[195] Unter Berücksichtigung dieser Äußerung liegt der Gedanke nahe, diese und die zuvorgemachte zusammen als ein *explain_hypothesis* zu konzeptualisieren. In Abschnitt 9.3.4 wird auf derartige Problematiken näher eingegangen.

			Fortsetzung der Tabelle der vorherigen Seite.
7	*P2.mumble_sth*	„Hmm, (.) für mich is' (!...!) (....) Ja, müssen wir schauen. Also ich würd' (!...!) (.....)“	*P2* fährt unmittelbar fort. Seine Äußerungen sind aber fragmentarisch. Dabei sieht er auf den Bildschirm und beißt sich gelegentlich auf die Lippen. Allem Anschein nach denkt er nach. Währenddessen sieht sein Partner auch auf den Bildschirm.[196]
8	*P2.propose_design*	„War's das denn einfacher, 'n neue Klasse in `<**Teil eines Paketnamens 1**>` einzuführen (.) und dann zwei Klassen zu haben, als 'ne vorhandene Klasse von `<**Teil eines Paketnamens 1**>.pro` nach `<**Teil eines Paketnamens 1**>` zu verschieben, die keine erkennbare (!!`pro`-Funktionalität!!) hat?“	Nach der letzten Sprechpause fährt *P2* fort. Auf den ersten Blick scheint es sich um ein *ask_knowledge* zu handeln. Unter Berücksichtigung des Kontextes ist aber zu erkennen, dass *P2* einen Gestaltungsvorschlag macht (Verschieben als Alternative zu zwei Klassen). Es handelt sich also um ein *propose_design* vom Typ 3 (Orientierung suchen – siehe Tabelle 7.5), wobei schon erkennbar ist, dass P2 ein Verschieben vorziehen würde. Gegen Ende wirft sein Partner ein „Hmmmm“ ein, was der Intonation nach Nachdenken anzeigt.
			Fortsetzung auf der nächsten Seite.

[196] Interessanterweise wird auf dem Bildschirm schon seit dem letzten Satz in Äußerung 1 nicht mehr `VirtualAttribute`, sondern die Klasse `EditColumnAttributesAction` angezeigt. *P2* hatte die Ansicht gewechselt. Im weiteren Diskussionsverlauf scheint diese Klasse aber keine Rolle zu spielen.

			Fortsetzung der Tabelle der vorherigen Seite.
9	*P1.decide_design* + *P1.explain_knowledge*	„Also ich hab'‿ich sperr' mich momentan nicht (!!dagegen!!), das zu verschieben.‿Das‿is,‿ich‿(!...!)‿Ich hab' nur gesagt, warum ich das noch nicht gemacht hab'."	*P1* erklärt sich nach einer Pause von ca. zwei Sekunden prinzipiell mit dem Verschieben einverstanden. Außerdem erklärt er, welche Intention er mit seinen letzten Äußerungen verfolgt hat. Es handelt sich allem Anschein nach um den illokutionären Akt des Rechtfertigens.[197] *P2* wirft ein „O.K.", also ein *agree_design*, ein.
10	*P2.propose_design*	„Wollen wir's einfach probieren?"	Obwohl *P1* schon prinzipiell zugestimmt hat, fragt *P2* noch einmal nach bzw. schlägt noch einmal vor, das Verschieben durchzuführen. Derartige Phänomene werden von der BKM nicht explizit adressiert.
11	*P2.agree_design*	„Ja"	

Tab. 7.47: Episode aus Sitzung PR2.1 (11:56:53–11:58:22), in der die Entwickler – zumindest in der Wahrnehmung von *P2* – unterschiedliche Standpunkte (im Sinne von subjektiven Meinungen) in Hinsicht auf die Bedeutung/Funktion eines bestimmten Aufzählungstyps (`VirtualAttribute`) und seiner damit verbundenen Position in der Paketstruktur des Projektes vertreten (siehe Zeile 2, 4 und 5). Die Episode startet, nachdem *P2* ein *explain_standard of knowledge* in Hinsicht auf die Ist-Situation, nämlich die momentane Position des Typs in der Paketstruktur (siehe Tabelle 7.39 Punkt 5) geäußert, nachfolgend die Datei `VirtualAttribute.java` geöffnet und ca. fünf Sekunden angesehen hat. Darüber hinaus verdeutlicht die Episode: (a) Partikel wie z.B. „Hmm" treten mit unterschiedlichen Bedeutungen auf. Diese Bedeutungen können nur in beschränktem Maß mittels der Intonation sowie des Kontextes ermittelt werden (vergleiche Zeile 3, 5 und 8). (b) Bei der Analyse sollte immer überprüft werden, ob es sich bei einem vermeintlichen *explain_knowledge* oder *ask_knowledge* nicht doch um eine andere Art von Phänomen, wie z.B. ein *propose_design*, handelt (siehe Zeile 8).

[197] An dieser Stelle zeigt sich eine weitere Auswirkung der Konzeption der BS/BKM dahingehend, dass viele primäre Illokutionen (hier ein Rechtfertigen) nicht explizit adressiert werden können.

die Äußerung als Frage markieren soll. Außerdem ist es möglich, dass der Modalpartikel Erstaunen signalisieren soll. Letzteres würde bedeuten, dass es sich gar nicht um eine Äußerung vom Typ *knowledge* handelt, sondern um eine vom Typ *finding*. Intonation und Kontext konnten in diesem Fall keine Entscheidungshilfen geben. In solchen Fällen sollte im Rahmen der BS/BKM analog zu den an anderen Stellen formulierten Regeln (erst einmal) *explain_knowledge* annotiert werden.

Abgrenzungen von *knowledge*-Konzepten zu anderen Konzepten

Im Weiteren werden die wichtigsten Abgrenzungen von *knowledge*-Konzepten zu Konzepten anderer Klassen erörtert, wobei auf die Unterschiede zwischen *explain_knowledge* und *remember_source of information* erst im Abschnitt 7.6.6 eingegangen wird.

- ***explain_knowledge* vs. *propose_step*:** Wie bereits erläutert können Vorschläge zum nächsten Arbeitsschritt auf Bestandswissen basieren bzw. Bestandswissen vermitteln. Trotzdem sind sie mit *propose_step* (oder *challenge_step* etc.) zu annotieren (siehe z.B. S. 154). Allerdings sollte im Rahmen einer BS-Kodierung, wie bereits auf S. 192 angesprochen, zwischen dem Vorschlagen eines Arbeitsschritts und Erklärungen, wie dieser durchzuführen ist, unterschieden werden. Die Erklärungen müssen, wenn sie keine Erkenntnisse im Sinne der Klassen *finding* oder *hypothesis* darstellen, einzeln mit *explain_knowledge* (oder *challenge_knowledge*, was in diesem Zusammenhang allerdings bisher nicht beobachtet werden konnte) konzeptualisiert werden. Zur Verdeutlichung wird in Tabelle 7.48 ein weiteres Beispiel diskutiert.

- ***explain_knowledge* vs. *propose_design*:** Wie schon auf S. 174 erläutert sollte auch bei Designvorschlägen zwischen Vorschlag und Vorschlagsbegründung differenziert und entsprechend kodiert werden.

  Achtung: Vorschläge zum Design können (wie bereits auf S. 180 angesprochen) in Wissensäußerungen (vom Typ *knowledge* oder *finding*) „verpackt“ formuliert sein (ein Beispiel – allerdings für *finding* – wird im Rahmen der Erläuterungen zu Klasse *activity* in Tabelle 7.55 gegeben). In derartigen Fällen sollte eine Doppelkodierung vorgenommen werden.

- ***explain_knowledge* vs. *explain_finding*:** Grundsätzliches zur Differenzierung zwischen *explain_knowledge* und *explain_finding* wurde bereits auf S. 151 und S. 244 f erläutert. Auf S. 250 ff wurde darüber hinaus festgelegt, dass (zumindest solange keine Hinweise vorliegen, die eine andere Vorgehensweise nahe legen) nur solche Äußerungen als Verbalisierungen von Erkenntnissen gewertet werden sollen, die bestimmten Kriterien genügen. Hierbei muss im Auge behalten werden, dass es sich bei diesen Kriterien

#	HHI-Kode	Äußerung	Kontext/Kommentar
1	*P2.agree_knowledge*	„Okayyy“	Der Observer stimmt einem *challenge_knowledge* des Partners zu (siehe Zeile 5 in Tabelle 7.44). Da die Korrektheit der Aussage von ihm offensichtlich nicht verifiziert wird, handelt es sich um ein $agree\_knowledge_{accepted}$.
2	*P2.explain_finding*	„Dann hast Du das umgeändert.“	Unmittelbar danach verbalisiert *P2* eine Erkenntnis vom Typ T_C, die sich für ihn aus dem Einwand des Partners ergeben hat. Die Äußerung lässt sich zum besseren Verständnis folgendermaßen paraphrasieren: „Dann hast Du die Standardbelegung der Tastenkombination *Alt-Shift-C* geändert.“
3	*P1.amend_finding*	„Dis is’ (.) ähm <**Name eines anderen Entwicklers**>.◡Und es gibt die (!!(.)!!) Absprache hier, (!!(~übrings), dass wir das so!!) geändert haben.“	Der Driver *P1* ergänzt die Erkenntnis des Partners um eine Begründung für die Änderung der Standardbelegung. Unter Berücksichtigung der zentralen Prinzipien (siehe Abschnitt 7.2) wird die Äußerung, obwohl *P1* in ihr nur Bestandswissen verbalisiert, nicht als *explain_knowledge* kodiert. Details hierzu werden auf S. 335 erläutert. Parallel zu Erklärung von *P1* äußert sich sein Partner zustimmend („[A]h[a].“ bzw. „Dann noch (!...!)◡Ja, klar.“).
4	*P2.agree_finding* + *P2.explain_knowledge*	„Ja◡klar,◡also◡wer,◡wer◡das◡mag,◡wer◡das◡ändern◡mag,◡klar.◡Ich (~~).“	*P2* stimmt der Ergänzung zu. In direktem Anschluss beurteilt er die vom Partner erläuterte Absprache. Die Beurteilung stellt eine Meinungsäußerung dar, die im Rahmen der BS als Verbalisierung von Bestandswissen klassifiziert wird (siehe S. 319).
5	*P2.explain_knowledge*	„Dann mach (!...!). Du musst aber hier drauf gehen <*Zeigt auf den Methodennamen innerhalb einer Signatur*> (..) Dann *Refactor*, *Change Method Signature*.“	*P2* geht wieder auf die ursprünglich von *P1* gestellte Frage ein (siehe Zeile 1 in Tabelle 7.44). Er erläutert, dass *P1* das Kontextmenü auf der Methode `setVirtualAttributes` öffnen und dort unter dem Punkt *Refactor* die Funktionalität *Change Method Signature* aufrufen muss.

Tab. 7.48: Episode aus Sitzung PR2.1 (12:05:16–12:05:35), die unmittelbar an die in Tabelle 7.44 dargestellte anschließt. Es sind u.a. folgende Phänomentypen illustriert: (a) Zustimmungen zu Wissensäußerungen, die nicht auf eigenem Wissen oder einer Überprüfung beruhen (Zeile 1), (b) Ergänzungen zu Erkenntnisäußerungen, die selbst nicht auf einer Erkenntnis basieren (Zeile 3), (c) Zubilligungen, die als *explain_knowledge* zu kodieren sind (Zeile 4) sowie (d) Erläuterungen in Hinsicht auf die Durchführung eines Arbeitsschrittes (Zeile 5).

um keine hundertprozentig zuverlässigen Indikatoren handelt bzw. handeln kann, da sie nicht dahingehend konzipiert wurden, nur Äußerungen festzuhalten, in denen die Sprecher explizit darauf hinweisen, dass es sich bei ihren Ausführungen um eine gerade gewonnene Erkenntnis handelt. Insbesondere ist es oft schwierig nachzuweisen bzw. plausibel zu machen, dass es sich bei einer Äußerung um eine Verbalisierung einer Erkenntnis vom Typ T_U oder T_C handelt. Aber auch bei Erläuterungen, die sich auf bereits bestehenden Kode beziehen, also bei solchen, die auf den ersten Blick darauf hindeuten, dass es sich um eine Erkenntnis vom Typ D handelt, kann nicht immer ausgeschlossen werden, dass es sich nicht doch um eine bereits im Vorfeld gewonnene Einsicht oder sogar um schon länger vorhandenes Wissen handelt. Hierzu betrachte man die beiden folgenden Beispiele:

1. Im Verlauf von Sitzung PR1.1 versucht das Paar, bestehenden Kode, genau genommen eine Funktion, in der Objekte aus einem Cache gelöscht werden, besser zu verstehen. Hierbei fragt *P1*:

   > „Was passiert'n eigentlich jetzt, wenn sich dieser Freund hier löscht <**P1* zeigt auf eine Notiz, die er sich auf einem Zettel gemacht hat.*>? (.....) Dann‿muss‿ich‿den‿doch‿streng‿genommen, dann‿muss‿ich‿doch‿streng‿genommen (.) auch mein, meine Freundesliste (~invalidieren).“[198]

   Sein Partner *P2* antwortet ihm unmittelbar darauf folgend:

   > „Das‿macht‿er‿ja.‿Er, (.) du bist ja (~auch) Freund von ihm, der sich da löscht. (.) Und hier in dieser Funktion wird ja auch (.) für alle (.) von seine Freunden, also auch für dich, (.) `MemCache`-Objekt (~rausgeschmissen).“

   Im Rahmen der Videoanalysen ließ sich nicht zweifelsfrei ermitteln, ob *P2* länger vorhandenes Wissen oder eine gerade, z.B. in den ca. 5 Sekunden, die *P1* beim Stellen seiner Frage verharrt, erlangte Erkenntnis verbalisiert.

2. Ähnliches gilt für den folgenden Dialog aus der gleichen Sitzung. Hier fragt *P2*:

   > „Speichern wir beide Werte oder speichern wir (!...!). (..) Oder reicht uns `last_change`? (.) Brauchen wir eigentlich `last_request`?“

   Ohne dass eine Pause entsteht, antwortet der Partner *P1*:

   > „Ach‿so.‿Du‿meinst‿wir (!...!).‿Ja wir brauchen beide Werte. Wir müssen einmal wissen, wann sie das letzte mal

[198] Hier geht es darum, was mit bestimmten Daten eines Nutzers („meine Freundesliste“) passiert, wenn sich ein anderer Nutzer („dieser Freund“) aus dem System löscht.

> gefragt haben und wir müssen wissen, wann sich das letzte mal was geändert hat. (.) Die Sachen haben nichts miteinander tun. (..) Zwei vollkommen unabhängige (!!Daten!!).“

Ob *P1* hier Wissen oder eine gerade gewonnene Erkenntnis verbalisiert, ließ sich nicht (zweifelsfrei) ermitteln. Zwar deutet der erste Äußerungsteil – vor allem das „Ach‿so“ – darauf hin, dass der Sprecher zu einer Erkenntnis gelangt ist. In dieser scheint ihm aber „nur“ klar geworden zu sein, worauf sein Partner mit der Frage hinaus will. Derartige Erkenntnisse werden aber im Rahmen der BS/BKM nicht adressiert (siehe die auf S. 252 ff gelisteten Kriterien). Es bleibt der eigentliche Antwortteil. Für ihn gilt: Auch wenn man den erweiterten Kontext der Äußerung betrachtet, kann nicht entschieden werden, ob *P1* schon länger bewusst ist, dass beide Informationen benötigt werden oder ob ihm dies gerade erst klar geworden ist.

Die BS/BKM schreibt für derartige Fälle vor, *explain_knowledge* zu annotieren. In weiterführenden Untersuchungen sollte aber, falls im Rahmen des verwendeten Blickwinkels Erkenntnisse eine besondere Rolle Spielen, über andere Vorgehensmodelle nachgedacht werden.

Darüber hinaus sollten folgende das Zusammenspiel von Erkenntnissen und Bestandswissens und somit auch die Abgrenzung zwischen *explain_knowledge* und *explain_finding* betreffende Phänomene beachtet werden:

1. **Bestandswissen als Basis von Erkenntnissen**: Wie schon im Rahmen der Erläuterung des Fokus der Klasse *finding* (S. 258) angesprochen ist es der Regelfall, dass beim Erlangen von Erkenntnissen Bestandswissen eine (entscheidende) Rolle spielt. Somit kann es vorkommen, dass dieses bei der Verbalisierung einer Erkenntnis – zumindest teilweise – mit geäußert wird. In solchen Fällen sollte es nur dann separat markiert werden, wenn es vom Sprecher eindeutig/explizit als solches hervorgehoben wird (siehe auch S. 270). In Matrix 7.49 sind derartige Fälle zusammengetragen.
2. **Verbalisieren als Erkenntniskatalysator**: Eine Erklärung, die auf Bestandswissen basiert, kann direkt in die Verbalisierung einer Erkenntnis übergehen. Das mit dem Erklären einhergehende (laut) Denken kann also in eine neue Einsicht münden. Beide Teile/Phänomene sollten in diesem Fall separat kodiert werden (auch hier siehe Matrix 7.49). Beispielsweise äußert sich *P1* in Sitzung PR2.1 wie folgt:

   > „Und zwar das Problem sind, äh, wenn du die äh Columns (.) benutzen willst, auch in `all`, musst du ja für einheitliche Attributspaltennamen (.) sorgen. Und dafür musst du an der Stelle (!...!) Ach so, ne, halt, das ist kein Problem. Die haben

wir ja hier. Das heisst, die können wir in der *Factory*[199] auch rein machen."

Es ist plausibel, dass diese Phänomene nicht nur im Zusammenhang mit *explain_finding*- bzw. *explain_knowledge*-Äußerungen auftreten können, sondern auch im Rahmen von *amend_finding*- und *challenge_finding*- bzw. *challenge_knowledge*-Äußerungen (siehe auch Matrix 7.50). Dies wurde allerdings bisher nicht für jeden dieser Fälle beobachtet.

– ***explain_knowledge* vs. *amend_finding*:** Nähere Erläuterungen bzw. Ergänzungen zu einem *explain_finding* oder auch *amend_finding* bzw. *challenge_finding* müssen nicht selbst Erkenntnisäußerungen (vom Typ *finding*) sein. Durch sie kann auch ausschließlich Bestandswissen vermittelt werden (siehe z.B. Zeile 3 in Tabelle 7.48). Es stellt sich die Frage, ob derartige Äußerungen mit *amend_finding* oder *explain_knowledge* annotiert werden sollten. Im Rahmen der Herleitung der BS/BKM wurde festgelegt, auch in solchen Fällen den zentralen Prinzipien der BS/BKM zu folgen (siehe Abschnitt 7.2) und *amend_finding* zu kodieren. Informationen darüber, wie der Sprecher zu seiner ergänzenden Aussage gekommen ist, gehen auf diese Weise allerdings verloren. Falls sie benötigt werden, sollten sie beispielsweise mit Hilfe von Eigenschaften erfasst werden (z.B. $P2.amend\_finding_{origin=knowledge}$).

  An dieser Stelle soll noch einmal an folgende Festlegungen erinnert werden:

  – Zeitnah hintereinander und ohne das Auftreten von als Unterbrechung wirksamen Einwürfen des Partners geäußerte Ergänzungen zu einem *finding* werden in der Regel zusammen durch nur ein *amend_finding* kodiert (siehe S. 270 und *explain_knowledge* vs. *explain_finding* bez. der Ausnahmen). Diese und verwandte Festlegungen sind in Matrix 7.50 zusammenfassend dargestellt.

  – Im Rahmen der BS/BKM ist es nicht vorgesehen, Äußerungen, für die aus dem Kontext ersichtlich wird, dass der Sprecher einen gewissen Wissensbedarf (seines Partners) erkannt hat und in denen er diesem Bedarf dann auch nachkommt, zusätzlich als Erkenntnisverbalisierungen zu markieren – zumindest dann nicht, wenn der Sprecher nur das Bestandswissen an und für sich in Worte fasst.

[199] Hier wird das Designpattern Factory (siehe [70]) referenziert.

Eigenschaften[a] E_i **einer initiativen Wissensäußerung** A	**+: E_i gilt; (+): E_i kann gelten; −: E_i gilt nicht**			
E_1: Initiales, allerdings dem Sprecher nicht unbedingt in dieser Weise bewusstes Ziel: Erkenntnis(se) verbalisieren.	+	+	−	−
E_2: Initiales, allerdings dem Sprecher nicht unbedingt in dieser Weise bewusstes Ziel: EK vermitteln. E_1 und E_2 schließen sich gegenseitig aus.	−	−	+	+
E_3: Mit A wird EK vermittelt, ohne es explizit als solches hervorzuheben. Siehe auch S. 258.	(+)	(+)	(+)	(+)
E_4: Mit A wird EK vermittelt und explizit als solches hervorgehoben.[b]	−	+ ⇒ S/MK	(+)	(+)
E_5: Im Laufe der Äußerung A erlangt und formuliert der Sprecher eine oder mehrere weitere Erkenntnisse.	(+)	(+)	−	+ ⇒ S/MK
Kodierung:	*finding*[c]	*finding*+*knowledge**[d]	*knowledge*	*knowledge*+*finding*[e]

[a] Genau genommen handelt es sich um Werte von Eigenschaften.

[b] Solche Phänomene konnten bisher, zumindest im Kontext von *finding*-Äußerungen, nicht beobachtet werden (siehe auch S. 270). Es muss noch untersucht werden, ob bzw. wann eine Kodierung des EK angebracht ist. Aus diesem Grund sind diesbezüglich denkbare *knowledge*-Kodierungen hier mit *knowledge** bezeichnet.

[c] Siehe z.B. S. 268 und S. 270. Falls der Sprecher im Laufe von A weitere Erkenntnisse erlangt und formuliert, kann es nötig werden, diese separat zu kodieren, z.B. mittels *challenge_finding*.

[d] Siehe Fußnote *c*.

[e] Siehe Fußnote *c*.

Tab. 7.49: Gegenüberstellung verschiedener Ausformungen initiativer Äußerungen (über Eigenschaften E_i spezifiziert), bei der der Fokus auf dem Zusammenspiel zwischen Erkenntnissen im Sinne der Klasse *finding* und Bestandswissen (EK) im Sinne der Klasse *knowledge* liegt. A sei eine initiative Äußerung und bestehe aus n ($1 \leq n$) zeitlich hintereinander formulierten Teiläußerungen, die als inhaltlich zusammengehörige Wissensäußerungen (im Sinne von $W_{UK\setminus activity}$) betrachtet werden können. A enthalte keine Teiläußerungen, die nicht mit *knowledge*- oder *finding*-Konzepten zu annotieren sind – also z.B. keine Vorschläge zur Gestaltung des Produktes. Im Rahmen der Formulierung von A äußere der Partner keine unterbrechenden Einwürfe. Für A sei nicht ausgeschlossen, dass sie (kurze) Sprechpausen enthält – zumindest solange wie die Äußerungsteile nach den Pausen als direkte inhaltliche Fortsetzung der Teile vor den Pausen angesehen werden können (siehe z.B. Zeile 5 in Tabelle 7.48). A enthalte keine Teiläußerungen vom Typ *agree*, *disagree* oder *ask*. Legende: S/MK: A wenn nötig und möglich entsprechend **s**plitten, ansonsten nötigenfalls entsprechend **m**ehrfach **k**odieren.

Eigenschaften[a] E_i **einer reaktiven Wissensäußerung** A	**+: E_i gilt; (+): E_i kann gelten; −: E_i gilt nicht**						
E_1: Ist Reaktion auf *explain_*-, *challenge_*- oder *amend_finding*.	+	+	+	+	−	−	−
E_2: Ist Reaktion auf *explain_*- oder *challenge_knowledge*.	−	−	−	−	+	+	+
E_3: Initiales, allerdings dem Sprecher nicht unbedingt genau in dieser Weise bewusstes Ziel: Erkenntnis(se) verbalisieren.	+	+	−	−	−	−	+
E_4: Initiales, allerdings dem Sprecher nicht unbedingt genau in dieser Weise bewusstes Ziel: EK vermitteln.	−	−	+	+	+	+	−
E_5: Mit A wird EK vermittelt, ohne es explizit als solches hervorzuheben. Siehe auch S. 258.	(+)	(+)	(+)	(+)	(+)	(+)	(+)
E_6: Mit A wird EK vermittelt u. explizit als solches hervorgehoben.[b]	−	+	(+)	(+)	(+)	(+)	(+)
E_7: Im Laufe der Äußerung A erlangt und formuliert der Sprecher eine oder mehrere weitere Erkenntnisse.	(+)	(+)	−	+	−	+	(+)
Kodierung:	f[c]	f[d]+k^*	f ($+k^*$)	f ($+k^*$)[e]	k	k+f	Abhängig von A[f]
Begründung (falls die Kodierung nicht kanonisch ist):	S. 270	S. 270	S. 270	S. 270, 335		S. 334	

[a] Genau genommen handelt es sich um Werte von Eigenschaften.

[b] Solche Phänomene konnten bisher, zumindest im Kontext von f-Äußerungen, nicht beobachtet werden (s. auch S. 270). Es muss noch untersucht werden, wann eine entsprechende Kodierung angebracht ist. Aus diesem Grund sind diesbezüglich denkbare k-Kodierungen hier mit k^* markiert.

[c] Falls der Sprecher im Laufe von A weitere Erkenntnisse erlangt und formuliert, kann es nötig werden, diese separat zu kodieren, z.B. mittels *challenge_finding*. Siehe außerdem S. 268.

[d] Siehe Fußnote *c*.

[e] Ein Fall, der all diese Phänomene beinhaltet, wurde bisher nicht beobachtet. Zusätzliche f-Annotationen können notwendig werden.

[f] Handelt es sich z.B. um eine Erkenntnis vom Typ T_B, ist u.a. ein f zu annotieren. Insgesamt konnten aber bisher zu wenige derartige Phänomene identifiziert werden, um eine allgemeine Regel aufstellen zu können.

Tab. 7.50: Gegenüberstellung verschiedener Ausformungen reaktiver Äußerungen (über Eigenschaften E_i spezifiziert). Der Fokus der Darstellung entspricht dem in Matrix 7.49. Die diskutierte Äußerung A habe die in Matrix 7.49 beschriebene Form, mit einer Ausnahme: A sei (zumindest in Teilen) reaktiv. Legende: f: *finding*; k: *knowledge*; k^*: Siehe Fußnote in der Tabelle; (k^*): Kodierung nur notwendig, wenn die entsprechenden Bedingungen erfüllt sind. Notwendig werdende Splits sind nicht markiert.

– ***explain_knowledge* vs. *challenge_finding/disagree_finding*:** Ein Widerspruch zu einer Erkenntnisäußerung kann mittels einer Äußerung erfolgen, in der ausschließlich Bestandswissen verbalisiert wird. Derartige Phänomene werden im Rahmen der BS/BKM durch *challenge_finding* konzeptualisiert (siehe hierzu auch S. 270 und Matrix 7.50).

– ***explain_knowledge* vs. *agree_design/disagree_design*:** Zustimmungen zu wie auch Ablehnungen von Designvorschlägen werden nicht immer explizit formuliert. Es kommt vor, dass der Sprecher sie in Wissensäußerungen „verpackt“ (siehe Zeile 2 in Tabelle 7.45). Die BS/BKM schlägt vor, in solchen Fällen Doppelkodierungen (mit *explain_knowledge* und *agree_*- bzw. *disagree_design*) vorzunehmen (siehe auch 9.3.1).

  Achtung: Bei Zustimmungen bzw. Ablehnungen, die in Bezug auf Vorschläge anderer *P&P*-Klassen, z.B. *strategy*, gemacht und dabei wie oben erläutert implizit formuliert werden, sollte analog verfahren werden. Allerdings konnten solche Phänomene bisher nicht für alle *P&P*-Klassen beobachtet werden.

– ***ask_knowledge* vs. *propose_hypothesis*:** Wenn man einerseits davon ausgeht, dass es vorkommt, dass Hypothesen auch in Frageform formuliert werden bzw. zulassen möchte, dass Fragen Hypothesen darstellen können (siehe Beispiel 2 in Zeile 1 von Tabelle 7.35) und andererseits, dass es nicht ungewöhnlich ist, dass beim Stellen von Fragen Antwortmöglichkeiten mit formuliert werden (siehe Beispiel 4 in Tabelle 7.43), ist nicht mehr für jede Äußerung offensichtlich, ob sie mit *propose_hypothesis* (z.B. einem solchen vom Typ H_{uck}) oder *ask_knowledge* annotiert werden sollte. Um zu einer angemessenen Kodierung zu kommen, sollte deshalb im Zweifelsfall folgender Frage nachgegangen werden: Geht der Sprecher davon aus, dass sein Partner die gestellte Frage derzeit (zuverlässig) beantworten kann?[200] Ist dem so, ist in der Regel eine Kodierung mit *ask_knowledge* angebracht, anderenfalls eine mit *propose_hypothesis*.

  Ein derartiger Nachweis lässt sich allerdings nicht immer in vollständig nachvollziehbarer Weise erbringen. Für die beiden erwähnten Äußerungen lautet die Argumentation wie folgt:

  – **Beispiel 2 in Zeile 1 von Tabelle 7.35**: Beide Partner versuchen seit geraumer Zeit, das Verhalten einer `if`-Anweisung zu verstehen. Dieses Verhalten ist abhängig von bestimmten Übergabewerten. Es ist deshalb nicht sehr wahrscheinlich, dass der Observer davon ausgeht, dass sein Partner das Verhalten des Programms für einen neu gesetzten Übergabewert sofort zuverlässig abschätzen kann. Vor allem auch deshalb, weil der Driver – zumindest allem Anschein nach – kurz vorher

[200] Dies bedeutet nicht, dass der Partner die Frage derzeit wirklich (zuverlässig) beantworten kann.

zu einer Erklärung angesetzt, den Versuch aber nach wenigen Worten wieder abgebrochen hatte. Eine Kodierung mit *propose_hypothesis* ist also angemessen.

– **Beispiel 4 in Tabelle 7.43**: Der Partner wird direkt angesprochen („Was machst Du hier?“). Dementsprechend kommt nur ein *ask_knowledge* in Frage.

In Tabelle 7.51 ist ein weiterer kurzer Dialog dargestellt, der noch einmal verdeutlicht, wie ähnlich sich *ask_knowledge*- und *propose_hypothesis*-Phänomene sehen können.

HHI-Kode	**Äußerung**	**Kontext/Kommentar**
P2.ask_knowledge	„So‿ging‿das‿mit‿'m‿Minus, oder?“	Nachdem *P2* als erstes die Zeichenfolge „cad-507“ (den Aufgabenbezeichner) in das Kommentarfeld geschrieben hat, „wandert“ er mit dem Cursor zurück zum Minuszeichen/Bindestrich und fragt, ob dieses der vorgeschriebenen Syntax entspricht. Seine Frage beinhaltet eine potentielle Antwort. Die Äußerung in ihrem Kontext macht aber deutlich, dass es *P2* nicht um das Aufstellen einer Hypothese geht, sondern dass er den illokutionären Akt des Fragestellens vollzieht.
P1.propose_hypothesis	„Ich glaub', das ‚cad' groß oder so?“	*P1* beantwortet die Frage mit einer Hypothese. Da sein Partner schon angedeutet hat, dass er sich mit der festgelegten Syntax für die Einträge nicht gut auskennt, ist nicht davon auszugehen, dass die Intention von *P2* darin besteht, von seinem Partner eine Frage beantwortet zu bekommen.

Tab. 7.51: Beispielepisode aus PR2.1 (12:29:04–14:29:10) dafür, wie ähnlich sich *ask_knowledge*- und *propose_hypothesis*-Phänomene sehen können. Kontext: *P1* ist dabei, ein SVN-*Commit* durchzuführen und hat bereits damit begonnen, das Kommentarfeld auszufüllen. Im Unternehmen ist festgelegt, dass ein solcher Kommentar durch einen wohldefinierten Aufgabenbezeichner einzuleiten ist. Ein Teil dieser Episode wurde bereits in Tabelle 7.35 erörtert.

– ***ask_knowledge* vs. *explain_finding*:** Fragen sind gewöhnlich eine Konsequenz aus gerade gewonnenen Erkenntnissen – nämlich aus solchen, die einer Person zeigen, dass sie etwas Bestimmtes nicht weiß, nicht versteht oder nicht interpretieren kann.[201] So stellt der Driver in Sitzung PR1.1 die Frage: „Was sind 'n das für Tabs hier?“ Diese Frage ergibt sich direkt aus der von ihm gemachten, aber nicht explizit formulierten Entdeckung, dass sich Tabulatoren im Quellkode befinden (siehe auch Tabelle 7.31). In der Regel sollten derartige Äußerungen mit *ask_knowledge* kodiert werden. Nur wenn davon ausgegangen werden muss, dass es sich um eine rhetorische Frage handelt, sollte stattdessen *explain_finding* eingesetzt werden.

  Achtung: Genau genommen ist die Situation im Beispiel etwas komplizierter als erläutert. Durch die Frage werden nämlich zwei Erkenntnisse artikuliert. Zum einem die Entdeckung der Tabulatoren und zum anderen die Einsicht, dass nicht klar ist, wo diese herkommen. Weiterführende Untersuchungen, speziell solche, die sich mit im Rahmen von Paarprogrammierungssitzungen entstehenden Erkenntnissen beschäftigen, müssen derartige Phänomene evtl. berücksichtigen.

– ***explain_knowledge* vs. *explain_standard of knowledge*:** Das Phänomen, dass einer der Entwickler das vom Partner gerade übermittelte Wissen wiederholt, z.B. indem er es paraphrasiert, sollte in der Regel durch das Konzept *explain_standard of knowledge* behandelt werden (siehe S. 302). Hierbei ist allerdings Folgendes zu beachten: Im Rahmen derartiger Äußerungen kann es vorkommen, dass auch Aspekte verbalisiert werden, die vom Partner ursprünglich gar nicht geäußert wurden (siehe Zeile 2 in Tabelle 7.46). Sofern sich diese nicht eindeutig, z.B. als Äußerungen vom Typ *explain_knowledge*, separieren lassen, sollten sie dem *explain_standard of knowledge* zugerechnet werden. Achtung: In weiterführenden Untersuchungen können derartige Phänomene von besonderem Interesse sein. Ggf. müssen dann zweckdienlichere Differenzierungskriterien erarbeitet werden.

– ***ask_knowledge* vs. *explain_standard of knowledge*:** Auf S. 298 wurde bereits festgehalten, dass an den Partner gerichtete Fragen, obwohl sie ja in gewisser Weise eine Wissensstandsäußerung darstellen, nicht mit *explain_standard of knowledge*, sondern mit *ask_knowledge* (bzw. gegebenenfalls auch anderen *ask*-Konzepten) annotiert werden sollten. Aus der Festlegung folgt aber keinesfalls, dass es sich bei in Frageform formulierten Äußerungen nicht um solche vom Typ *explain_standard of knowledge* handeln kann. Entscheidend ist die (primäre) Illokution der Äußerung. Deutet beispielsweise alles darauf hin, dass es sich um eine rhetorische Frage handelt, mit der der Sprecher vor allem darauf hinweisen möchte, dass er gerade erkannt hat, dass er einen bestimmten Sachverhalt (doch) nicht versteht,

[201] An dieser Stelle wird davon abgesehen, dass es natürlich Fragen gibt, die sich eine Person schon eine längere Zeit stellt.

muss *explain_standard of knowledge* annotiert werden (siehe z.B. Zeile 5 in Tabelle 7.41).

– ***agree_knowledge* vs. *explain_standard of knowledge*:** Nicht immer kann weitgehend zweifelsfrei ermittelt werden, ob es sich bei einer Äußerung um ein *agree_knowledge* oder ein *explain_standard of knowledge* (vom Typ *AK*) handelt. Im Rahmen der Herleitung der BS/BKM wurde deshalb festgelegt, in Zweifelsfällen *agree_knowledge* zu kodieren. Hierdurch wird sichergestellt, dass den zentralen Prinzipien (siehe Abschnitt 7.2) in Teilen gefolgt wird. Darüber hinaus kann es vorkommen, dass der Sprecher beide Intentionen verfolgt und dies auch in Teiläußerungen expliziert (siehe z.B. Tabelle 7.42 oder Tabelle 7.46). Hier sollten die Teiläußerungen entsprechend kodiert werden.

Weitere Anmerkungen zu *knowledge*-Konzepten

Abschließend soll nun noch erörtert werden, in welchem Bezug der in der vorliegenden Arbeit eingeführte Begriff Bestandswissen zu den in der Literatur oft verwendeten Klassifizierungen von *deklarativem* und *prozeduralem Wissen* bzw. *deklarativem* und *nichtdeklarativem* (auch *prozedural* genanntem) *Gedächtnis* steht. Hier muss vorweg bemerkt werden, dass diese Klassen in der Regel von unterschiedlichen Autoren bzw. unterschiedlichen wissenschaftlichen Disziplinen in einer nicht vollständig identischen Weise verwendet werden. So versteht der Kognitionspsychologe John Robert Anderson[202] [3, S. 78] unter deklarativem Wissen Faktenwissen („knowledge of facts about the world“), z.B. den Inhalt einer Geschichte, unter prozeduralem Wissen solches darüber, wie man etwas tut („knowledge about how to do something“), also z.B. Regeln und Fertigkeiten. Für Anderson entspricht die Unterscheidung zwischen deklarativem und prozeduralem Wissen derjenigen zwischen Daten und Programm in der Informatik. Der Neurophysiologe Wolfram Schultz [186] beschreibt prozedurales Wissen, indem er sagt, dass „Handlungsabläufe, Fertigkeiten und motorische Tätigkeiten [...] aus Sequenzen vieler Komponenten [bestehen]. Sie werden durch viele Wiederholungen erlernt und zu Gewohnheiten, die als vollständige Sequenzen im prozeduralen Gedächtnis gespeichert werden. Diese Gewohnheiten betreffen die meisten täglichen Tätigkeiten, wie Ankleiden, Essen, Arbeitsprozesse, Freizeitbetätigungen und viele automatisierte Handlungen, die wir im Laufe eines Tages durchführen.“ Guy R. Lefrançois verweist in dem Lehrbuch „Psychologie des Lernens“ darauf, dass „aktuelle Modelle des Langzeitgedächnisses [...] eher assoziationistisch [sind] (sie gehen also davon aus, dass alles Wissen miteinander in Beziehung steht), und [...] oft zwischen explizitem, potentiell bewusstem Gedächtnis (dem so genannten deklarativen Gedächtnis) und den impliziten, unbewussten, nicht verbalisierbaren

[202] John Robert Anderson entwickelte eine Theorie zur Erklärung der kognitiven Leistung von Menschen (*Adaptive Control of Thought*), welche es u.a. ermöglichen soll, kognitive Leistungen des Menschen im Computer zu simulieren.

Wirkungen von Lernen (dem so genannten nichtdeklarativen oder prozeduralem Gedächtnis) [unterscheiden]. Zum expliziten oder deklarativen Gedächtnis gehört das semantische Gedächtnis (stabiles, abstraktes Wissen) und das episodische Gedächtnis (persönliche, autobiographische Erinnerungen, an bestimmte Zeiten und Orte gekoppelt)“ [115, S. 340].[203]

Im Rahmen der vorliegenden Arbeit soll einer Beschreibung des Professors für Psychiatrie, Neurowissenschaften und Psychologie Larry R. Squire und des Neurowissenschaftlers Eric R. Kandel gefolgt werden [195, S. 17]. Sie verstehen unter dem deklarativen Gedächtnis eines für „Tatsachen, Vorstellungen und Ereignisse – für Information, die bewußt in Erinnerung gerufen werden kann, sei es als Verbalisierung oder geistiges Bild. Es ist die Art von Gedächtnis, in dem der Name eines Freundes, der letzte Sommerurlaub, die Unterhaltung heute morgen gespeichert ist. [...] Das nichtdeklarative Gedächtnis basiert ebenfalls auf Erfahrung, drückt sich aber als Verhaltensänderung und nicht etwa als Erinnerung aus. Im Gegensatz zum deklarativen Gedächtnis ist das nichtdeklarative Gedächtnis unbewußt [...]“.

Der im Rahmen der Erläuterungen der BS/BKM verwendetet Begriff Bestandswissen unterscheidet an dieser Stelle nicht. Er adressiert sowohl Inhalte des deklarativen wie auch Inhalte des nicht deklarativen Gedächtnisses. Allerdings wird er bei der Analyse letztendlich nur im Zusammenhang mit Verbalisierungen verwendet, also bezogen auf das deklarative Gedächtnis, genau genommen – wenn man „nicht verbalisierbare geistige Bilder“ ausschließen möchte – das verbalisierbare deklarative Gedächtnis. Außerdem: Da in der oben wiedergegebenen Beschreibung von Squire und Kandel nicht näher beschrieben ist, was unter „in Erinnerung rufen“ verstanden werden soll, beziehungsweise ab wann etwas zum Gedächtnis gehören soll, adressiert der hier verwendete Begriff Bestandswissen nur einen Teil dessen, was Squire und Kandel als deklaratives Gedächtnis bezeichnen – nämlich zumindest auf bestimmte Weise identifizierbare Erkenntnisse nicht. Dieser Teil des deklarativen Gedächtnisses sei hier als *Kern des deklarativen Gedächtnisses* bezeichnet.

Wenn also im Rahmen der Erläuterungen von Konzepten der BKM der Begriff Bestandswissen verwendet wird, geht es in der Regel um den Inhalt des Kerns des deklarativen Gedächtnisses einer Person zu einem Zeitpunkt t ($t \in [Sitzungsanfang, Sitzungsende]$).

7.6.6 Das Konzept *remember_source of information*

Das Konzept *remember_source of information* adressiert Äußerungen, in denen sich ein Sprecher an bestimmte Informationsquellen erinnert, z.B. an (Teile von) Dokumentationen oder Aufgabenbeschreibungen. Hierbei handelt es sich in der Regel um solche Informationsquellen, die in Hinsicht auf die Bewältigung eines

[203] Lefrançois [115, S. 340] verwendet die Begriffe explizit und deklarativ sowie implizit und prozedural synonym. Multhaup [144] weist darauf hin, dass dies in der Fachliteratur öfter geschieht.

aktuell anstehenden Problems als potentiell hilfreich bewertet werden. So äußert *P2* in Sitzung ST1.1:

> „Da gab's doch diese Seite, wo man sich angucken konnte, was im ähmm JNDI[204]-Verzeichnis zu tun ist. (~~)"

Er erinnert auf diese Weise daran, dass dem Paar in der Lehrveranstaltung, in deren Rahmen das Experiment, dem sie gerade beiwohnen, stattfindet, JNDI-Dokumentation zur Verfügung gestellt wurde. Er geht allem Anschein nach davon aus, dass diese Dokumentation dem Paar dabei helfen kann, ein identifiziertes Problem zu lösen.

Genau genommen handelt es sich bei den von *remember_source of information* adressierten Phänomenen um eine Untermenge der von dem Konzept *explain_knowledge* adressierten. Da aber Hinweise auf bzw. Erinnerungen an Informationsquellen im Rahmen der hier geschilderten Untersuchungen als besonders interessant eingestuft wurden, sind diesbezügliche Äußerungen aus der Menge der durch *explain_knowledge* zu konzeptualisierenden herausgenommen worden. Da die Intention, sich bzw. seinen Partner an etwas zu erinnern, als etwas Wesentliches wahrgenommen wurde, ist das Konzept *remember_source of information* und nicht etwa *explain_source of information* genannt worden.

7.7 Fassadenkonzepte/*activity*-Konzepte

Fokus der Fassadenkonzepte (insbesondere der *activity*-Konzepte)

Die Klasse der Fassadenkonzepte, deren Name in Anlehnung an das Entwurfsmuster Fassade (*facade*) [70] gewählt wurde, bündelt selbst keine einzelnen Konzepte, sondern Konzeptklassen (so genannte *Fassaden eines Typs*) – und zwar solche, die den folgenden Eigenschaften genügen:

1. Sie stellen eine auf einen engen Bereich fokussierte Perspektive auf komplexere oder länger laufende verbale Vorgänge zur Verfügung, z.B. auf solche, die nicht nur an einer einzelnen Äußerung hängen (wie beispielsweise *design*). Die Bedeutung von „komplexer" bzw. „länger laufend" muss hierbei im Einzelfall (das heißt pro Fassade eines Typs) und in Abgrenzung zu den anderen Konzeptklassen der BS herausgearbeitet werden.

2. Ihre Elemente (Fassadenkonzepte eines Fassadentyps) können in der Regel unabhängig von anderen (BS-) Konzepten bzw. Konzeptklassen verwendet

[204] JNDI ist das „Java Naming and Directory Interface" (http://www.oracle.com/technetwork/java/jndi/index.html (Abruf: 29.09.2011)), eine Java-Programmierschnittstelle für Namens- und Verzeichnisdienste.

werden. Fassadenkonzepte sind somit im Normalfall von vornherein dahingehend konzipiert, im Rahmen von Doppel- oder Mehrfachkodierungen zum Einsatz zu kommen.[205]

3. Die Verwendung ihrer Elemente kann unabhängig davon erfolgen, ob es in einer Äußerung oder Folge von Äußerungen um das (zu entwickelnde) Produkt, den Prozess oder beides geht. Fassaden werden deshalb im Rahmen der BS der Klasse der *UK* zugerechnet.

Achtung: Im Software Engineering wird durch die Verwendung des Entwurfsmusters Fassade [70] eine einheitliche und vereinfachte Schnittstelle auf ein Teilsystem zur Verfügung gestellt. Überträgt man diese Sichtweise auf die hier vorgenommenen Untersuchungen, so könnte man argumentieren, dass alle Objektklassen genau dies leisten und deshalb als Fassaden zu bezeichnen wären . Beispielsweise vereinheitlicht und vereinfacht die Konzeptklasse *design* den Blick auf das Material dahingehend, dass nur noch Designentscheidungsprozesse als solche sichtbar sind. Dies macht deutlich, dass die hier angesprochene Anlehnung an das Entwurfsmuster Fassade nur sehr lose ist. Denn schließlich zeigen die drei oben erläuterten Fassadeneigenschaften, dass eine Klasse wie *design* hier eben nicht den Fassaden zuzurechnen ist.

Die BS/BKM stellt nur einen Fassadentyp zur Verfügung, die so genannten *activity*-Konzepte. Sie dienen dazu, solche Äußerungen (oder auch Folgen von zusammengehörigen Äußerungen) markieren und nachfolgend näher untersuchen zu können, die sich kommentierend auf gerade ablaufende oder gerade abgeschlossene HCI- oder HEI-Aktivitäten beziehen. So dienen sie beispielsweise dazu, Anmerkungen eines Entwicklers festzuhalten, in denen dieser Kodeänderungen, die sein Partner gerade durchführt oder durchgeführt hat, ablehnend oder zustimmend kommentiert. Die *activity*-Konzepte schließen also eine wichtige, von den anderen Elementen der BKM nicht adressierte Lücke: Die Beziehung zwischen Äußerungen und anderen Handlungen.

Im Zentrum der Klasse *activity* steht das Konzept *think aloud_activity*. Wie in Abschnitt 7.1.3 bereits angedeutet, adressiert es Äußerungen, in denen ein Entwickler seine HCI/HEI-Tätigkeiten zumindest in Teilen „on the fly“ verbalisiert. Hierbei wird nicht zwischen dem reinen Mitsprechen von Tätigkeiten und dem Verbalisieren von mit den Tätigkeiten zusammenhängenden „sonstigen Gedanken“ unterschieden. Vielmehr sind jegliche mit der aktuellen eigenen Tätigkeit zusammenhängenden Verbalisierungen mit *think aloud_activity* zu kodieren, insbesondere solche, in denen die die eigene Handlung bestimmenden Aspekte ausgeführt, kommentiert oder mit zusätzlicher Information angereichert werden. Es gilt sogar: Wenn unterschiedliche Formen derartiger Äußerungen im Rahmen eines Kommentars, das heißt im Rahmen einer ununterbrochenen Verbalisierung,

[205] Doppel- und Mehrfachkodierung muss in diesem Zusammenhang nicht unbedingt bedeuten, dass ein fester Datenausschnitt mehrfach annotiert wird. Es kann auch heißen, dass sich verschiedene, unterschiedlich zu konzeptualisierende Datenausschnitte „in einem Teilstück überlappen“.

zu einer als Einheit zu betrachtenden Tätigkeit auftreten (also im Rahmen eines Phänomens, welches mit *einem* HCI/HEI-Konzept zu annotieren ist (siehe Abschnitt 8.1)), werden sie in der Regel zusammen mit nur einem *think aloud_activity* annotiert. Beispielsweise kommentiert *P2* in Sitzung PR2.1 die von ihm gerade durchgeführten Änderungen an einer Methode wie folgt:

> „<*Beginn des Editierens der Methode*>
> Eigentlich müsste das *so* implementiert sein. (, , ,)
> <*Ende des Editierens der Methode*>
> (.)
> <*Beginn des ‚Wanderns' mit dem Cursor ab- und aufwärts*>
> Und es darf nicht, definitiv nicht, in's `Abstract` rein. (, ,) Meiner Meinung nach.
> <*Ende des ‚Wanderns' mit dem Cursor ab- und aufwärts*>"[206]

Die Äußerung wird im Rahmen der BS/BKM einerseits durch ein *think aloud_activity* markiert und andererseits in zwei Teiläußerungen gesplittet, die jeweils mit einem *propose_design* zu annotieren sind. Weitere Erörterungen zu dieser Äußerung finden sich auf S. 352.

Achtung: Endet ein mit *einem* HCI/HEI-Konzept zu annotierendes Phänomen und beginnt unmittelbar nachfolgend ein neues, ebenfalls in die Klasse der HCI/HEI-Phänomene fallendes, muss in der Regel auch ein evtl. parallel stattfindendes *think aloud_activity* als beendet angesehen werden. Nachfolgende aktionsbezogene Äußerungen sollten dann durch eine oder mehrere weitere *think aloud_activity*-Annotationen markiert werden. Kurz gesprochen: Ein *think aloud_activity*-Phänomen ist im Rahmen der BS an genau eine Handlung (im Sinne eines HCI/HEI-Phänomens) gebunden (umgekehrt gilt dies nicht – siehe hierzu auch Abbildung 7.16). Dies schließt aber, wie noch diskutiert werden wird, nicht aus, dass *think aloud_activity*-Phänomene in bestimmten Fällen über die zeitlichen Grenzen der Handlung „hinausragen" können. Es entsteht allerdings eine Kopplung zwischen der Granularität der HCI/HEI-Konzepte und der Granularität von *think aloud_activity*-Annotationen. Je mehr unterschiedliche HCI/HEI-Konzepte eingeführt werden, desto mehr *think aloud_activity*-Annotationen werden potentiell notwendig. Man betrachte hierzu noch einmal das letzte Beispiel. Sollte in weiterführenden Untersuchungen von Interesse sein, wie Entwickler, speziell Driver, handeln, wenn sie sich nicht hundertprozentig sicher sind, was als nächstes zu tun ist, könnte versucht werden, Phänomene festzuhalten, die Übersprungshandlungen ähneln. Hier wäre es dann evtl. angebracht, ein keinem unmittelbaren Zweck zu dienen scheinendes „Wandern mit dem Cursor" als separates *do_displacement activity* festzuhalten. Hieraus würde folgern, dass die zweite Äußerung im obigen Beispiel ein eigenständiges *think aloud_activity* darstellt. Weiterführende Untersuchungen müssen entscheiden, in welchem Maß sie der hier festgelegten Kopplung genügen wollen.

[206] Mit `Abstract` bezeichnet *P2* die Klasse `AbstractFeatureAttributeConfiguration`, in der er gerade Änderungen vorgenommen hat.

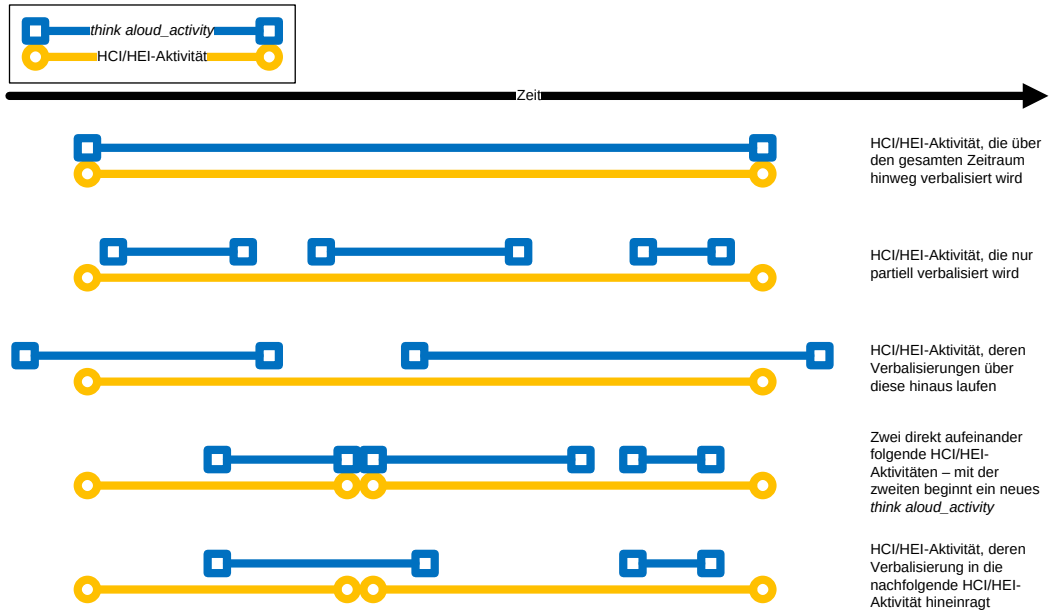


Abb. 7.16: Beispiele für unterschiedliche Formen von Verbalisierungen von HCI/HEI-Aktivitäten durch *think aloud_activity*-Äußerungen. Achtung: Es werden nur Aussagen über die Länge bzw. den Start- und Endzeitpunkt von *think aloud_activity*-Äußerungen gemacht und keine in Hinsicht darauf, wie detailliert oder vollständig eine einzelne HCE/HEI-Aktivität erläutert wird.

In der Regel werden durch eine *think aloud_activity*-Verbalisierung eines Entwicklers mehrere der folgenden Aspekte/Fragen erörtert:

TA1: Was mache ich gerade? Dies kann z.B. durch das Mitsprechen von Eingaben und Mausklicks deutlich gemacht werden. Ist in vielen Fällen mit *design*-Konzepten, insbesondere *propose_design*, zu kodieren.

TA2: Warum mache ich das, was ich gerade mache bzw. will ich (zeitnah) etwas Bestimmtes machen? Ist in der Regel ein *explain_knowledge*.

TA3: Wie mache ich das, was ich gerade mache bzw. wie will ich etwas Bestimmtes (zeitnah) machen? Hierbei handelt es sich um Erläuterungen zur Durchführung von Tätigkeiten und zur Modifikation von Inhalten. Ist in der Regel vom Typ *knowledge*.

TA4: Welche Entscheidungen fälle ich bei der Durchführung? Ist oft mit *design*- oder *step*-Konzepten, insbesondere *propose_design* oder *propose_step* zu annotieren.

TA5: Welche Entscheidungen liegen meiner Tätigkeit zu Grunde? Hierbei kann es sich z.B. um Strategien handeln.

TA6: Welche Einsichten (die evtl. die nächsten Schritte beeinflussen) erlange ich im Rahmen der Durchführung? Sie sind häufig über *finding*-Konzepte zu konzeptualisieren.

TA7: Wie bewerte/deute ich zwischenzeitliches Feedback durch den Rechner? Es handelt sich in der Regel um Äußerungen vom Typ *finding*.

TA8: Wie sicher bin ich mir dessen, was ich mache? Kann z.B. ein *explain_standard of knowledge* sein.

TA9: Wie beurteile ich die Qualität dessen, was ich mache? Hierbei kann es sich z.B. um ein *explain_finding* handeln.

TA10: Wie beurteile ich den Fortschritt dessen, was ich mache? Ist in der Regel vom Typ *completion*.

TA11: Wie beeinflusst mich die Infrastruktur bei dem, was ich mache? Ist in der Regel mit *knowledge*- oder *finding*-Konzepten zu kodieren.

In Tabelle 7.52 werden Beispiele hierzu erörtert.

Das Konzept *think aloud_activity* ist also eine „Fassade“, die von Details der Verbalisierung (z.B. einem *propose_step*) abstrahiert. Dabei stellt es zwei Merkmale in den Vordergrund,

- den Bezug der Äußerung bzw. der Folge von Äußerungen auf die aktuell ablaufende HCI/HEI-Aktivität und
- das zumindest potentiell stärkere Einbinden des Partners in die Motivationen, Entscheidungen, Ziele etc. des Handelnden. Ob dies bewusst oder unbewusst erfolgt bzw. welche Intentionen dem Akt der Verbalisierung zu Grunde liegen, wird dabei (erst einmal) nicht berücksichtigt.

Identifizierte *activity*-Konzepte und ihre speziellen Eigenschaften

In den analysierten Sitzungen wurden sechs unterschiedliche *activity*-Konzepte identifiziert:

- Konzepte, mit denen Äußerungen adressiert werden, in denen sich der Sprecher auf eigene aktuell stattfindende HCI/HEI-Aktivitäten bezieht: *think aloud_activity*
- Konzepte, mit denen Äußerungen adressiert werden, in denen sich der Sprecher auf aktuell stattfindende HCI/HEI-Aktivitäten des Partners bezieht: *challenge_activity*, *disagree_activity*, *amend_activity*, *stop_activity* und *agree_activity*[207]

[207] Auf S. 365 wird erläutert, dass es in weiterführenden Untersuchungen sinnvoll sein kann, diese Konzepte auch bei Äußerungen einzusetzen, mit denen sich der Sprecher auf die eigenen aktuell stattfindenden HCI/HEI-Aktivitäten bezieht.

Typen	*P1.think aloud_activity* [Dauer]	Kontext/Kommentar
TA1, TA4, TA11	„<*TA11 (Start *write_sth*):*> Ich⌣muss⌣mich⌣jedes⌣mal⌣wieder⌣umgewöhnen, weil ich keine Apfeltaste mehr hab. (,) <*P2: „([A]hm.)“*> <*TA4:*> So, und zwar machen wir da einfach auch *localhost*, würd' ich sagen. (, , , , ,) (Und dann ist es)(!...!) (!!(.)!!) <*P2: „Den rufst Du aus der Shell auf, ne?“*> Ja. (,) <TA1:> (##`projects` ähm <**Teil eines URL**> (..) `trunc` (.) `api` (, ,) <**Teil eines URL**>##) (~~). <*Ende *write_sth*><*TA11:*> Der ist ganz schön klein, der Screen.“ [33 Sekunden]	*P1* ist dabei, den URL in einem cURL-Aufruf[208] in einem Shellskript zu editieren. Er weist dabei auch auf zwei Eigenschaften des Settings hin, die anders sind, als an seinem Arbeitsplatz. *P2* macht zwei Einwürfe. Beim Zweiten handelt es sich um eine Frage, die *P1* beantwortet, ohne das Editieren zu unterbrechen.
TA2	„<*TA2: (*do_sth* läuft bereits)*>Hmmm. *Fonts and Colors* gibt's doch irgendwo. <**do_sth* läuft weiter*>“ [3 Sekunden]	*P1* hat bereits die *Preferences* der IDE Eclipse aufgerufen. Er öffnet nun einzelne Knoten im dargestellten *Treeview*[209]. Dabei erläutert er, dass er auf der Suche nach einem Eintrag *Fonts and Colors* ist.
TA3, TA5, TA6	<*Start *do_sth*> (, , ,) <*TA5:*> O.K. wir machen das nicht mit *TextPad*[210]. (, ,) Auf keinen Fall. (, ,) <*TA6:*> (~Da) müssen wir ihm erst mal sagen, womit er's öffnen soll. (, , ,) <*P2: „Kannst Du es nicht rüber ziehen? (~~)“*> (Ohh Gott) (,) <*TA3:*> (~Da müssen wir das hier sagen: *Open in editor*.<**do_sth* läuft weiter*>)“ [27 Sekunden]	*P1* hat vor, eine PHP-Datei zu öffnen. Im Navigator[211] der IDE Eclipse tätigt er deshalb einen Doppelklick auf den Dateinamen. Die Datei wird daraufhin nicht im Editor von Eclipse, sondern in einem externen Programm (*TextPad*) geöffnet. *P1* erläutert, dass dies nicht erwünscht ist und beendet *TextPad*. Danach öffnet er die Datei über das ihr im Navigator zugeordnete Kontextmenü. Daraufhin erscheint die Datei wieder in *TextPad*. Er quittiert dies mit einem „Ohh Gott“, schließt den Editor wieder und versucht es erneut mit dem Kontextmenü. Diesmal wählt er aber einen anderen Menüpunkt. Den letzten Schritt erläutert er. Auf den Vorschlag seines Partners geht *P1* nicht ein.
		Fortsetzung auf der nächsten Seite.

[208] Siehe Fußnote *a* in Tabelle 7.43.

[209] Der Zugriff auf die Einstellungen der IDE Eclipse erfolgt hierarchisch. Als Steuerelement (*widget*) wird ein Baum bzw. *Treeview* verwendet.

[210] Bei der Applikation *TextPad* handelt es sich um einen Editor.

[211] Der Navigator von Eclipse liefert eine Dateiansicht in Baumform.

		Fortsetzung der Tabelle der vorherigen Seite.
TA4, TA7	„(~~) <*Start *verify_sth*> (, , ,) <*TA4:*> Dann können wir's hier gleich nochmal probieren. (, , , ,) <*TA7:*> O.K. der Connect will immer noch nicht. <*Ende *verify_sth*>“ [13 Sekunden]	Nachdem *P1* eine Änderung an einer Konfigurationsdatei vorgenommen hat, startet er in einer Konsole (zum wiederholten Mal) den Aufruf `wget "http://dev-intern"`[212]. Der Aufruf bricht mit der Meldung „Verbindungsaufbau abgelehnt“ ab. Dies erkennt *P1*. Genau genommen hat *P1* die Tätigkeit zu dem Zeitpunkt, an dem er seine Erkenntnis äußert, bereits abgeschlossen. Da sie aber der Tätigkeit unmittelbar folgt, wird sie im Rahmen der BS trotzdem dem *think aloud_activity* zugerechnet.
TA1, TA4, TA9	„<*TA4 (Start *do_sth*):*> O.K. dann (~) wir Suche. (, ,) Ähh, <*TA1:*> (##*Window*, *Show View*, *Search*##). (.) <*TA9:*> Und, ich hab' nämlich wieder mal hier so blöd geöffnet. (, ,) <*TA1:*> Ich muss das nämlich eigentlich so machen. <*TA1:*> Sonst macht er das nämlich immer auf dem gleichen Tab auf, dieses Suchergebnis. (.) <*TA9:*> Hm. Das hat natürlich einen Nachteil. <*P2: „Das wir nicht direkt bei dem (~Ziel) sind.“*> Das wir nicht direkt beim (~) sind. [...] <*Ende *do_sth*>“ [Insgesamt 42 Sekunden]	Im Vorfeld hatte das Paar im Quellkode nach Aufrufen einer bestimmten PHP-Funktion gesucht. Hierzu hatten sie die Suchfunktionalität der IDE Eclipse verwendet. Nun brauchen sie die Ergebnisse der Suche wieder. Allerdings wurde der *Search View* zwischenzeitlich geschlossen. *P1* will ihn nun wieder öffnen und spricht mit, welche Menü- und Untermenüaufrufe er hierzu durchführt („*Window*, *Show View*, *Search*“). Außerdem erläutert er, dass er die Dateien, in denen die Aufrufe vorkommen und die im Search View aufgelistet sind, anders öffnen sollte, als er es zuvor getan hat. Er führt eine solche alternative Operation durch, indem er im *Search View* die Operation *Open* in einem Kontextmenü eines Suchergebnisses aufruft. Er stellt fest, dass die von ihm propagierte Methode nicht optimal ist, weil zwar die entsprechende Datei geöffnet wird, als Sichtbereich aber der Anfang der Datei erscheint und nicht die Stelle, an der der gesuchte Funktionsaufruf steht.
TA4, TA6, TA10	„<**do_sth* läuft bereits*> [...] <*TA4:*> (Schauen‿wir‿mal.) Was hat'n Courier? (, , , , , , , , , , , ,). Ihhh. (, ,) <*TA10:*> Kein toller Fortschritt. <*TA6:*> Das Problem ist halt, dass er nicht viel kleiner geht. [...] <Ende *do_sth*>“ [Insgesamt 21 Sekunden]	*P1* befindet sich in den *Preferences* von Eclipse. Er möchte die Art und Größe der im Editor dargestellten Schrift mit dem Ziel ändern, mehr Kode ohne Scrollen in Ansicht zu bringen. Um dies zu erreichen, navigiert er zu der Stelle, an der der Schriftgrad der Schriftart Courier geändert werden kann. Hier kann als kleinster Wert *10* gewählt werden. Nachdem er diesen Wert eingestellt hat, erkennt er, dass die Änderung die Situation nicht wirklich verbessert hat.

Tab. 7.52: Beispiele aus Sitzung PR1.1 für unterschiedliche Formen von Äußerungen vom Typ *think aloud_activity*. Die verwendeten HCI/HEI-Konzepte (z.B. *write_sth*) werden erst in Abschnitt 8.1 erläutert und können vorerst ignoriert werden.

[212] *Wget* ist ein Kommandozeilenprogramm zum Herunterladen von Dateien aus dem Internet (http://www.gnu.org/software/wget/ (Abruf: 08.08.2011)).

Soweit dies noch nicht in der Einleitung des vorliegenden Abschnitts oder in Abbildung 7.3 geschehen ist, werden nun die Eigenschaften dieser Konzepte erläutert. Hierbei gilt: Mit vielen der in diesem Zusammenhang beschriebenen Regeln wird das Ziel verfolgt, möglichst große zusammenhängende Blöcke von Äußerungen mit *activity*-Konzepten, in der Regel mit *think aloud_activity*, markieren zu können. So gibt es beispielsweise Regeln, die festlegen, dass bestimmte Formen von Sprechpausen nicht berücksichtigt werden sollten. Es muss beachtet werden, dass diese Regeln in weiterführenden Analysen in besonderem Maß zur Diskussion gestellt werden sollten. Beispielsweise kann es dazu kommen, dass Forscher im Rahmen von Untersuchungen, in denen es eine Rolle spielt, ob und in welcher Weise Erläuterungen zu HCI/HHI-Aktivitäten stockend formuliert werden, dazu gezwungen sind, von diesen Regeln abzuweichen.

1. **Granularität von *think aloud_activity*-Phänomenen:** Bei der Verwendung von *think aloud_activity* werden Äußerungen oder Folgen von Äußerungen adressiert, in denen sich ein Entwickler (in der Regel der Driver) auf eine seiner eigenen HCI/HEI-Aktivitäten bezieht. Endet die HCI/HEI-Aktivität, endet somit in der Regel definitionsgemäß auch das *think aloud_activity* bzw. geht in ein neues über. In diesem Zusammenhang stellt sich eine Reihe von Fragen:

   i. Wann kann oder will man davon sprechen, dass eine HCI/HEI-Aktivität beginnt bzw. endet?

   ii. Wie ist zu verfahren, wenn eine Erläuterung länger dauert als die eigentliche HCI/HEI-Aktivität bzw. schon vor der Aktivität beginnt?

   iii. Wie viele handlungsbestimmende Aspekte können maximal unerwähnt bleiben, damit noch davon gesprochen werden kann, dass es sich um eine *think aloud_activity* handelt – bzw. wie viele müssen mindestens erwähnt werden?

   iv. Wie kontinuierlich muss eine Erläuterung sein, das heißt, wie viele oder wie große Sprechpausen dürfen in einer Erläuterung vorkommen, damit noch von einem zusammenhängenden *think aloud_activity* gesprochen werden kann bzw. soll?

   v. Wie sollen Abschweifungen vom Thema die Anwendung des Konzeptes *think aloud_activity* beeinflussen?

   vi. Wie wirken sich Unterbrechungen bzw. Einwürfe durch den Partner, z.B. Fragen auf die Verwendung des Konzeptes *think aloud_activity*, aus?

   Die Antworten auf diese Fragen fallen pragmatisch aus und sollten im Rahmen weiterführender Untersuchungen bez. ihrer Angemessenheit hinterfragt werden.

Zu i.: Letztendlich kann auf diese Frage erst im Rahmen der Erörterung der HCI/HEI-Konzepte detailliert eingegangen werden (siehe Kapitel 8). Hier sei nur vorweggenommen, dass es beispielsweise im Rahmen von Kodiertätigkeiten weder sinnvoll ist, Aktivitäten grundsätzlich auf Anweisungs-, Zeilen-, Funktions-, Methoden- bzw. Klassenebene oder auf Basis sonstiger syntaktischer Elemente vorzunehmen, noch als zweckdienlich angesehen werden kann, das Kodieren einer Person so lange als eine zusammenhängende Aktivität zu werten, wie von ihr weitgehend unterbrechungsfrei (was immer dies auch konkret bedeuten soll) Editierschritte vorgenommen werden. Vielmehr ist es angebracht, Kodieraktivitäten über die mit ihnen zusammenhängenden, sich auf inhaltliche Aspekte beziehenden, aber nicht notwendigerweise verbalisierten Arbeitsschritte zu definieren, also eine Segmentierung auf inhaltlicher Ebene vorzunehmen. Dies bedeutet, dass die Granularität der HCI/HEI-Aktivitäten an die Granularität der vom Driver angegangenen Arbeitsschritte gekoppelt wird. Driver-Wechsel sollten hierbei als das Beenden einer Aktivität gewertet werden.

Zu ii.: Äußerungen bzw. Äußerungsteile, in denen der Handelnde seine HCI/HEI-Aktivitäten begleitend kommentiert, werden auch dann einem *think aloud_activity* zugerechnet, wenn sie in die eigentliche Tätigkeit „hinein-“ oder aus dieser „herauslaufen“ (siehe auch Abbildung 7.16). Allerdings sollte versucht werden, den auf diese Weise erfassten „Ein-“ bzw. „Auslauf“ so kurz wie möglich zu halten. Die entsprechenden Äußerungen sollten z.B. dahingehend analysiert werden, ob sie direkt aus dem Handeln erwachsen sind oder dieses Handeln nachträglich betrachten.

Als Beispiel sei zum einen auf den in Zeile 4 in Tabelle 7.52 dargestellten Grenzfall verwiesen: Unmittelbar nachdem *P1* das *Wget*-Kommando abschickt hat, gibt das Programm eine Fehlermeldung aus. *P1* reagiert im selben Moment. Somit scheint es angebracht, die Äußerung noch dem *think aloud_activity* zuzurechnen.[213] Ein weiteres Beispiel ist in Abbildung 7.17 zu finden. Zum anderen ist in Kapitel 8.1 in Tabelle 8.6 ein Beispiel dafür zu finden, dass ein Driver noch während er einen Vorgehensvorschlag macht, mit dessen Umsetzung beginnt. Somit ist der in die HCI/HEI-Aktivität hineinlaufende Vorschlag einem *think aloud_activity* zuzurechnen.

Zu iii.: Bei einer Kodierung von Verbalisierungen durch ein *think aloud_activity* ist es unerheblich, wie viele der direkt oder indirekt handlungsbestimmenden Aspekte durch den handelnden Entwickler

[213] Im Zusammenhang mit Erläuterungen zum Konzept *explore_sth* wird auf S. 383 erläutert werden, dass, obwohl die im Beispiel wiedergegebene Erkenntnis in gewisser Weise nur aufgrund eines Erkundens des Ergebnisses möglich war, dieses Erkunden im Rahmen der Herleitung der BS/BKM nicht separat als *explore_sth* markiert wurde.

verbalisiert werden. Es ist ausreichend, wenn er *einen* solchen Aspekt thematisiert.

Zu iv.: Redepausen unterschiedlicher Natur und Länge treten beim Sprechen in natürlicher Weise auf. Ihr Ausmaß bzw. ihre Bedeutungen ergeben sich dabei in weiten Teilen auch aus persönlichen „Sprecheigenschaften". Somit erscheint es wenig sinnvoll, Regeln festzulegen, mit denen in absoluten Werten unverrückbar festgelegt wird, wie groß eine Redepause sein darf, damit nicht vom Enden oder Unterbrechen eines *think aloud*s gesprochen werden sollte. An dieser Stelle sollte man sich noch einmal vor Augen halten, welcher Zweck mit der Kodierung von *think aloud_activity*-Phänomenen im Rahmen der BS verfolgt wird: Es sollen solche Äußerungen markiert werden, die sich auf eine als eine Einheit betrachtbare HCI/HEI-Aktivität beziehen und in denen ein Driver seinem Partner (bewusst oder unbewusst) ermöglicht, ihm nicht nur anhand der HCI/HEI-Aktivität selbst, sondern auch durch Verbalisierungen dieser zu folgen.[214] Pausen, die der Sprecher dabei einlegt, z.B. um einen Gedanken klarer fassen zu können, sollten dementsprechend in der Regel nicht als Beenden eines *think aloud*s gewertet werden. Allerdings kann es im Laufe einer Untersuchung vor dem Hintergrund neuer Erkenntnisse notwendig werden, ehemals zusammen kodierte Äußerungen aufzusplitten und einzeln zu behandeln. Man betrachte hierzu noch einmal zwei bereits diskutierte Beispiele:

- In der auf S. 345 erläuterten Äußerung entsteht eine ca. dreisekündige Sprechpause, weil der Driver die „eigentlichen" Editierschritte beendet. Danach fährt er (fast) unmittelbar mit seiner Äußerung fort, indem er einen weiteren mit dem Editierschritt zusammenhängenden Designvorschlag macht. Solange man das dem Editieren folgende „Wandern mit dem Cursor" hierbei noch als letzten Ausläufer der eigentlichen Modifikationstätigkeit wertet, scheint es angebracht, die gesamte Äußerung ungeachtet der Pause mit einem *think aloud_activity* zu kodieren. Kommt man aber, wie bereits auf S. 345 diskutiert, im Laufe seiner Untersuchungen zu der Einsicht, dass Übersprunghandlungen des Drivers von Interesse sind und das Wandern mit dem Cursor eine solche darstellt, ist der zweite Teil einzeln mit einem eigenen *think aloud_activity* zu kodieren.
- In Beispiel 6 in Tabelle 7.52 macht der Sprecher eine Sprechpause von ca. 15 Sekunden. Sie kommt dadurch zustande, dass er den zu Beginn angekündigten Arbeitsschritt („(Schauen‿wir‿mal.) Was hat'n Courier?") durchführt. Noch beim Sprechen beginnt er damit, durch die Benutzereinstellungen (*Preferences*) von Eclipse bis

[214] An dieser Stelle soll davon abgesehen werden, dass Verbalisierungen vom Typ *think aloud_activity* natürlich auch ganz anderen Nutzen entfalten können.

zu der Stelle, an der der Schriftgrad der Schriftart Courier geändert werden kann, zu navigieren. Dann ändert er die Schriftart von Tahoma auf Courier, den Wert für den Schriftgrad auf den kleinsten einstellbaren (10) und wendet seine Änderungen an (durch Betätigung der Schaltfläche *Apply*). Danach fährt er mit dem *think aloud* fort und verbalisiert das (nicht die Erwartungen erfüllende) Ergebnis („Ihhh. (, ,) Kein toller Fortschritt. Das Problem ist halt, [...]“). Während er dies tut, ruft er die Schriftart- und Schriftgradeinstellung erneut auf. Da sich alle Äußerungen auf eine als eine Einheit auffassbare Aktivität beziehen und die (zugegebenermaßen lange) Sprechpause nur der Tatsache geschuldet ist, dass es – außer evtl. die einzelnen Mausklicks mitzusprechen – in diesem Moment wohl nichts zu erläutern gibt, wurde die Folge von Äußerungen im Rahmen der Herleitung der BS/BKM als ein *think aloud_activity* aufgefasst.

– Ein weiteres Beispiel ist in Abbildung 7.17 zu finden.

Die Beispiele verdeutlichen, dass im Rahmen der BS auch längere Sprechpausen nicht unbedingt zum Aufsplitten einer Äußerung oder Folge von Äußerungen in mehrere *think aloud_activity*-Phänomene führen müssen. Entscheidend ist vielmehr, als was die Pause vom Kodierenden in Hinsicht auf die gerade ablaufende Aktivität interpretiert wird. Nur wenn zu erkennen ist (intersubjektiv nachvollziehbar dargestellt werden kann), dass der Sprecher damit beginnt, Tätigkeiten in keiner Weise mehr zu erläutern oder mit verbalen Informationen anzureichern, sollte dies als Ende eines *think aloud_activity* betrachtet werden – auch dann, wenn er einige Zeit später wieder anfängt, Erklärungen abzugeben (neues *think aloud_activity*). Eine Formulierung einer solchen intersubjektiv nachvollziehbare Erklärung kann allerdings in manchen Fällen schwierig sein, vor allem dann, wenn die Handlung nicht vorrangig physischer Natur ist, wie z.B. bei vielen Phänomen vom Typ *explore_sth*, also bei solchen, in denen Artefakte erkundet werden (siehe Abschnitt 8.1.3).[215] Weiterführende Untersuchungen müssen hier evtl. konkretere Regeln, möglicherweise sogar solche, die inhaltliche Ausrichtungen der Verbalisierungen berücksichtigen, entwickeln bzw. gänzlich anders vorgehen. Hierbei muss aber der Nutzen im Auge behalten werden, den die *think aloud_activity*-Kodierungen erbringen sollen.

Zu v.: Abschweifungen, die noch in einem erkennbaren Zusammenhang mit der Aktivität stehen, sollten dem gerade laufenden *think*

[215] Eine Möglichkeit, an dieser Stelle systematischer vorzugehen, besteht darin, in einem ersten Schritt nur alle zusammenhängenden, auf eine Aktivität bezogenen Äußerungen eines Drivers zu beachten und mit einem Konzept wie z.B. *part of (think aloud_activity)* zu annotieren. Erst in einem zweiten Schritt sollte dann überprüft werden, welche dieser Kodierungen intersubjektiv nachvollziehbar zu einem *think aloud_activity* zusammengefasst werden können.

aloud_activity zugerechnet werden. So erläutert der Driver *P1* in Sitzung PR1.1 im Rahmen eines Editierschrittes, welche, in seinen Augen gerade hilfreiche Syntaxerweiterung er sich für die Programmiersprache PHP wünscht. Diese Äußerung wurde nicht als *think aloud_activity*-abbrechend gewertet. Da nicht viele als Abschweifung klassifizierbare Phänomene beobachtet werden konnten, müssen detailliertere Ausführungen zu diesem Thema weiterführenden Untersuchungen überlassen werden. Hier muss auch analysiert werden, inwieweit es durch Abschweifungen zu Unterbrechungen der HCI/HEI-Aktivität kommt.

Zu vi.: Einwürfe durch den Partner, z.B. Fragen, auf die en passant oder gar nicht reagiert wird, sollten nicht als *think aloud_activity*-unterbrechend bzw. -abbrechend gewertet werden. Beispiele hierzu sind in Abbildung 7.17 zu finden. Hingegen sind Einwürfe, die als Zäsur wirken, also z.B. ein vom Driver berücksichtigtes *disagree_activity*, als Indiz für das Ende eines aktuell ablaufenden *think aloud_activity* anzusehen. Allerdings muss bemerkt werden, dass erst weiterführende Untersuchungen, z.B. solche, die sich im Detail mit derartigen Unterbrechungen durch den Observer auseinander setzen, zu letztendlich angemessenen Regeln kommen können. Schließlich kann derzeit die Bedeutung und Wirkung von Unterbrechungen/Einwürfen und somit auch eine angemessene Granularität von *think aloud_activity*-Phänomenen nur grob abgeschätzt werden.

2. ***think aloud_activity*-Phänomene, die in Fragen münden:** In den Fällen, in denen ein Driver im Rahmen der Erläuterung seiner HCI/HEI-Aktivitäten eine Frage an den Partner richtet, muss eruiert werden, ob dies als Abschluss der *think aloud_activity*-Äußerung zu werten ist. Für den Fall, dass gleichzeitig auch die zugehörige Aktivität abgebrochen bzw. unterbrochen wird, ist genau dies bereits festgeschrieben (ein Beispiel ist in Tabelle 7.53 zu finden). Läuft die Aktivität hingegen weiter, sind zwei Fälle zu unterscheiden:

   – Falls der Driver nachfolgend keine Sprechpause einlegt (und sofern seine weiteren Äußerungen im Zusammenhang mit seiner Aktivität stehen), sollte dies dahingehend interpretiert werden, dass er sich weiterhin im selben *think aloud_activity* befindet.
   – Legt er hingegen eine Sprechpause ein, sollte entsprechend den Erörterungen auf S. 352 f verfahren werden.

   Achtung: Ob einzelne isolierte, vom Driver parallel zu einer seiner HCI/HEI-Aktivitäten geäußerte Fragen auch als *think aloud_activity* gewertet werden sollten, muss separat in weiterführenden Analysen entsprechend ihrer Untersuchungsziele sowie der im Rahmen der Untersuchungen bereits

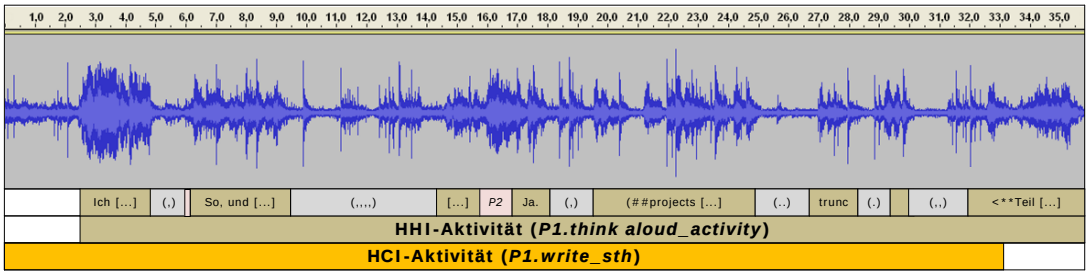


Abb. 7.17: Es ist dargestellt, wie mit Sprechpausen und Einwürfen des Partners beim Einsatz des Konzeptes *think aloud_activity* umzugehen ist. Dargestellt ist eine kurze Episode aus Sitzung PR1.1, in der folgendes geäußert wird:
„<*P1 (Driver):*> Ich‿muss‿mich‿ jedes‿mal‿wieder‿umgewöhnen, weil ich keine Apfeltaste mehr hab. (,) <*P2: „([A]hm.)“*> So, und zwar machen wir da einfach auch *localhost*, würd' ich sagen. (, , , , ,) (Und dann ist es) (!...!) (!!(.)!!) <*P2: „Den rufst Du aus der Shell auf, ne?“*> Ja. (,) (##`projects` ähm <**Teil eines URL**> (..) `trunc` (.) `api` (, ,) <**Teil eines URL**>##) (~~). Der ist ganz schon klein, der Screen.“
Der Kontext der Äußerung ist in der ersten Zeile der Tabelle 7.52 erläutert. In den Zeilen der vorliegenden Abbildung sind folgende Informationen dargestellt: In Zeile 1 die Zeitachse in Sekunden; in Zeile 2 die Wellenform (grafische Darstellung des aufgezeichneten Audiosignals; bei den sehr kurzen Peaks handelt es sich um Tastaturgeräusche (mechanisches Klicken beim Schreiben); diese treten dementsprechend zum Teil auch dann auf, wenn beide Entwickler schweigen); in Zeile 3 die Teiläußerungen in jeweils stark abgekürzter Form (Verbalisierungsteile von *P1*: braun; Sprechpausen von *P1*: grau; Einwürfe von *P2*: rosa (wobei in diesem Fall der Inhalt nicht einmal verkürzt wiedergegeben wird)); in Zeile 4 das *think aloud_activity*-Phänomen als Ganzes; in Zeile 5 die HCI-Aktivität als Ganzes (Editieren eines Shell-Skriptes). Es ist zu erkennen, dass das „Mitsprechen“ der HCI-Aktivität nicht direkt mit der Aktivität beginnt, aber über diese hinausläuft. Außerdem ist zu sehen, dass *P1* Sprechpausen einlegt. In den meisten dieser Pausen ist er mit Schreiben beschäftigt. Grund: Das Schreiben benötigt oft mehr Zeit als die Verbalisierung dessen, was geschrieben wird oder werden soll. Manche kurze Sprechpausen rühren allem Anschein nach auch daher, dass *P1* nachdenken muss. Die Verbalisierung reißt allerdings an keiner Stelle unvermittelt ab, so dass sie im Rahmen der BS als Ganzes mit einem *think aloud_activity* zu kodieren ist. Hieran ändern auch die Einwürfe des Partners nichts.

erlangten (vorläufigen) Erkenntnisse entschieden werden. Bei der Herleitung der BS/BKM wurden derartige Fragen als *think aloud_activity*-Äußerungen gewertet, falls sie einen zumindest indirekten Bezug zur Aktivität aufwiesen.

3. **HCI/HEI-Aktivitäten als Folge einer Äußerung:** Genauso, wie HCI/HEI-Aktivitäten zu begleitenden Äußerungen führen können, ist es auch umgekehrt möglich, dass Äußerungen in begleitende HCI/HEI-Aktivitäten münden. So wurde beispielsweise in Sitzung PR1.1 beobachtet, dass ein Entwickler im Rahmen der Erläuterung einer Erkenntnis (*P1.explain_finding*) mit der Maus (und unter Zuhilfenahme von Scrollen) auf einzelne Stellen im Kode zeigt (*P1.show_sth* – nähere Erläuterungen zu diesem Konzept sind in Abschnitt 8.1.7 zu finden). Derartige Handlungen entsprechen bez. ihrer Intention nicht der Ausrichtung der Klasse *activity* und sind dementsprechend nicht durch Elemente dieser zu konzeptualisieren. Weiterführende Untersuchungen (z.B. solche, die sich mit dem Thema *Awareness* beschäftigen) müssen sich damit beschäftigen, ob und in welcher Weise diese Phänomene – z.B. unter Verwendung einer weiteren Fassadenklasse – in die Analyse eingehen sollten.

   Achtung: Natürlich kann es zu Mischformen der beiden hier angesprochenen Phänomentypen kommen. So ist es beispielsweise im Rahmen eines bereits laufenden *think aloud_activity* möglich, dass der Driver eine Erkenntnis erläutert und im Laufe seiner Verbalisierung auf Kodestellen zeigt. In derartigen Fällen muss, falls es in Hinsicht auf die aktuellen Analysen angebracht erscheint, evaluiert werden, ob bzw. in welcher Weise es hierbei zu einer Unterbrechung des *think aloud_activity* kommt oder eben auch nicht kommt.

4. **Unabhängigkeit von HCI/HEI-Aktivität und Verbalisierung:** Damit eine Äußerung oder Folge von Äußerungen als *think aloud_activity* bezeichnet werden kann, muss sie – wie bereits dargelegt – einen Bezug zur gerade ablaufenden HCI/HEI-Aktivität aufweisen. Äußerungen bzw. Folgen von Äußerungen, die vom Handelnden zwar parallel zu HCI/HEI-Aktivität gemacht werden, aber einen solchen Bezug in keiner Weise aufweisen, sind nicht mit *think aloud_activity* zu kodieren. Auch an dieser Stelle muss in weiterführenden Untersuchungen evtl. über die Einführung einer neuen (Fassaden-)Klasse entschieden werden.

   Achtung: Derartige Phänomene, die innerhalb eines bereits laufenden *think aloud_activity* auftreten, sollten in analoger Weise wie Pausen (siehe S. 352) behandelt werden.

5. **Kommentare zu HCI/HEI-Aktivität vs. Kommentare zu Äußerungen:** Bei der Verwendung von *challenge_activity*, *disagree_activity*, *amend_activity*, *stop_activity* und *agree_activity* muss darauf geachtet

werden, ob sich die zu kodierende Aussage wirklich auf eine HCI/HEI-Aktivität oder „nur“ auf eine mit der Aktivität im Zusammenhang stehende Verbalisierung bezieht. Die aufgelisteten Konzepte sollten nur im ersten Fall verwendet werden.

Achtung:

- Zum Teil fallen HCI/HEI-Aktivität und ihre zugehörige Verbalisierung so dicht zusammen, dass nicht entschieden werden kann, ob sich der Partner auf die HCI/HEI-Aktivität selbst oder auf die Äußerung zu dieser bezieht. In diesen Fällen sollten *_*activity*-Kodes vergeben werden. Ein diesbezügliches Beispiel ist in Tabelle 7.54 zu finden.
- Bei HCI/HEI-Aktivität, die nur mit wenigen oder keinen physischen Handlungen einhergehen, z.B. beim Erkunden eines unbekannten, sich bereits in Ansicht befindlichen Artefaktabschnitts (*explore_sth*; siehe Abschnitt 8.1.3), kann es in der Regel nur zu wenig spezifischen Kommentaren von Typ *activity* kommen, z.B. zu Äußerungen vom Typ *stop_activity*.

6. **Ablehnen einer HCI/HEI-Aktivität des Partners und gleichzeitiges Äußern eines Gegenvorschlags:** Äußerungen, in denen ein Entwickler einer HCI/HEI-Aktivität oder Teilen dieser widerspricht und gleichzeitig einen Gegenvorschlag äußert, sind mit *challenge_activity* zu kodieren. Derartige Widersprüche können sich sowohl auf prozedurale (siehe Beispiel in Tabelle 7.54) wie auch auf inhaltliche Aspekte (siehe Beispiel in Tabelle 7.55) beziehen.

Äußerung	Typ	Kontext/Kommentar
<*P1:*> „*New*, *Package*. (.., , , ,) Richtig? (~Ja.)<*P2: „Ja.“*>“	*think aloud_activity*-Äußerung, die mit einer Frage endet. Die zugehörige HCI-Aktivität endet kurz danach ebenfalls.	Sitzung: PR2.1; HCI-Aktivität: Anlegen eines neuen (Unter-)Pakets unter Zuhilfenahme des „*New Java Package*“-Wizard von Eclipse; Details: Der Driver ruft über das Kontextmenu eines im *Package Explorer* von Eclipse gelisteten Pakets *New -> Package* auf. Danach gibt er den Namen des neuen (Unter-)Pakets ein. Bevor er die *Finish*-Schaltfläche betätigt, fragt er: „Richtig?“[a]. Sein Partner stimmt unmittelbar zu und *P1* betätigt die *Finish*-Schaltfläche. Anmerkung: Obwohl *P1* seine Tätigkeiten nur sehr „sparsam“ kommentiert und den eigentlichen Editierschritt nicht mitspricht, wurde die Folge von Äußerung als ein *think aloud_activity* kodiert.
<*P1:*> „So. (, , ,) (Das gibt jetzt ’n (##Featureäähhh##) <*Genau genommen schreibt er nur `Fe`.*>) (.,) ((##`VirtualColumn`(.)`Factory` Punkt##) <*Genau genommen schreibt er nur `VCF`. Diese Zeichenkette wird dann von der IDE zu `VirtualColumnFactory` expandiert.*>“ (...) (##`createVirtualColumns`##) <*Auswahl aus der Autovervollständigung*> Der braucht die `Virtual` (!...!)) Okayy. Und dann (!...!) Hal (!...!) Der braucht dann `All`, ne? Vermutlich. <*P2: „Keine Ahnung was das Ding macht.“*> Ja, (.) O.K.	*think aloud_activity*-Äußerung, in welcher der Driver *P1* eine Frage an seinen Partner richtet. Das *think aloud_activity* ist im Rahmen der BS durch die Frage nicht als beendet zu werten, da die Aktivität danach weiter läuft und kommentiert wird.	Sitzung: PR2.1; HCI-Aktivität: Editieren einer Methode; Details: Die Zeile „`return virtualAttributes;`“ wird zu „`return VirtualColumnFactory.createVirtualColumns(virtualAttributes, getAllColumnAttributes());`“ geändert. Durch den in der Frage verwendeten Begiff `All` referenziert der Driver die (im Argument (potentiell) aufzurufende) Methode `getAllColumnAttributes`. Unmittelbar nach der Frage, noch während er das „Vermutlich“ ausspricht, ruft *P1* die Autovervollständigung auf und erkennt, dass er mit seiner in Frageform verpackten Vermutung (einem *ask_knowledge* (siehe S. 326)) richtig lag und kommentiert dies mit „Ja, (.) O.K.“.

[a] Auf S. 172 wurde erläutert, dass es sich bei dieser Frage um ein *propose_design* handelt. Genau genommen ist die Situation allerdings nicht eindeutig. Es ist nämlich nicht unwahrscheinlich, dass *P1* nur bestätigt haben möchte, dass er den Paketnamen richtig geschrieben hat, es sich also eigentlich um ein *ask_knowledge* handelt. Auf eine solche Interpretation deutet sowohl eine leider in großen Teilen akustisch nicht verständliche Äußerung im Vorfeld hin, wie auch die von *P1* vorgenommene, einem Blick auf den Bildschirm unmittelbar folgende – allerdings fast zu überhörende – Bejahung der eigenen Frage (Anmerkung: Beim Editieren selbst hatte *P1* nicht auf das Display, sondern auf die Tastatur gesehen.)

Tab. 7.53: Beispiele für *think aloud_activity*-Äußerungen, in denen der Sprecher Fragen an seinen Partner richtet. Es muss entsprechend der Regeln auf S. 354 f verfahren werden.

Zeitverlauf:	13:46:49→13:46:51	13:46:51→13:46:52	13:46:53→13:46:57	13:46:59→13:46:59	13:46:59→13:47:01
Äußerungen:	<*P1:*> „Ahh, vielleicht müssen wir den Chat *enablen*.“	<*P2:*> ((~[A]hm), oder (~sowas.))	<*P1:*> „Das wär' 'ne Möglichkeit. Dafür brauchen wir die `auto_prepend`. <*P1* bewegt den Cursor in den *Navigator*-View von Eclipse zur Datei `auto_prepend`.*>“	<*P2:*> „Nee, dass ist doch in der `Conf`.“	<*P1:*> „Ich mach jetzt erstmal 'nen *Working Set*“ <*P1 ruft die Eclipse-Funktionalität *Select Working Set* auf.*>
HCI-Kodierungen (*P1*):	⇝*explore_sth*				*do_sth*⇝
***activity*-Kodierungen (*P1*):**	*think aloud_activity*				*think aloud_activity*⇝
***activity*-Kodierungen (*P2*):**				*challenge_activity*	
Weitere Kodierungen (*P1*):	*propose_hypothesis*		*propose_step*[a]		*propose_step*
Weitere Kodierungen (*P2*):		*agree_hypothesis*		*challenge_step*	
Anmerkungen:			*P1* schlägt vor, die Datei `auto_prepend` zu öffnen. Er geht davon aus, dass der Chat dort aktiviert werden kann.	*P2* widerspricht der gerade begonnenen Handlung von *P1* und wirft ein, dass die Einstellung in der Datei *Conf* vorzunehmen sei.	

[a] Der Vorschlag wird nicht direkt geäußert, sondern ist in einen Erkenntnistransfer „verpackt“. Somit ist eine zusätzliche Kodierung mit *explain_finding* angebracht (siehe hierzu auch S. 205). Analoges gilt für den nachfolgenden Widerspruch von *P2*.

Tab. 7.54: Beispielepisode (PR1.1), in der der Observer *P2* ein *auf den Prozess bezogenes challenge_activity* einwirft. Achtung: 1. Das Beispiel stellt einen Grenzfall dar. Schließlich lässt sich nicht zweifelsfrei ermitteln, ob sich *P2* auf die gerade vom Driver begonnene Aktivität oder auf das von diesem vorher geäußerte *propose_step* bezieht (siehe auch S. 356). 2. Die Konzepte *explore_sth* und *do_sth* werden erst in Abschnitt 8.1.3 bzw. 8.1.8 erläutert. Legende: ⇝: Aktivität hat bereits vorher begonnen bzw. Aktivität läuft noch weiter.

Zeitverlauf:	14:30:39→14:30:41	14:30:43→14:30:44	14:30:44→14:30:47
Äußerungen:	<*P2:*> „`return 0`, sonst aber (~), genau.“	<*P2:*> „Nee,◡dass◡war◡schon◡korrekt.“	<*P1:*> „Ahh,◡du◡willst◡das◡da◡oben◡machen.◡Gut.“
HCI-Kodierungen (*P1*):	⇝*write_sth*⇝		
***activity*-Kodierungen (*P1*):**	<*P1 macht keinerlei Äußerungen.*>		
***activity*-Kodierungen (*P2*):**	*amend_activity* + *agree_activity*	*challenge_activity*	*agree_activity*
Weitere Kodierungen (*P1*):			
Weitere Kodierungen (*P2*):	*propose_design* + *explain_completion*	*propose_design*[a]	*explain_standard of knowledge*
Anmerkungen:	Nachdem *P1* damit begonnen hat, die `return`-Anweisung einer Funktion zu editieren, formuliert *P2* einen Vorschlag bezüglich dessen Gestalt. *P1* kodiert weiter, und *P2* betrachtet seinen Vorschlag direkt nachfolgend als umgesetzt („genau“).	*P1* markiert einen Teil des `return`-Statements. *P2* erkennt, dass *P1* diesen Teil löschen will, und widerspricht, indem er betont, dass das Statement in seiner bestehenden Form korrekt ist.	*P2* erkennt, was *P1* vor hat und stimmt dem nun doch zu.

[a] Der Vorschlag wird nicht direkt geäußert, sondern ist in einen Erkenntnistransfer „verpackt“. Somit ist eine zusätzliche Kodierung mit *explain_finding* angebracht (siehe auch S. 331). Man beachte außerdem die Erörterungen zu Äußerungen, die sich auf implizite Vorschläge beziehen auf S. 161.

Tab. 7.55: Beispielepisode (PR1.1), in welcher der Driver dabei ist, eine Funktion zu editieren und der Observer *P2* ein *auf den neuen Kode bezogenes challenge_activity* einwirft. Achtung: Das Konzept *write_sth* wird erst in Abschnitt 8.1.1 erläutert. Legende: ⇝: Aktivität hat bereits vorher begonnen bzw. Aktivität läuft noch weiter.

7. **Ablehnen einer HCI/HEI-Aktivität, ohne gleichzeitig einen Gegenvorschlag zu äußern:** Ablehnungen von HCI/HEI-Aktivitäten sind nicht immer konstruktiv. Oft werden keine Gegenvorschläge gemacht – zumindest nicht sofort. Solche Widersprüche sind mit *disagree_activity* zu kodieren und fallen in der Regel sehr kurz aus (z.B. „Ȟmm.“, „[A]hm, hmhm.“). Genauso wie Äußerungen vom Typ *challenge_activity* können sie sich sowohl auf prozedurale wie auch auf inhaltliche Aspekte beziehen. Beispielsweise äußert der Observer *P1* in Sitzung PR2.1 – kurz bevor sein Partner einen *SVN-Commit*[216] über den Button *Commit All* in der Eclipse-Oberfläche starten kann – „Nicht *All*“.

8. **Vorauseilende Zustimmungen, Ablehnungen oder Ergänzungen:** Wie eben gesehen können Kommentare zu HCI/HEI-Aktivitäten erfolgen, bevor diese eigentlich durchgeführt worden sind. So erhebt *P1* seinen Einspruch „Nicht *All*“, bevor sein Partner es schafft, den Button *Commit All* zu betätigen. Dies ist ihm hier möglich, weil er die Bewegung der Maus auf dem Bildschirm beobachtet. Obwohl die Aktivität nicht abgeschlossen ist, wird der Einwurf des Partners mit *disagree_activity* kodiert. In weiterführenden Untersuchungen muss evtl. zwischen diesbezüglich unterschiedlichen Ablehnungen unterschieden werden.

9. **Zustimmungen zu HCI/HEI-Aktivitäten:** Auf HCI/HEI-Aktivitäten bezogene affirmative Äußerungen können sehr kurz ausfallen. So äußert der Observer *P2* in Sitzung PR2.1 nur „[A]hmm“, als sein Partner, nachdem dieser unter Verwendung der Autovervollständigung einen Methodenaufruf mit mehreren Parametern hinzugefügt hat, kurz inne hält. Ein weiteres Beispiel für ein kurzes sowie ein Beispiel für ein längeres *agree_activity* sind in Tabelle 7.55 erläutert.

10. **Zustimmungen, Ablehnungen und Ergänzungen zu HCI/HEI-Aktivität nach deren Ende:** Ein Kommentar zu einer HCI/HEI-Aktivität sollte ggf. auch dann als *agree_activity*, *disagree_activity*, *challenge_activity* oder *amend_activity* kodiert werden, wenn die HCI/HEI-Aktivität selbst bereits beendet ist. Allerdings nur dann, wenn der Kommentar derart zeitnah erfolgt, dass nachvollziehbar dargelegt werden kann, dass er sich nicht nur auf das Ergebnis der Handlung, z.B. geschriebenen Kode, bezieht, sondern zumindest in Teilen auch auf das Handeln selbst als unmittelbar oder gerade stattgefundenen Akt. In der Regel wird also ein Einwurf nur dann noch als **_activity* deutbar sein, wenn er höchstens wenige Sekunden nach einer HCI/HEI-Aktivität erfolgt.

11. **Ergänzungen zu einer HCI/HEI-Aktivität:** Macht ein Entwickler ergänzende Anmerkungen zu einer HCI/HEI-Aktivität des Partners, ohne

[216] Siehe Fußnote 66 auf S. 191.

dass dabei die Aktivität oder ein Teil dieser abgelehnt oder als falsch beurteilt wird, muss das Konzept *amend_activity* eingesetzt werden (siehe z.B. Tabelle 7.55) – und das ungeachtet dessen, ob sich die Anmerkung auf das in Entwicklung befindliche Produkt oder den dabei verwendeten Prozess bezieht. Zu derartigen Phänomenen zählen in der Regel auch Situationen, in denen der Driver (sichtlich) zögert und der Observer mit einem Vorschlag, einer Idee oder ähnlichem „einspringt".

12. ***amend_activity* vs. *challenge_activity*:** Nicht immer ist eindeutig entscheidbar, ob sich ein Sprecher im Rahmen einer *activity*-Äußerung rein ergänzend äußert, das heißt, nicht doch gleichzeitig auch negative Kritik an der gerade ablaufenden HCI/HEI-Aktivität mitformuliert bzw. seine Äußerung als derartige Kritik gedeutet werden sollte. Solche Schwierigkeiten treten insbesondere dann auf, wenn ein Driver ohne etwas zu tun verharrt und der Partner ihn mit seiner Äußerung aus dieser (evtl. vermeintlichen) Starre herausholen und möglicherweise sogar (bewusst oder unbewusst) in eine neue HCI/HEI-Aktivität hineinlotsen möchte (siehe Beispiel in Tabelle 7.56). Wie an vielen anderen Stellen auch sind solche Abgrenzungsprobleme weniger als Manko der BS/BKM anzusehen als als Beleg dafür, dass einem Denken, welches „die Welt im Rahmen eindeutig unterschiedener Kategorien" zu begreifen versucht, Skepsis gegenüber angebracht ist ([69, S. 11] nach [140, S. 212]; siehe hierzu auch S. 415). Sie zeigen darüber hinaus, in welcher Weise nötigenfalls in weiterführenden Untersuchungen über Phänomene nachgedacht werden kann.

Zeitverlauf:	12:04:21→12:04:26		12:04:31→12:04:33	12:04:33→12:04:42
Äußerungen:	<*P1:*> „O.K. (, , , ,) Das wär'n (!...!)“		<*P2:*> „`getVirtualAttributes`, dann vielleicht.“	
HCI-Kodierungen (*P1*):	*explore_sth*			*write_sth*
***activity*-Kodierungen (*P1*):**	*think aloud_activity*			
***activity*-Kodierungen (*P2*):**			*amend_activity*	
Weitere Kodierungen (*P1*):	*mumble_sth*			
Weitere Kodierungen (*P2*):			*propose_design*	
Anmerkungen:	Die Äußerung von *P1* bezieht sich wohl auf seine HCI-Aktivität. Mit dem „O.K.“ könnte er zum Ausdruck bringen wollen, dass es nun los geht. Insgesamt ist die Intention der Äußerung allerdings kaum deutbar.		Grenzfall zwischen *amend_activity* und *challenge_activity* (siehe auch S. 362). Will man davon ausgehen, dass *P2* die Tätigkeit von *P1* als Start eines *write_sth* betrachtet, so ist *amend* angebracht, anderenfalls eher *challenge*.	

Tab. 7.56: Beispielepisode (PR2.1), in welcher der Observer *P2* einen Vorschlag (in Hinsicht auf die Änderung des Namens einer Methode (*setVirtualColumns*) eines Interfaces) macht und es unmittelbar nachfolgend zu einem Aktivitätswechsel des Partners kommt. Das Paar hatte sich vorher darüber ausgetauscht, in welcher Weise das Interface geändert werden sollte. Doch anstatt nachfolgend damit zu beginnen, das (bereits geöffnete) Interface diesbezüglich zu editieren, navigiert der Observer durch den *Package Explorer* von Eclipse und wandert dann mit der Maus in den Editor zur Methode *setVirtualColumns* des Interfaces, um dort einen kleinen Moment (ca. eine Sekunde) zu verharren (siehe Spalte 1 und 2). An dieser Stelle macht *P2* seinen Vorschlag (siehe Spalte 3). Der Driver beginnt unmittelbar danach mit dem Editieren des Methodennamens (siehe Spalte 4). Durch den Einwurf des Partners kommt es also zu einem Wechsel der HCI-Aktivität, wobei allerdings nicht ausgeschlossen werden kann, dass es zu diesem Zeitpunkt auch ohne die Äußerung von *P2* zum Wechsel gekommen wäre.

13. **Den Abbruch einer Aktivität vorschlagen:** Es kommt vor, dass ein Entwickler sich explizit dafür ausspricht, eine aktuell laufende HCI/HEI-Aktivität abzubrechen bzw. dieser nicht weiter nachzugehen. Solche Äußerungen sind mit *stop_activity* zu kodieren. Im Gegensatz zu *challenge_activity* oder *disagree_activity* wird *stop_activity* nur eingesetzt, wenn der Sprecher Bezug auf die gesamte HCI/HEI-Aktivität und nicht nur auf Teile dieser nimmt. Beispielsweise wirft in Sitzung PR2.1 der Observer *P1*, während der Driver gerade dabei ist, eine Klasse unter Verwendung von JDemo[217] zu testen, ein:

    > „O.K. Aber das, was wir gemacht haben, hat eher Auswirkungen auf die `Action` (.) als auf (.) die GUI. So (.), das heißt, ich würd's gern aus <**Applikationsname**> (!...!).“

    Er schlägt hiermit vor, den JDemo-Test abzubrechen und die Änderungen stattdessen im Rahmen eines Aufrufs der gesamten Applikation zu überprüfen. Es handelt sich um ein *stop_activity*, welches aus einem *explain_knowledge* und einem *propose_step* besteht.

14. **Einwürfe des Partners, die zu Aktivitätsänderungen führen:** Wie in Tabelle 7.56 erläutert, können Anmerkungen zu HCI/HEI-Aktivitäten, selbst wenn es sich nicht um solche vom Typ *stop_activity* handelt, dazu führen, dass der Partner einen Aktivitätswechsel vornimmt (auch wenn es im Rahmen des in der Tabelle erläuterten Beispiels möglich ist, dass der Partner den Wechsel ohnehin vollzogen hätte).[218] Derartige Phänomene sind – zumindest potentiell – nicht nur im Zusammenhang mit einem *amend_activity*, sondern auch im Kontext anderer *activity*-Äußerungen möglich.

15. ***think aloud_activity* eines Observers:** Das Konzept *think aloud_activity* adressiert in erster Linie Driver-Äußerungen. Es sind aber auch Umstände vorstellbar, in denen Äußerungen eines Observers mit diesem Konzept annotiert werden sollten. Dies betrifft Situationen, in denen sich die aktuelle Driver-Observer-Rollenverteilung auflöst oder kein geeignetes Modell mehr darstellt, z.B. dann, wenn ein Entwickler, der gerade noch die Observer-Rolle inne hatte, beginnt, Skizzen oder ähnliches anzufertigen, und diese Tätigkeiten verbalisiert.

[217] Java demonstration framework: Ein Hilfsmittel für die Durchführung testgetriebener Entwicklung. Es werden Demoklassen geschrieben, mit denen einzelne Kodeteile vorgeführt werden können (http://www.jdemo.de/ (Abruf: 16.09.2011)).

[218] Obwohl die Äußerung von *P2* in Tabelle 7.56 zumindest potentiell den Grund für einen Aktivitätswechsel darstellt, wird sie im Rahmen der BS nicht als *stop_activity* kodiert. Schließlich expliziert *P2* in keiner Weise den Wunsch danach, dass sein Partner die aktuelle Aktivität abbricht. Weiterführende Untersuchungen müssen entscheiden, wie sie mit impliziten Stopps umgehen wollen.

16. ***think aloud_activity* als Fassade höherer Ordnung:** Es kommt vor, dass Entwickler, in der Regel die Driver, im Rahmen von parallel zu ihren HCI/HEI-Aktivitäten geäußerten Erläuterungen auch direkt auf ihr eigenes aktuelles Handeln bezogene Kritik formulieren. Genau genommen müssten derartige Äußerungsteile zusätzlich mit *disagree_activity*, *challenge_activity*, *amend_activity*, *agree_activity* oder *stop_activity* kodiert werden. Im Rahmen der Herleitung der BS/BKM wurde allerdings nicht derart verfahren. Weiterführende Untersuchungen müssen ein solches Vorgehen aber in Erwägung ziehen.

Weitere Anmerkungen zu *activity*-Konzepten

Die von dem Konzept *think aloud_activity* adressierten Phänomene unterscheiden sich in einem Punkt grundlegend von denen, die in weiten Teilen der (psychologischen) Forschung im Zusammenhang mit dem Terminus *think aloud* (siehe z.B. [58]) adressiert werden. Bei den letzteren handelt es sich nämlich um solche, die mit einem lauten Denken einhergehen, zu dem eine Person in der Regel explizit (und zum Teil fortwährend), meist zu Beobachtungszwecken aufgefordert wird und welches sie von sich aus nicht (unbedingt) praktiziert hätte. Derartige Aufforderungen – in diesem Fall durch den Partner – konnten im Rahmen der PP bisher nicht beobachtet werden.[219]

7.8 Sonstige Konzepte

Zur Menge der HHI-Konzepte gehören auch *mumble_sth* und *say_off topic*. Sie haben allerdings einen eher deskriptiv markierenden als einen konzeptualisierenden Charakter. So dient das Konzept *mumble_sth* dazu, Äußerungen zu kennzeichnen, deren Deutung nicht gelingt, sei es, weil sie rein akustisch nicht zu verstehen sind oder weil sie ein auch im Kontext nicht interpretierbares Satzfragment darstellen – z.B. „Dann is'‿es‿so, dass wir die momentan (.) dann (.) (∼∼)“ (PR2.1). Mit *say_off topic* werden Äußerungen kodiert, in denen sich der Sprecher mit Themen beschäftigt, die inhaltlich keinen Bezug zur gerade stattfindenden Entwicklungstätigkeit aufweisen. Z.B. macht P1 in Sitzung PR2.1 eine Bemerkung, die sich nur darauf bezieht, dass das Paar mit einer Kamera beobachtet wird: „Was die sich wohl denken, die nicht wissen, was wir hier tun“[220].

[219] An dieser Stelle sei daran erinnert, dass den Probanden in den hier analysierten Sitzungen keinerlei Vorgaben bez. des Prozesses gemacht worden sind, insbesondere keine Auflagen in Hinsicht auf vorzunehmende Verbalisierungen.

[220] Insgesamt gab es in Sitzung PR2.1 nur sehr wenige Äußerungen, die sich auf die Beobachtungssituation bezogen. Dem ersten Anschein nach spielte das „Beobachtetwerden“ – genauso wie in den anderen hier diskutierten Sitzungen – so gut wie keine Rolle. Allerdings wurden in den Sitzungen insgesamt nur sehr wenige (jeweils weit weniger als zehn) Äußerungen vom Typ *say_off topic* gemacht. Dies kann ein Hinweis darauf sein, dass die Beobachtungssituation in mancher Hinsicht doch einen Einfluss hatte. Evtl. sind die wenigen Abschweifungen aber auch auf den von Williams et al. [216; 217; 218] erörterten Effekt des *pair pressure*s zurückzuführen.

Im Rahmen der BS wird hierbei auf eine Differenzierung zwischen initiativen und reaktiven Äußerungen verzichtet.

7.9 Shifts und shift-begründete Kodierungen

Die BS ist in weiten Teilen dahingehend konzipiert worden, sprachliche Bezüge sichtbar machen zu können – einerseits durch die Einführung bzw. Verwendung von reaktiven Verben, andererseits durch die Festlegung von Kodierregeln, durch die die ermittelten charakterisierenden Eigenschaften von initiativen Äußerungen, insbesondere solchen, die zur Festlegung des Objektes dienen, auch bei der Kodierung von verbalen Reaktionen berücksichtigt werden. Eine solche Regel wurde z.B. für die Kodierung von Zustimmungen zu Äußerungen vom Typ *explain_knowledge* festgelegt. Denn obwohl Äußerungen vom Typ *explain_knowledge* selbst derart formuliert sein müssen, dass sie keinen Vermutungscharakter aufweisen, wird eine affirmative Antwort auch dann mit *agree_knowledge* annotiert, wenn sich ein Sprecher „letzte Zweifel“ an der Richtigkeit vorbehält, also seine Zustimmung in gewisser Weise den Charakter einer Vermutung hat (siehe S. 326).[221] Weitere derartige Regeln sind beispielsweise auf S. 269 (Ergänzungen zu Erkenntnisäußerungen) oder S. 275 (Widersprüche zu Erkenntnisäußerungen) zu finden.

Trotzdem ist es nicht immer angebracht, dem zentralen Prinzip des Sichtbarmachens von sprachlichen Zusammenhängen (siehe S. 161) uneingeschränkt folgen zu wollen – zumindest nicht ohne ergänzende Annotationen oder Hilfskodes zu verwenden. Bestimmte Phänomene können es vielmehr notwendig machen, bei der Kodierung von Erwiderungen einen Klassenwechsel vorzunehmen, z.B. dann, wenn sich eine Antwort nicht auf die primäre Aussage der ursprünglichen Äußerung, sondern auf einen Subaspekt dieser bezieht, der Sprecher also eine *thematische Verschiebung*, im Weiteren kurz *Shift* genannt, vollzieht. In Tabelle 7.57 ist ein diesbezügliches Beispiel zu finden: Hier adressiert der Sprecher *P2* mit seiner Erwiderung weder den vom Partner geäußerten Designvorschlag noch die mit diesem zusammen geäußerte Hypothese, sondern „nur“ die beiden Äußerungsteilen zugrunde liegende Erkenntnis. Da derartige Erkenntnisse im Rahmen der BS standardmäßig nicht explizit kodiert werden (siehe z.B. Abbildung 7.1) stellt sich die Frage, wie mit der Äußerung von *P2* zu verfahren ist. Die BS schlägt hier zwei Verfahrensweisen vor:

1. **Shift-begründete Kodierungen**: Äußerungen, auf die ein Shift folgt, werden mit *zusätzlichen* Kodierungen versehen, und zwar mit solchen, die es ermöglichen, die nachfolgenden Erwiderungen derart BS-konform annotieren zu können, dass der Zusammenhang zwischen Äußerung und Erwiderung erkennbar wird. Solche Kodierungen werden im Rahmen der BS als

[221] Die BS versucht allerdings nicht festzulegen, was noch als „mit letzten Zweifeln behaftet“ verstanden werden soll, bzw. was schon als Ablehnung zu werten ist.

shift-begründet bezeichnet. Ein Beispiel hierfür ist in Tabelle 7.57 zu finden. Hier wird die Äußerung von *P1* nur deshalb zusätzlich mit *explain_finding* annotiert, um den Zusammenhang mit der nachfolgenden Erwiderung von *P2* durch die Annotation eines *agree_finding* sichtbar machen zu können.

2. **Verwendung eines deskriptiven Hilfskodes**: Shifts werden explizit durch die Verwendung eines *shift*-Kodes oder einer Konzepteigenschaft *shift* markiert.

An dieser Stelle soll noch einmal daran erinnert werden: Shift-begründete Kodierungen und Hilfskodes wie auch Kodierung im Allgemeinen sollten nicht aus reinem Selbstzweck verwendet werden. Vielmehr soll das Nachdenken über unterschiedliche Kodierungsmöglichkeiten dabei helfen, einzelne Episode besser zu verstehen, um so zu „höheren“ Konzepten zu kommen. In diesem Zusammenhang kann es z.B. sinnvoll sein, zwischen verschiedenen Arten von Shifts zu unterscheiden, z.B. zwischen diskreten und stetigen, wobei Shifts dann als diskret bezeichnet werden könnten, wenn der Fokus mehr oder weniger abrupt gewechselt wird (z.B. von *design* nach *finding*) und als stetig, wenn der Wechsel eher fein abgestuft oder schleichend erfolgt (z.B. von *knowledge* hin zu *hypothesis*).

#	HHI-Kode	Äußerung	Kontext	Hinweise zur Analyse
1	*P1.amend_design* + *P1.propose_hypothesis*/ *P1.explain_finding*	„Nur warum schmeißen wir nicht dann auch noch ähm (.) den für diese `id` mit raus? (.) Oder machen sie das wieder explizit irgendwo?“	Das Paar spricht über Modifikationen an einer bereits existierenden `for`-Schleife. In dieser Schleife werden Elemente aus einem Cache-Objekt gelöscht. Im ersten Teil seiner Äußerung ergänzt *P1* einen zuvor vom Partner geäußerten Vorschlag um den Hinweis, dass evtl. ein weiteres Element (auf das er mit einem Stift zeigt) gelöscht werden muss. Seiner in Frageform formulierten Vorschlagsergänzung fügt er eine Hypothese hinzu: Er hält es für möglich, dass die von ihm adressierte Operation bereits an anderer Stelle des Kodes durchgeführt wird. Das in der Formulierung verwendete Personalpronomen „sie“ bezieht sich auf Entwickler, die zuvor am Kode gearbeitet haben.	**Zum ersten Satz:** 1. Aus dem Kontext kann geschlossen werden, dass es sich um eine rhetorische bzw. keine direkt an den Partner gerichtete Frage handelt. 2. Somit ist es ein *amend_design* vom Typ SO (siehe Tabelle 7.5). 3. Hinweis: Da im Rahmen einer Kodierung mit der BKM produktorientierte Konzepte den Vorrang vor *UK* haben, wird nicht *propose_hypothesis* kodiert. **Zum zweiten Satz:** 1. Auch hier handelt es sich um eine rhetorische Frage. 2. Es ist ein *propose_hypothesis*. Der Sprecher stellt eine Vermutung über eine Eigenschaft des vorhandenen Kodes auf. 3. Hinweis: Eine Kodierung mit *disagree_design* ist nicht angebracht, da *P1* beide Äußerungen ohne wirkliche Pause hintereinander macht und in der zweiten kein expliziter Widerruf erfolgt. In gewisser Weise begründet *P1* mit der zweiten Äußerung in Hypothesenform, warum er seinen Vorschlag nicht entschiedener formuliert hat. **Weiteres:** Beiden Äußerungen liegt die Erkenntnis zugrunde, dass das besagte Element grundsätzlich – das heißt entweder im aktuell betrachteten Kodefragment oder an einer anderen Stelle – gelöscht werden sollte. Da sich *P2* im Rahmen seiner Reaktion auf die Äußerung genau auf diese Erkenntnis bezieht, sollte die Äußerung zusätzlich auch mit *explain_finding* annotiert werden.
2	*P2.agree_finding*	„Das ist ein guter Punkt. Das fehlt hier.“	Nach der Äußerung von *P1* schweigen beide Entwickler ca. 10 Sekunden mehr oder weniger regungslos. Dann äußert sich *P2* in zustimmender Weise.	*P2* stimmt der der Äußerung des Partners zugrunde liegenden Erkenntnis zu. Hierbei bestätigt er, dass das Entfernen des besagten Elementes im betrachteten Kodefragment nicht durchgeführt wird. Achtung: *P2* stimmt weder zu, die besagte Kodestelle zu modifizieren, noch äußert er sich explizit dazu, ob das Löschen an einer anderen Stelle im Kode bereits implementiert ist.

Tab. 7.57: Beispielepisode (PR1.1: 14:57:00–14:57:19) für eine shift-begründete Kodierung. Ein Entwickler geht auf einen Vorschlag mit nachfolgender Hypothese nur insoweit ein, dass er betont, dass die den beiden Äußerungen zugrunde liegende Erkenntnis richtig ist. Die Episode bildet ein Beispiel für shift-begründete Kodierungen, da ein *explain_finding* nur deshalb zu annotieren ist, weil in einer nachfolgenden Äußerung explizit auf den Erkenntnisaspekt Bezug genommen wird.

Kapitel 8

Die HCI/HEI-Konzepte sowie ergänzende Konzepte

Die Menschen wollen immer noch Definitionen, aber
es ist jetzt ganz klar, daß nichts
definiert werden kann.
John Cage, Tagebuch: Wie die Welt zu verbessern ist
(Du machst alles nur noch schlimmer) 1965

Im vorliegenden Kapitel werden zuerst die HCI/HEI-Konzepte näher erläutert (Abschnitt 8.1). Nachfolgend wird auf eine Reihe ergänzender, eher deskriptiver Kodes eingegangen (Abschnitt 8.2) .

Eine erste Fassung der HCI/HEI- und der ergänzenden Konzepte wurde (ebenso wie eine initiale Version der HHI-Konzepte) 2008 auf einem Workshop der *Psychology of Programming Interest Group (PPIG)* vorgestellt [182].

8.1 Die HCI/HEI-Konzepte

Bei den HCI/HEI-Konzepten handelt es sich, wie bereits in Abschnitt 6.5 beschrieben, um solche, mit denen Interaktionen einer Person mit ihrer Umgebung, das heißt sowohl mit dem Rechner (z.B. mittels Tastatur und Maus; HCI-Konzepte (*human-computer interaction*)) wie auch mit der sonstigen Umgebung (HEI-Konzepte (*human-environment interaction*)) – ausgenommen dem Partner – beschrieben werden können. Die HCI/HEI-Konzepte adressieren dabei Phänomene bzw. genau genommen Teile von Phänomenen, die im weiteren Verlauf *Handlungsentitäten* genannt werden. Eine Handlungsentität umfasst eine Folge von (atomaren) Tätigkeiten und zwar eine solche, die durch ein kurzfristiges Handlungsziel bestimmt ist[1]. Das Handlungsziel kann im Vorfeld, in der Regel in Form einer *step*- oder *design*-Äußerung, expliziert worden sein. In der Regel besteht eine Handlungsentität nicht nur aus einer einzelnen atomaren Tätigkeit (siehe S. 189), wie z.B. dem Eingeben eines Zeichens, dem Schreiben eines Wortes oder dem Anklicken eines Elementes der graphischen Benutzeroberfläche. Beispielsweise bilden alle der Äußerung

> „Wir können einfach, äh, einmal das‿Ding in 'ne, 'ne normale `id` umwandeln, also `code_to_id` und dann wieder `id_to_code`. (~~)“
> (für Details siehe Tabelle 7.14)

[1] Der Begriff „kurzfristig“ wird hier verwendet, um deutlich zu machen, dass es nicht um eines der auf S. 229 diskutierten übergeordneten Ziele geht, die im Rahmen einer Strategie anvisiert werden.

unmittelbar nachfolgenden Schreibaktivitäten einer Person, die der Umsetzung des Vorschlags dienen, eine Handlungsentität. Eine derartige Explizierung des Handlungszieles erfolgt aber bei weitem nicht immer.

Die hier vorgenommene „Definition“ des Begriffs Handlungsentität weist zumindest in Hinsicht auf ihre Operationalisierbarkeit im Rahmen der Verwendung der BS/BKM Schwachstellen auf:

1. Bei nicht explizierten Zielen kann es für Außenstehende schwierig werden zu entscheiden, an welcher Stelle davon gesprochen werden sollte, dass eine Handlungsentität endet und/oder eine neue beginnt. Vielmehr muss sogar davon ausgegangen werden, dass der Handelnde zeitweise nicht in Form von festen Handlungsentitäten agiert.

2. Das Ziel einer Handlung kann sich im Laufe der Handlung ändern. So können z.B. neue Teilziele hinzukommen oder Teilziele modifiziert werden, zum Teil auch ohne dass dies expliziert wird.

3. Die Umsetzung eine Zieles muss nicht kontinuierlich erfolgen. So kann der Driver aus unterschiedlichen Gründen Pausen einlegen. Es stellt sich also die Frage, in welcher Weise Pausen als Indizien für das Ende einer Handlungsentität zu werten sind.

Im Rahmen der Herleitung der BS/BKM wurde nicht versucht, diese Probleme vollständig zu lösen. Der Hauptgrund für diese Entscheidung ist der folgende: Die HCI/HEI-Konzepte – mit denen im Rahmen der BS/BKM Teile von Handlungsentitäten adressiert werden – bilden nur eine grobe Klassifizierung beobachtbarer HCI/HEI-Phänomene. Unter dem Konzept *verify_sth* werden beispielsweise grundlegend verschiedene Formen von Überprüfungen zusammengefasst. Somit kann es im Rahmen der Verwendung der HCI/HEI-Konzepte erst einmal nicht darum gehen, „die einzig richtige“ Kodierung einzelner Phänomene zu erhalten, sondern es sollte vielmehr das Ziel sein, sich unter Zuhilfenahme der HCI/HEI-Konzepte einer spezifischeren Charakterisierung derartiger Phänomene und somit der Beantwortung der aktuellen qualitativen Fragestellungen annähern zu können. Hierbei muss eine sich an konkreten Situationen und Forschungsausrichtungen orientierende „Kodierweise“ herausgearbeitet werden.

Nichtsdestotrotz gibt die BS aber eine Reihe von Hilfestellungen zur Kodierung von Teilen von Handlungsentitäten. So besteht ein Grundprinzip der BS darin, dass unterschiedliche Handlungsformen, auch wenn sie ein und demselben Handlungsziel dienen und dementsprechend zusammen eine Handlungsentität bilden, einzeln mit unterschiedlichen HCI/HEI-Konzepten zu annotieren sind. Im Weiteren wird hier von *Teilhandlungen* gesprochen. Ist es beispielsweise zum Erreichen eines Handlungszieles erforderlich, nach einer Datei mit einer bestimmten Eigenschaft zu suchen und sie danach in einer bestimmten Weise zu editieren, kommen, wenn vom Handelnden auch derart verfahren wird, zumindest zwei Konzepte, nämlich *search_sth* (siehe Abschnitt 8.1.2) und *write_sth* (siehe Ab-

schnitt 8.1.1) zum Einsatz. Dabei werden von den HCI/HEI-Konzepten in der Regel nicht nur rein physische Tätigkeiten, also körperliche Aktivitäten adressiert, sondern komplexe Handlungen, mit denen physische Tätigkeiten einhergehen können oder die sich um physische Tätigkeiten „ranken“. Beispielsweise adressiert das Konzept *write_sth* nicht das Tippen einer Person im engeren Sinne sondern die Schreibhandlung, die das Tippen beinhaltet. Im Rahmen der Erläuterungen zu Konzept *write_sth* (Abschnitt 8.1.1) wird hierauf genauer eingegangen.

Um den Einstieg in die HCI/HEI-Kodierung zu erleichtern, werden im Rahmen der nachfolgenden Erläuterungen der einzelnen Konzepte weitere Regeln vorgestellt, z.B. solche, die sich auf den Umgang mit Handlungspausen beziehen (siehe Unterabschnitt 8.1.1).

Achtung:

- Die BS/BKM stellt keine Mittel zur Verfügung, um Handlungsentiäten an und für sich zu kodieren, sondern nur solche, mit denen Teilhandlungen von Handlungsentitäten adressiert werden können. Wenn eine Handlungsentität allerdings aus nur einer fortlaufend gleichartigen Handlung besteht, entspricht die Kodierung dieser Handlung einer Kodierung der Handlungsentität.

- Es ist wichtig im Auge zu behalten, dass im Rahmen der Herleitung der BS/BKM die Menge der HCI/HEI-Konzepte weit weniger detailliert ausgearbeitet wurde als die Klasse HHI. So unterscheiden die HCI/HEI-Konzepte – wie bereits angesprochen – nur grob zwischen verschiedenen Aktivitätstypen (*do*, *search*, *explore*, *write*, *verify*, *read*, *show*, *sketch* – für eine Kurzübersicht siehe Tabelle 8.1) und gar nicht zwischen verschiedenen Objekten, auf die sich einzelne Aktivitäten beziehen können (es gibt nur das unspezifische Objekt *something*, kurz *sth*). Differenzierungen werden weiterführenden Untersuchungen überlassen.

- Ein und dieselbe physische Tätigkeit muss zu unterschiedlichen Zeitpunkten nicht zur Annotation desselben HCI/HEI-Konzeptes führen. Dies liegt daran, dass im Rahmen einer BS-Kodierung, wie bereits erläutert, vor allem das Handlungsziel von Bedeutung ist. So kann der Aufruf (Eintippen + Return) eines über eine Datei konfigurierbaren Kommandos in einer Konsole in einer Situation der Überprüfung der vorher editierten Konfigurationsparameter dienen und in einer anderen den Zweck haben, die Funktionalität des Kommandos zu nutzen. Im ersten Fall sieht die BS/BKM eine Kodierung mit dem Konzept *verify_sth* vor (siehe Abschnitt 8.1.4), im zweiten eine Kodierung mit dem Konzept *do_sth* (siehe Abschnitt 8.1.8).

- In vielen Fällen können HCI/HEI-Konzepte, zumindest wenn es nur um die Handlungen einer Person geht, disjunkt verwendet werden. Dies bedeutet, dass sich Intervalle, in denen Kodierungen von Handlungen ein und derselben Person mit verschiedenen HCI/HEI-Konzepten vorgenommen werden,

Verb	Beschreibung
do	Wird verwendet, wenn alle anderen HCI/HEI-Verben der BS/BKM nicht eingesetzt werden können.
explore	Erkunden von Artefakten oder Gruppen von Artefakten bzw. Dokumenten und Gruppen von Dokumenten.
read	Vorlesen von Bildschirminhalten oder Inhalten von physisch vorliegenden Dokumenten/Artefakten.
search	Suchen nach wohldefinierten Objekten.
show	Zeigen auf bestimmte Teile von physischen oder nicht physischen Artefakten.
sketch	Anfertigen von Skizzen auf Papier oder unter Zuhilfenahme von Software.
verify	Überprüfen von vorgenommenen Modifikationen an Artefakten und sonstigen Dokumenten (inkl. Konfigurationen).
write	Editieren von direkt oder indirekt zum Produkt gehörenden Artefakten/Dokumenten.

Tab. 8.1: Grobe Beschreibung der Verbbestandteile der in den Daten identifizierten HCI/HEI-Konzepte. Jedes dieser Verben wird im Rahmen der BS nur im Zusammenhang mit dem Objekt *sth* verwendet, kommt also nur in einem Element der BKM vor.

nicht überschneiden. Dies gilt aber nicht immer. So kann es vorkommen, dass eine Person während sie dabei ist, einem bestimmten Handlungsteilziel nachzugehen, ein zweites anvisiert und durch Handlungen zu erreichen versucht. Ein diesbezügliches Beispiel wird im Zusammenhang mit den Erläuterungen zum Konzept *verify_sth* in Tabelle 8.6 diskutiert.

8.1.1 Das Konzept *write_sth*

Das HCI-Konzept *write_sth* adressiert *Schreibhandlungen* und zwar solche, in denen direkt oder indirekt zum Produkt gehörende Artefakte/Dokumente editiert werden (siehe auch Exkurs 12). Es ist also sowohl bei Kodeänderungen wie auch bei Modifikationen von Konfigurationsdateien oder Dokumentationen einzusetzen.[2] Hierbei ist es egal, ob die Änderungen händisch oder unter Zuhilfenahme von Wizards durchgeführt werden. Eine *Refactoring*-Operation wie z.B. ein *Move* in der IDE Eclipse wäre also als *write_sth* zu kodieren. Außerdem wurde festgelegt, dass sich ein Phänomen vom Typ *write_sth* immer auf *eine* durch ihren Inhalt bzw. ihr Ziel bestimmte Handlungsentität, z.B. auf einen beschlossenen *design*-Vorschlag, der gerade umgesetzt wird, zu beziehen hat.[3] Das erste *write_sth*

[2] Im Rahmen der Herleitung der BS/BKM wurden auch solche Änderungen an Skripten mit *write_sth* annotiert, die nur dazu dienten, bestimmte Tätigkeiten im Rahmen der Paarprogrammierungssitzung zu automatisieren oder strukturieren, z.B. ein Übertragen von Dateien auf einen Server.

[3] Dies bedeutet nicht, dass das Vorhaben verbalisiert worden sein muss.

Exkurs 12: *Schreibhandlung (write_sth) vs. Tippen (Texteingabe)*
Im Rahmen der BS wird bei Modifikationen von direkt oder indirekt zum Produkt gehörenden Artefakten/Dokumenten[a] zwischen einer *Schreibhandlung* (einem *write_sth*) und dem Eingeben von Symbolen (*Tippen*) unterschieden. Eine Schreibhandlung findet im Rahmen einer bestimmten Handlungsentität statt und ist auf das Erreichen des Zieles (bzw. eines Teilzieles) der Handlungsentität gerichtet. Sie muss zwar Symboleingaben durch Tastatur und/oder Maus (Tippen) beinhalten, adressiert diese allerdings im Kontext der mit diesen Eingaben zusammenhängenden kognitiven Vorgängen des Schreibers. Somit umfasst eine Schreibhandlung beispielsweise auch Pausen in der physischen Aktivität, z.B. solche, die der Schreiber allem Anschein nach zum Nachdenken verwendet. Auch Unterbrechungen, die nicht zu einer losgelösten, eigenständigen Handlung des Schreibenden (oder einem Abbruch aller Aktivitäten) führen, sondern sozusagen en passant behandelt werden, werden im Rahmen der BS als zur Schreibhandlung gehörend kodiert. Eine *write_sth*-Kodierung erstreckt sich also oft über mehr als eine zusammenhängende, nur auf die Eingabe von Symbolen bezogene Tastatur- bzw. Mausbedienung. Für Symboleingaben selbst gilt, dass sie im Rahmen der BS ausschließlich über die Schreibhandlung erfasst werden, zu der sie gehören.
Im vorliegenden Dokument wird noch ein dritter Begriff, nämlich *Editierschritt* verwendet. Dieser ist weitgehend synonym zu dem des Tippens. Im Gegensatz zu dem Begriff Tippen wird er allerdings verwendet, wenn betont werden soll, dass die zugehörigen Schreibhandlungen, innerhalb derer die so adressierten Tätigkeiten stattfinden, nicht notwendigerweise mit einer Symboleingabe beginnen und/oder enden. So kann es im Rahmen des Kodierens notwendig werden, ein *write_sth* vor der Eingabe des ersten Zeichens starten zu lassen, z.B. wenn der Schreibende offensichtlich (kurz) innehält, um zu überlegen, wie er beginnen möchte.

[a] Direkt zum Produkt gehören z.B. Klassendateien, indirekt *Build*-Skripte.

einer Handlungsentität beginnt mit dem ersten Editierschritt, der innerhalb dieser stattfindet, und endet direkt nach dem letzten Editierschritt, der

i. im Rahmen des Versuchs, das Handlungsziel zu erreichen, durchgeführt wird,

ii. vor einer Unterbrechung stattfindet,

iii. vor einem Driverwechsel vollzogen wird,

iv. vor einer vollständigen Änderung des Handlungsziels erfolgt oder

v. vor dem Zeitpunkt liegt, an dem für die handelnde Person eine Kodierung mit einem anderen HCI/HEI-Konzept notwendig wird.

Weitere *write_sth*-Kodierungen (zur selben Handlungsentität) beginnen dann, wenn vom Handelnden ein Editierschritt durchgeführt wird, aber noch kein *write_sth* „läuft“. Für sie gelten die gleichen Endkriterien.

Folgendes muss zu den einzelnen Endkriterien angemerkt werden:

Zu i. Im Idealfall handelt es sich hierbei um den letzten Editierschritt, der zur Erreichung des Handlungszieles nötig war. Dies muss aber nicht sein. So werden *step*s auch abgebrochen, sei es explizit durch Verbalisierung eines Entschlusses oder implizit, indem einfach nicht weiter an der Umsetzung des *step*s gearbeitet wird. In diesen Fällen endet das *write_sth* direkt nach dem letzten Editierschritt vor dem Abbruch.

Zu ii. Eine Unterbrechung kann einerseits durch ein von außen kommendes Ereignis entstehen, z.B. durch eine Äußerung des Partners, für die der Driver seine Tätigkeit (vorübergehend) aussetzt, und andererseits vom Schreibenden selbst initiiert werden, z.B. indem dieser eine Pause einlegt, um dem Partner zu erklären, was er gerade tut, oder um kurz über etwas nachdenken zu können. Wie in Exkurs 12 angesprochen sollten Unterbrechungen (in der Regel solche von kurzer Dauer), die nicht Folge einer als eigenständig anzusehenden Aktion sind, nicht als Abbruch eines *write_sth* gewertet werden. Hier sind insbesondere solche gemeint, die allem Anschein nach durch kurze Denkpausen entstehen. Aber auch Pausen, die daher rühren, dass Fragen des Partners mehr oder weniger en passant beantwortet werden, fungieren im Rahmen der BS nicht als Abbruchkriterium. Die hiermit beschriebene Art der Sichtweise auf Phänomene wird im Weiteren *pragmatisch* genannt. Zur Veranschaulichung betrachte man das erste Beispiel in Tabelle 7.52. Hier hält *P1*, nachdem er das Wort „`projects`“ mitsprechend eingegeben hat, für einen kurzen Moment von ca. einer Sekunde mit dem Editieren inne. Dabei äußert er den Laut „ähm“. An dieser Stelle erscheint es angemessen, trotzdem von einem fortlaufenden *write_sth* zu sprechen. Es ist ohne weiteres vorstellbar, dass solche das Schreiben unterbrechende Denkpausen auch mehr Zeit in Anspruch nehmen können, ohne dass es sinnvoll erscheint davon zu sprechen, dass ein *write_sth* endet und kurze Zeit später ein neues beginnt.[4]

Achtung: Nicht bei jeder Analyse ist es sinnvoll, die hier erläuterte pragmatische Sichtweise einzunehmen. Als Alternativen bieten sich Sichtweisen an, mit denen versucht wird, nur „stark zusammenhängende“ Tastatur- und Mausbenutzungen zusammen mit einem *write_sth* zu kodieren. Hierbei muss im Einzelnen festgelegt werden, anhand welcher Kriterien ein starker Zusammenhang festgestellt werden soll – z.B. durch die Definition der maximalen Länge von Tipppausen, die innerhalb eines mit *write_sth* zu

[4] Bei einer Kodierung im Rahmen der BS können Tipppausen auch in einer systematischeren Weise behandelt werden. Hier sollte analog dem in Fußnote 215 (auf S. 353) für Annotationen mit dem Konzept *think aloud_activity* erläuterten Verfahren vorgegangen werden. Es sollten also zuerst alle zusammenhängenden Tippoperationen mit einem Konzept wie z.B. *part of (write_sth)* oder *type_sth* annotiert werden. Danach sollte untersucht werden, welche der so notierten Teile zu einem *write_sth* zusammenzufassen sind. An dieser Stelle muss aber betont werden, dass die Kosten eines solches Vorgehens in vielen Fällen ihren Nutzen bei weitem übersteigen werden.

kodierenden Phänomens auftreten dürfen. Derartige Sichtweisen werden im Weiteren *orthodox* genannt (siehe z.B. Abbildung 8.1)

Zu iii. Mit einem Driverwechsel, also der Übernahme von Maus und Tastatur durch den Partner, wird ein evtl. noch laufender Editiervorgang (also ein *write_sth*) zwangsläufig abgebrochen. An dieser Stelle sei davon abgesehen, dass es vorkommen kann, dass nur ein Eingabegerät, also z.B. nur die Maus, übernommen wird. In solchen Fällen muss individuell entschieden werden, ob ein Abbruch des Editiervorganges vorliegt.

Zu iv. Eine Änderung des Handlungsziels kann auf zwei Ebenen stattfinden:

(a) Das ursprüngliche Ziel wird im Grundsatz beibehalten. Es wird lediglich eine Justierung, z.B. in Form einer Ergänzung, vorgenommen

(b) Es wird ein komplett neues Ziel anvisiert.

Im zweiten Fall beginnt eine neue Handlungsentität. Somit endet auch ein bis zu diesem Zeitpunkt laufendes *write_sth*. Falls der Driver mit dem Editieren fortfährt (z.B. eine gänzlich andere Funktionalität an einer anderen Stelle des Kodes bearbeitet), beginnt ein neues *write_sth*. Im ersten Fall sollte der Kodierer eine pragmatische Sicht einnehmen: Ein zum Zeitpunkt des Wechsels stattfindendes *write_sth* sollte als fortlaufend angesehen werden, wenn die einzelnen sich aus den Änderungen der Handlungsziele ergebenden Schritte im Sinne des Konzeptes *amend_activity* direkt auf diesem aufbauen und ihre etwaigen Verbalisierungen (in Vorschlagsform) nicht als Zäsur im Schreibprozess wirken (siehe auch Beispiel in Tabelle 8.2).[5]

Zu v. Geht ein Driver im Rahmen einer Handlungsentität vom Editieren von Artefakten/Dokumenten zu einer anderen eigenständigen Tätigkeit über, z.B. indem er eine Suchfunktionalität der IDE aufruft und startet, endet ein laufendes *write_sth* und eine HCI/HEI-Aktivität eines anderen Typs, hier ein *search_sth*, beginnt. Hierbei sollte beachtet werden: Aktivitäten, die zwar nicht direkt als Schreiboperationen anzusehen sind, also z.B. ein Weiterscrollen, sollten nicht als *write_sth*-unterbrechend bzw. -abbrechend angesehen werden, wenn sie primär dem Fortfahren des Editierens dienen und nicht als eigenständig sinnvolle Aktion angesehen werden können. Im Rahmen weiterführender Untersuchungen kann es allerdings notwendig werden, solche Phänomene aufzusplitten.

[5] Bei Änderungen, die die Form eines *challenge_activity* haben, muss im Einzelfall entschieden werden, ob sie derart wirken, dass ein weiterlaufender Schreibprozess nicht mit einem neuen *write_sth* annotiert werden sollte.

#	Kodes [Dauer]	Äußerung	Kontext/Kommentar
1	*P1.explain_finding*/ Start eines *think aloud_activity* von *P1* [ca. 8 sek.]	„Das ist dann jeglich <*P1 beginnt mit dem Editieren*> unabhängig von der (..) Implementation. Solange die invers sind, (~), (.) Oder?“	Der Driver *P1* erklärt, warum es sich seiner Meinung nach bei dem vorher vom Partner abgegebenen Designvorschlag um eine gute Idee handelt (kurz vorher hatte er dem Vorschlag bereits zugestimmt). Das nachgeschobene „Oder?“ scheint rhetorischer Natur zu sein, denn *P1* beginnt schon beim Aussprechen des Wortes „unabhängig“ mit dem Editieren der betroffenen Kodezeile.
2	*P2.agree_finding* [< 1 sek.]	„[A]hm“	*P2* stimmt der Aussage des Partners unmittelbar folgend zu. *P1* editiert weiter.
3	*P1.agree_design* [ca. 1 sek.]	„Können‿wir‿ausprobieren“	*P1* schiebt unmittelbar nachfolgend eine sich auf den ursprünglichen Vorschlag von *P2* beziehende Zustimmung ein. Er stoppt nicht mit dem Editieren.
4	*P1.explain_finding* [ca. 3 sek.]	„<*P1 stoppt das Editieren*> Äh, was machen wir eigentlich (!!...!!) Haben wir überhaupt schon (!!...!!) Ach ne unser Skript läuft (!!...!!) Doch. Gut.“	Nach weniger als einer Sekunde denkt *P1* laut nach und kommt aufgrund dessen zu einer Erkenntnis.
5	*P1.amend_design* [ca. 2 sek.]	„Und einer muss jetzt `id_to_code` werden.“	Nach einer kurzen Pause (ca. 1 sek.) konkretisiert *P1* den Designvorschlag, indem er anmerkt, dass in der bearbeiteten Zeile einer der beiden Aufrufe von `code_to_id` zu `id_to_code` geändert werden muss.
6	*P1.explain_finding*/ Hiernach endet das *think aloud_activity* von *P1* [ca. 3 sek.]	„Wir haben einen `code`, `code_to_id` (!!...!!) <*P1 setzt das Editieren fort*>“	*P1* fährt unmittelbar mit Sprechen fort, indem er noch einmal festhält, welche Gestalt die editierte Zeile gerade aufweist. Hierbei fährt er mit der Maus über die entsprechenden Teile des Kodes (es handelt sich aber allem Anschein nach nicht um ein *show_sth*). Es stellt sich die Frage, ob mit dieser Äußerung ein anderer Zweck verfolgt wird als Zeit zu überbrücken. Unmittelbar danach fährt *P1* mit dem Editieren fort.
7	[ca. 10 sek.]		*P1* führt weitere Änderungen an der Zeile durch. Danach wechselt er in den Browser Firefox, um das editierte Skript zu testen. Damit endet das *write_sth* und eine neue HCI-Aktivität beginnt.

Tab. 8.2: Episode aus Sitzung PR1.1 (Start 13:55:24), die aus pragmatischer Sicht mit einem *write_sth* zu kodieren ist. Sie beginnt, nachdem *P1* einem Designvorschlag des Partners zugestimmt hat. Es ist zu erkennen, dass hierbei kurze Schreibpausen (Zeile 4-6; Dauer ca. 9 sek.), die im direkten Zusammenhang mit dem Editieren stehen, nicht dazu führen, dass eine Handlungsentität mit mehreren Konzepten vom Typ *write_sth* zu annotieren ist. Für weitere Anmerkungen siehe Abbildung 8.1.

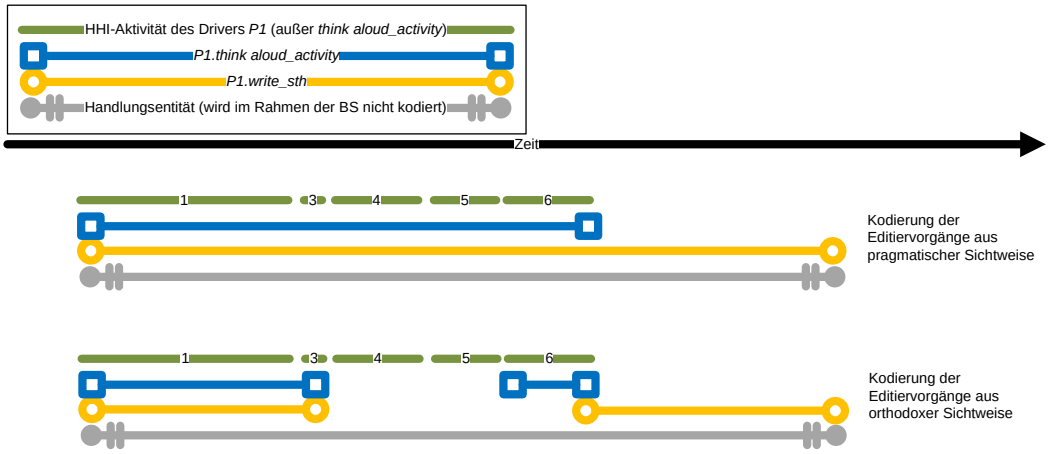


Abb. 8.1: Pragmatische vs. orthodoxe Sicht auf Editiervorgänge am Beispiel der in Tabelle 8.2 erläuterten Episode. Die Ziffern an den HHI-Aktivitäten verweisen auf die zugehörigen Zeilen in Tabelle 8.2. Die orthodoxe Sicht ist hier derart definiert, dass Schreibpausen von mehr als zwei Sekunden als Ende eines *write_sth* gewertet werden müssen. Es ergibt sich, dass Äußerungen, die vom Inhalt her sicherlich ein *think aloud_activity* darstellen (wie die fünfte: „Und einer muss jetzt `id_to_code` werden.“) nicht mehr einem solchen zurechenbar sind – zumindest dann, wenn nicht auf den Kunstgriff eines langen „Einlaufs“ (siehe S. 351) zurückgegriffen wird. Die sechste Äußerung („Wir haben einen `code`, `code_to_id` (!!...!!)“) wurde als ein solcher Einlauf gewertet. Allerdings ergibt sich aus dieser Entscheidung der Sonderfall, dass das *think aloud_activity* genau zu dem Zeitpunkt endet, an dem die zugehörige Aktivität beginnt.

8.1.2 Das Konzept *search_sth*

Mit dem Konzept *search_sth* werden Teile von Handlungsentitäten adressiert, in denen der Handelnde nach dem Vorkommen von wohldefinierten Objekten oder Gruppen von Objekten wie z.B. Klassen, Methodendeklarationen oder Strings sucht (für die Suche nach nicht wohldefinierten Objekten siehe Erläuterungen zu *explore_sth* in Abschnitt 8.1.3). Es gilt:

- Ein Objekt oder eine Gruppe von Objekten soll hier *wohldefiniert* heißen, wenn der Suchende glaubt, ausreichende Informationen dahingehend zu besitzen, das gesuchte Objekt derart referenzieren zu können, dass er es finden kann, dass er also – zumindest allem Anschein nach – davon ausgeht, seine geplante Suche derart operationalisieren zu können, dass er das gesuchte Objekt letztendlich entweder findet oder nachgewiesen hat, dass dieses (höchstwahrscheinlich[6]) nicht existiert. Dies ist z.B. der Fall, wenn er weiß, genau genommen zu wissen meint,

  - wie die Methode heißt, deren Deklaration oder deren Aufruf er sucht,

[6] Es kommt vor, dass letzte Zweifel bleiben.

  - unter welcher Zeilennummer die gesuchte Information zu finden ist oder
  - wie der reguläre Ausdruck aussieht, mit dessen Hilfe er bestimmte Stellen im Kode anspringen kann.

- Das Konzept *search_sth* ist sowohl dann zu verwenden, wenn es darum geht festzustellen, ob etwas überhaupt existiert, als auch dann, wenn herausbekommen werden soll, wo es zu finden ist.

- Mit dem Konzept *search_sth* kann sowohl eine händische Suche, z.B. via Scrollen durch den Kode, wie auch eine Suche unter Zuhilfenahme von Werkzeugen, wie z.B. der Suchfunktionalität einer IDE, adressiert werden.[7] Achtung: Im Rahmen der BS wird eine werkzeugunterstützte Suche bis zu dem Zeitpunkt einem *search_sth* zugerechnet, an dem das Werkzeug ein Ergebnis liefert oder der Suchende seine Aufmerksamkeit einer anderen Tätigkeit zuwendet. Dies bedeutet, dass ein Warten auf Ergebnisse, also eine weitgehend passive Tätigkeit (*aktives Warten*[8]) noch als *search_sth* zu werten ist. Hinter dieser Regel steht die folgende Überlegung: Auch wenn der Entwickler keine Handlungen mehr vollzieht, befindet er sich dennoch immer noch im Suchmodus. Dasselbe gilt in der Regel auch, wenn die Zwangspause für eine andere Tätigkeit genutzt wird (hierauf wird noch einmal im Zusammenhang mit dem Konzept *verify_sth* in Tabelle 8.6 eingegangen). In weiterführenden Untersuchungen kann es vielfältige Gründe geben, diese Regel zu brechen oder aufzuweichen, z.B. indem „passive“ Aktivitäten durch Eigenschaften markiert werden. Darüber hinaus stellt sich die Frage, ob ein Mitwarten des Partners auch als *search_sth* zu kodieren ist. Im Rahmen der Herleitung der BS wurde nicht so verfahren, im Rahmen weiterführender Untersuchungen kann dies aber angebracht sein. Auf derartige Phänomene wird in Abschnitt 8.1.9 noch einmal eingegangen.

- Für den Einsatz des Konzeptes *search_sth* ist es unerheblich, ob in virtuellen Dokumenten, also z.B. in Dateien, die am Bildschirm dargestellt sind, oder in physischen Dokumenten, z.B. ausgedruckten Dokumentationen, gesucht wird. Das Konzept gehört somit sowohl zur Klasse HCI wie auch zur Klasse HEI.

- Ebenso spielt es keine Rolle, ob ein geeignetes/effizientes oder eher ungeeignetes/uneffizientes Suchverfahren angewendet wird.

[7] Viele Entwicklungsumgebungen besitzen Wizards, mit denen Objekte derart geöffnet werden können, dass der Entwickler beim Tippen direkt die zu seiner bisher eingegebenen Zeichenkette passenden Objekte angezeigt bekommt (z.B. der „Open Typ“-Dialog der IDE Eclipse). Ob bzw. in welchen Situationen die Verwendung solcher oder ähnlicher Hilfsmittel als Suche gewertet werden soll, überlässt die BS/BKM nachfolgenden Untersuchungen.

[8] `solange (werkzeugunterstützte Suche noch nicht abgeschlossen) beginne keine neue HCI/HEI-Aktivität;`

Ein *search_sth*-Phänomen beginnt in der Regel mit der ersten physischen Tätigkeit die zur Suche gehört und endet, wie bereits erläutert, entweder mit der letzten diesbezüglichen physischen Tätigkeit bzw. wenn sich der Suchende einer anderen Tätigkeit zuwendet oder wenn die Suche abgeschlossen ist. Aktives Warten wird im Rahmen der BS – wie oben beschrieben – als zur Suche gehörig gewertet. Mit Unterbrechungen sollte, ähnlich wie es im Zusammenhang mit dem Konzept *write_sth* erläutert wurde, pragmatisch umgegangen werden. Ein diesbezüglich interessantes Beispiel wurde bereits in Tabelle 7.10 (siehe Zeile 1) angesprochen und wird nun in Tabelle 8.3 näher betrachtet. Driverwechsel während einer Suche konnten bisher nicht beobachtet werden.

8.1.3 Das Konzept *explore_sth*

Das Konzept *explore_sth* adressiert Teile von Handlungsentitäten, in denen der Handelnde Artefakte oder Gruppen von Artefakten bzw. Dateien/Dokumente oder Gruppen von Dateien/Dokumenten (z.B. Kode oder Dokumentationen) erkundet oder sondiert.[9]

Es gilt:

- Das Konzept *explore_sth* wird sowohl dann eingesetzt, wenn sich ein Entwickler Überblick über den Inhalt von ihm (weitgehend) unbekannten, z.B. bereits vor dem Beginn der Sitzung existierenden Artefakten bzw. Dokumenten oder Gruppen von Artefakten bzw. Gruppen von Dokumenten verschaffen will, wie auch dann, wenn er – meist in einzelnen solchen Artefakten/Dokumenten – nach derzeit nur grob verstandenen oder schwer greifbaren Objekten sucht. Das Konzept sollte insbesondere nur dann verwendet werden, wenn es nicht um Inhalte geht, die von einem der Entwickler im Rahmen der Sitzung erstellt wurden.

- Im Einzelnen ist das Konzept *explore_sth* in drei Fällen zu verwenden

  1. Wenn es darum geht, festzustellen ob etwas grob charakterisiertes (also nicht wohldefiniertes) überhaupt existiert (z.B. eine Methode, die eine bestimmte Funktionalität erfüllt).
  2. Wenn herausbekommen werden soll, wo etwas derartiges zu finden ist.
  3. Wenn etwas derartiges oder etwas nicht gut bzw. gar nicht Verstandenes (bzw. seine Eigenschaften) besser verstanden werden soll. Insbesondere kommt es auch dann zum Einsatz, wenn ein Entwickler durch bestehende Artefakte/Dokumente navigiert, um sich dahingehend inspirieren zu lassen, wie weiter vorgegangen werden sollte.

[9] Dies kann auch bedeuten, dass ein Entwickler die Einstellungen der verwendeten Applikationen, z.B. die der IDE, erkundet. Schließlich sind auch diese letztendlich in Dateien abgelegt.

#	Kodes [Dauer]	Äußerung	Kontext/Kommentar
1	*P1.propose_step/ P1.show_sth* [ca. 2 sek.]	„Wir◡können◡auch◡ mal◡kucken◡wo◡das◡ hier◡überall◡ verwendet◡wird.“	Der Driver schlägt vor, nach Aufrufen einer bestimmten Methode zu suchen. Dabei zeigt er mit dem Cursor auf einen Methodennamen.
2	Start eines *P1.search_sth* || Start eines *P2.search_sth*		Unmittelbar nach dem Vorschlag beginnt *P1* mit der Suche. Er verwendet die Suchfunktionalität von Eclipse.[a]
3	*P1.explain_finding/ P1.think aloud_activity* [ca. 4 sek.]	„(#Workspace#) (!...!) (, ,) Workspace ist (~).“	Ca. 4 Sekunden später äußert sich *P1* zu seinen Aktivitäten („(#Workspace#)“). Zu diesem Zeitpunkt ist der *Search*-Dialog von Eclipse geöffnet. Der Cursor befindet sich über dem Teil, in dem der Anwendungsbereich (*Scope*) der Suche festgelegt werden kann. Es kann zwischen *Workspace* (selektierter Default) und *Working Set* gewählt werden. *P1* ändert nichts und startet die Suche über die Schaltfläche *Search*. Eine Fortschrittsanzeige erscheint (*Scanning file 1 of 2.348: [...]*). Danach fährt er mit seinem *think aloud_activity* fort.
4	*P1.say_off topic* [ca. 1 sek.]	„Muffins◡gibt's◡hinterher.“	Ca. 2 Sekunden später macht *P1* eine Off-Topic-Bemerkung.
5	*P2.say_off topic* [ca. 3 sek.]	„<*Lachen*>. Belohnung gibt's danach.“	*P2* reagiert auf den Einwurf des Partners. Die Suche läuft noch.
6	*P2.explain_finding* [ca. 2 sek.]	„2350 Dateien haben wir!? Oiiii.“	Ca. 2 Sekunden später äußert *P2* eine Erkenntnis, die allem Anschein nach durch die Wahrnehmung des Textes in der Fortschrittsanzeige ausgelöst wurde.
7	*P1.explain_finding* + *P1.amend_step*[b] [ca. 6 sek.]	„Ähhh, (.) ja, ist natürlich auch totaler Unsinn – wir können in unserem *Working Set* suchen. Ich dachte (~) ich hätte eh nur das ausgecheckt, aber (!...!)“	Ca. eine Sekunde später erkennt *P1*, dass die Festlegung des Anwendungsbereichs der Suche ungünstig war (Type T_C). Während er spricht beendet er die Suche, die gerade beim Scannen der 666sten Datei ist.
8			Danach startet *P1* die Suche neu. Er wählt ein *Working Set* aus.

[a] Die HCI-Kodierung für *P2* wird erst in Abschnitt 8.1.9 erläutert.

[b] Da *P1* im ursprünglichen Vorschlag die adverbiale Bestimmung „überall“ verwendete, handelt es sich um eine Äußerung auf der „Grenze“ zwischen *amend* und *challenge*.

Tab. 8.3: Episode aus Sitzung PR1.1 (Start 14:51:41), in der ein werkzeugunterstütztes *search_sth* gestartet, wegen Unangemessenheit wieder gestoppt und danach in modifizierter Weise erneut gestartet wird. Da diese Aktionen kontinuierlich im Rahmen der Verfolgung eines nicht wesentlich modifizierten Handlungszieles stattfinden und von den Entwicklern aktives Warten (in Hinsicht auf HCI/HEI-Aktivitäten) praktiziert wird, ist im Rahmen der *BS* nur ein *search_sth* zu kodieren bzw. nicht von zwei Suchhandlungen zu sprechen.

Auch wenn mehrere dieser Ziele im Rahmen einer Erkundung zusammen direkt hintereinander angegangen werden, wird das zugehörige Phänomen als ein *explore_sth* kodiert.

- Im Rahmen der BS/BKM wird nicht zwischen einem eher oberflächlichen und einem detaillierten Erkunden unterschieden. Somit ist auch ein weitgehend stilles Lesen von Dokumenten mit *explore_sth* zu kodieren. Wenn es sich allerdings um ein reines Vorlesen handelt, kommt das Konzept *read_sth* zum Einsatz (siehe Abschnitt 8.1.5), wenn die im Rahmen der Sitzung geleistete Arbeit verifiziert wird, *verify_sth*.

- Im Rahmen der hier verwendeten Analysemethoden kann in der Regel nicht zwischen einem Erkunden und einem (zwischenzeitlichen) stillen Nachdenken über das bisher Erkundete oder im Weiteren zu Erkundende unterschieden werden. Derartige kognitive Leistungen werden deshalb im Rahmen der BS dem *explore_sth* zugerechnet. Dies hat zur Folge, dass es in machen Situationen sehr schwierig ist, einigermaßen sicher festzustellen, zu welchem Zeitpunkt ein Entwickler mit einer Erkundung fertig ist.

- In ähnlicher Weise wie beim Konzept *search_sth*, ist es auch beim Einsatz von *explore_sth* unerheblich, ob virtuelle Artefakte/Dokumente (Dateien und Strukturen auf dem Bildschirm) oder physische Artefakte/Dokumente (in der Regel in Papierform) erkundet werden. Somit gehört auch *explore_sth* sowohl zur Klasse HCI wie auch zur Klasse HEI. Ein Erkunden von physischen Dokumenten konnte allerdings bisher nicht beobachtet werden. Außerdem spielt es keine Rolle, ob es sich um temporäre Artefakte/Dokumente (z.B. dynamisch erzeugte Seiten, die im Browser angezeigt werden) oder nicht-temporäre Artefakte/Dokumente handelt.

- Bisher konnten folgende, ein *explore_sth* manifestierende physische Tätigkeiten beobachtet werden:

  - Scrollen durch eine oder mehrere Programmdateien.
  - Scrollen durch Dokumentationen.
  - Navigieren durch den Dateibaum im *Package Explorer* von Eclipse.
  - Navigieren durch den Dateibaum auf der Festplatte (unter Zuhilfenahme des Dateiverwaltungsprogramms Windows-Explorer).
  - Scrollen durch Fehlermeldungen (z.B. durch solche, die nach einem Aufruf einer JSP-Seite[10] im Browser erscheinen). Hierbei handelt es sich in der Regel um eine Nachfolgeaktivität zu einem *verify_sth*. Details hierzu werden auf S. 389 erörtert.

[10] Siehe Fußnote 20 auf S. 90.

– Navigieren durch die Einstellungen (*Preferences*) der IDE.[11]

Beispielsweise schlägt der Driver *P1* in Sitzung ST1.1 vor, nachzusehen „was der `Subscriber` so an Methoden anbietet“, wobei er sich mit dem Begriff `Subscriber` auf die Klasse `TopicSubscriber` der JMS-API[12] bezieht. Unmittelbar danach beginnt er damit durch die Javadoc-Seite[13] des Interfaces `TopicSubscriber` und dann durch die des Interfaces `MessageConsumer`[14] zu scrollen. Dabei verbalisiert er „Entdeckungen“ von Methoden, die potentiell interessant sein könnten.

Das Identifizieren von *explore_sth*-Phänomenen ist oft weniger einfach als das von *write_sth*- oder auch *search_sth*-Phänomenen. Dies liegt unter anderem daran, dass anders als bei vielen Phänomenen vom Typ *write_sth* oder *search_sth* mit einem *explore_sth* keine physischen (Rechner-) Operationen einhergehen müssen, die es eindeutig als ein solches identifizieren. Insbesondere kann es schwierig werden zwischen einem *search_sth* und einem *explore_sth* zu unterscheiden. So können sich beide dadurch manifestieren, dass der Driver durch ein Klasse scrollt. Im Rahmen der Herleitung der BS/BKM wurden derartige Phänomene, falls sie nicht sicher als *search_sth* interpretiert werden konnten, *explore_sth* zugewiesen.

Eine Kodierung mit einem *explore_sth* muss spätestens dann beginnen, wenn der Erkundende die erste physische Tätigkeit, die zu seiner Erkundung gehört, durchführt und endet frühestens dann, wenn die letzte diesbezügliche physische Tätigkeit stattfindet. Es ist aber nicht sinnvoll beim Identifizieren von Erkundungen nur physische Tätigkeiten zu berücksichtigen. Ein *explore_sth* kann z.B. lange nach der letzten physischen Aktivität eines Erkundenden enden, zum Beispiel erst dann, wenn er sich einer anderen Tätigkeit zuwendet. Wie bei *write_sth* und bei *search_sth* sollte auch bei *explore_sth* eine pragmatische Sichtweise eingenommen werden. Wechselt der Driver beim Erkunden z.B. die Datei, wird dieses Ereignis nicht einzeln (durch ein *do_sth* (siehe Abschnitt 8.1.8)) und unterbrechend kodiert. Driverwechsel während einer Erkundung konnten bisher nicht beobachtet werden.

Drei Eigenschaften von *explore_sth*-Phänomenen sollen noch hervorgehoben werden:

1. Wie bereits angedeutet, finden im Rahmen von *explore_sth*-Phänomenen nicht notwendigerweise physische Aktivitäten statt. Solche Aktivitäten fehlen z.B. dann, wenn ein Observer unbekannten Kode, der gerade auf dem

[11] Im beobachteten Fall hatte der Driver allem Anschein nach die Hoffnung, dass er durch das Navigieren durch bestimmte Bereiche der Einstellungen der IDE eine Idee bekommt, warum sich diese auf eine bestimmte Weise verhält.

[12] Siehe Fußnote 21 auf S. 90.

[13] Mit Javadoc können API-Dokumentationen in HTML für Java-Programme erzeugt werden (http://www.oracle.com/technetwork/java/javase/documentation/index-jsp-135444.html (Abruf: 31.10.2011)).

[14] `public interface TopicSubscriber extends MessageConsumer`

Display angezeigt wird, erkundet. In derartigen Fällen ist es besonders schwierig, eine Kodierung mit *explore_sth* zu begründen. Schließlich kann ein längeres Blicken auf das Display auch eine andere Ursache haben. Oft hilft es hier, sich den Kontext, speziell die nachfolgenden Aktivitäten der Person, näher anzusehen.

2. Man kann zumindest zwischen drei Formen von Erkundungen bzw. Sondierungen unterscheiden:

   (a) **Erkundungen, die Handlungsentitäten notwendigerweise fortführen**: Hierbei handelt es sich um solche Erkundungen, die sich mehr oder weniger zwangsläufig aus einer anderen HCI/HEI-Aktivität ergeben bzw. die einer vorhergehenden Tätigkeit letztendlich erst Sinn verleihen. Ein Beispiel hierfür bilden die Durchsichten der Ergebnisse von Testaufrufen einer PHP-Seite, also Sondierungen der Resultate eines *verify_sth*. Solche Durchsichten können für einen Beobachter kaum wahrnehmbar sein, da ein Entwickler sie durch einen kurzen Blick auf den Bildschirm ausüben kann. Im Rahmen der Herleitung der BS/BKM wurde diese Form der Erkundungen nur dann als *explore_sth* markiert, wenn es sich nicht nur um ein kurzes „Registrieren“ des Ergebnisses handelte (siehe auch Abbildung 8.2 im Abschnitt 8.1.4). Ein nachfolgendes kurzes „Registrieren“ von Ergebnissen wurde als Teil des *verify_sth* angesehen – zumindest dann, wenn es sich um eine Aktion derjenigen Person handelt, die auch das *verify_sth* durchführte. Dies kann zur Folge haben, dass eine Verbalisierung einer solchen Erkundung, z.B. in Form einer *explain_finding*-Äußerung, einem dem *verify_sth* zugehörigen *think aloud_activity* zugerechnet werden muss (siehe z.B. Zeile 7 in Tabelle 8.5). In weiterführenden Untersuchungen muss entschieden werden, ob es sinnvoll ist, dieser pragmatischen Vorgabe zu folgen.

   (b) **Erkundungen, die zu einer anderen HCI/HEI-Aktivität gehören:** Hierunter werden solche Erkundungen subsumiert, die nicht als eigenständige Handlungen zu werten sind, da sie im Rahmen anderer HCI/HEI-Aktivitäten stattfinden bzw. innerhalb einer solchen en passant durchgeführt werden. Beispielsweise kommt es vor, dass Driver im Rahmen eines „händischen“ Tests der in Entwicklung befindlichen Applikation Meldungen auf der Konsole der IDE im Auge behalten, also in gewisser Weise ein temporäres Artefakt sondieren. Im Rahmen der Herleitung der BS/BKM wurden solche Handlungen nicht einzeln als *explore_sth* markiert.

   (c) **Eigenständige Erkundungen:** Derart werden solche Erkundungen bezeichnet, die eine eigene Handlungsentität bilden oder zumindest einen weitgehend eigenständigen Teil einer solchen, also nicht nur einer anderen Tätigkeit untergeordnet oder zwangsläufig nachgeordnet

sind. Sie sind es, die primär durch das Konzept *explore_sth* adressiert werden.

3. Wenn *explore_sth*-Phänomene von einem *think aloud_activity* begleitet werden, kann dieses die Form eines (partiellen) Vorlesens besitzen. In diesen Fällen geht es dem Sprecher oft darum, bestimmte Entdeckungen weiterzugeben (*explain_finding*). Ein Beispiel ist in Tabelle 8.4 erläutert. Achtung: Bei derartigen Phänomenen kann es sich potentiell auch um solche vom Typ *read_sth* handeln (siehe Abschnitt 8.1.5).

#	Kodes [Dauer]	Äußerung	Kontext/Kommentar
1	*(P1.explain_finding* + *P1.ask_knowledge)*/ ((Start *P1.explore_sth)*/ (Start *P1.think aloud_activity*)) [ca. 1 sek.]	„Ey.‿ Hmm.‿ Was‿is'‿ 'n‿da‿ los?“	Nach dem Abschicken des Kommandos `wget "http://localhost"` (*do_sth*) erscheint eine Fehlermeldung: [...] `Auflösen des Rechnernamens 'localhost'` [...] `Verbindungsaufbau mit` [...] `fehlgeschlagen: Verbindungsaufbau abgelehnt.` *P1* sieht die Ausgabe durch und erkennt, dass es sich um eine Fehlermeldung handelt. Dann fragt er seinen Partner, nach einem Grund für deren Auftreten.
2	*P2.propose_hypothesis*/ (Start *P2.explore_sth)*/ *P2.think aloud_activity* [ca. 1 sek.]	„Firewall?“	*P2* äußert eine Hypothese. Mit dieser fällt er seinem Partner fast ins Wort. Spätestens mit dieser Äußerung wird ersichtlich, dass auch der Observer die Fehlermeldung erkundet.
3	*P1.explain_finding*/ Ende *P1.think aloud_activity* [ca. 1 sek.]	„(#Auflösen des Rechnernamens 'localhost'#)“	Nun fällt *P1* seinem Partner fast ins Wort. Er liest eine der Zeilen der Fehlermeldung vor, die er allem Anschein nach für besonders interessant erachtet. Im Rahmen der Herleitung der BS wurde das *P1.think aloud_activity* hiermit als beendet betrachtet. Grund: *P1* sieht zwar weiterhin auf den Monitor, äußert sich aber vorerst nicht mehr.
4	*P2.propose_hypothesis*/ *P2.think aloud_activity* [ca. 2 sek.]	„Vielleicht nur die (~~)?“	Nach zwei Sekunden Pause, in denen beide Entwickler schweigend in Richtung Monitor gesehen haben, äußert *P2* eine weitere Hypothese. Es konnte nicht ermittelt werden, was er genau vermutet. Allerdings scheint es sich bei der neuen Hypothese weder um eine Erweiterung noch um eine Korrektur der zuvor geäußerten zu handeln.
5	*P1.agree_hypothesis*/ Start *P1.think aloud_activity* [ca. 1 sek.]	„(Genau)“	Nach weiteren zwei Sekunden Pause, in denen beide in Richtung Monitor gesehen haben, stimmt *P1* der Hypothese des Partners zu.
6	*P1.explain_finding*/ Ende *P1.think aloud_activity*/ Ende *P1.explore_sth*/ Ende *P2.explore_sth* [ca. 1 sek.]	„(~) ist O.K.“	Unmittelbar anschließend macht der Driver eine Anmerkung, die allem Anschein nach noch im Zusammenhang mit dem Erkunden der Ausgabe steht. Danach fährt er mit Vorschlägen zum weiteren Vorgehen fort. Im Rahmen dieser Vorschläge wechselt er auch vom Konsolenfenster in ein Browser-Fenster.

Tab. 8.4: Episode aus Sitzung PR1.1 (Start 13:42:11), in der der Driver *P1* im Rahmen eines *explore_sth* Bildschirminhalte vorliest. Die Episode beginnt, direkt nachdem der Driver einen Vorschlag des Partners durchgeführt hat (Aufruf des Befehls `wget "http://localhost"` (siehe Fußnote 155 in Tabelle 7.35) in einer Konsole). Da das Erkunden des Ergebnisses der *do_sth*-Aktivität durch *P1* nicht nur in einem kurzen Registrieren besteht, wurde es separat kodiert. Ihm muss dann auch das erste *P1.explain_finding* zugerechnet werden (siehe auch S. 383). Auf die Kodierung von HCI/HEI-Aktivitäten des Observers wird in Abschnitt 8.1.9 genauer eingegangen.

8.1.4 Das Konzept *verify_sth*

Mit dem Konzept *verify_sth* werden solche Teile von Handlungsentitäten adressiert, in denen ein Entwickler Modifikationen an Artefakten, Dokumenten und Konfigurationen überprüft – und zwar solche, die innerhalb der laufenden Paarprogrammierungssitzung vorgenommen wurden. Hierbei spielt es keine Rolle, ob die Modifikationen von ihm selbst oder von seinem Partner vorgenommen wurden. Das Konzept unterscheidet außerdem nicht zwischen einer manuellen Überprüfung im Sinne einer Durchsicht (siehe z.B. [61]), Softwaretests (z.B. durch „ausprobieren“ oder die Verwendung von JUnit[15] bzw. anderer Testframeworks), Aufrufen von sonstigen Applikationen und Mischformen.[16] Auch ist es unerheblich, ob der Entwickler davon ausgeht, dass der zu überprüfende Kode richtig ist, oder davon, dass er Defekte enthält.

Ein *verify_sth* kann, muss aber nicht durch ein *propose_step* wie z.B. „Schau'n wir doch mal“[17] angekündigt worden sein. Durchsichten sind allerdings oft nur dann eindeutig als solche zu identifizieren, wenn entweder eine solche Ankündigung erfolgte oder ein begleitendes *think aloud_activity* stattfindet. Achtung: Im Rahmen der BS werden mit dem Begriff *verify* nicht ausschließlich Verifikationen adressiert. Bei einem *verify_sth* kann es sich vielmehr sowohl um eine Verifikation wie auch um eine Validierung handeln.[18]

Konkret konnten folgende Formen von *verify_sth*-Phänomenen beobachtet werden (zum Teil auch in Kombination):

- Durchsichten als Abschluss von Kodeänderungen. Es konnte beobachtet werden, dass hierbei nicht nur der modifizierte, sondern auch Kode im Kontext der Änderung betrachtet wird.
- Durchsichten als Abschluss von Änderungen an Konfigurationsdateien.
- Spätere (evtl. nochmalige) Durchsichten von länger zurückliegenden Änderungen. Auch bei solchen Durchsichten kommt es vor, dass Kode im Kontext der Änderungen betrachtet wird. Außerdem konnte beobachtet werden, dass vorgenommene Implementierungen auch unter direkter Zuhilfenahme von Dokumentation überprüft werden. In einem solchen Fall wurde zwischen dem Kode und der Dokumentation (PDF-Datei) hin und her gewechselt.
- Testaufrufe eines gerade modifizierten PHP-Skriptes im Browser.

[15] Siehe Fußnote *a* in Tabelle 7.17.

[16] Ein Konzept, mit dem Testaufrufe annotiert werden können, in denen es nicht primär darum geht, in der Sitzung selbst modifizierten Kode, sondern im Vorfeld geschriebenen zu überprüfen, existiert zur Zeit nicht, da derartige Handlungen/Phänomene bisher nicht beobachtet werden konnten.

[17] Hiermit leitet der Driver in Sitzung PR2.1 den testweisen Aufruf der in Entwicklung befindlichen Software ein.

[18] In der Softwareentwicklung geht es bei Verifikation darum zu überprüfen, ob das Produkt richtig gebaut wird und bei der Validierung darum, ob das richtige Produkt gebaut wird.

- Testaufrufe einer von einer Modifikation betroffenen JSP-Seite[19] einer Webapplikation.
- Aufrufe der in Entwicklung befindlichen Applikation und ihrer durch die gerade vorgenommenen Änderungen betroffenen Funktionalitäten.
- Funktionsaufrufe innerhalb einer bereits laufenden Instanz der in Entwicklung befindlichen Applikation/Software.
- Aufrufe von JDemo[20].
- Start eines Deploy-Vorganges[21] mit dem Ziel, Änderungen an Konfigurationsdateien zu verifizieren.
- Start eines *Build*-Vorganges[22], mit dem Ziel zu überprüfen ob Änderungen am Kode fehlerfrei durchgeführt wurden.
- Durchsicht der Logdatei/Logausgaben des Applikationserver JBoss[23], um zu überprüfen, ob bestimmte Funktionalitäten der in Entwicklung befindlichen Applikation durchgeführt wurden.
- Aufruf einer externen Applikation, um festzustellen, ob bestimmte Funktionionalitäten der in Entwicklung befindlichen Applikation wie erwartet funktionieren. In einem beobachteten Fall wurde z.B. ein Mailprogramm gestartet, um zu überprüfen, ob die in Entwicklung befindliche Applikation eine Mail an einen bestimmten Adressaten versendet hat.
- Durchsicht von Konfigurationseinstellungen in der IDE Eclipse.
- Aufruf der JMX Console[24] von JBoss, um zu überprüfen ob JBoss-Konfigurationen richtig durchgeführt wurden.
- Abschließende Durchsichten von gerade vorgenommenen Änderungen an Batch-Dateien. In den beobachteten Fällen wurden durch die Anweisungen in den Batch-Dateien keine Funktionalitäten in Hinsicht auf die in Entwicklung befindliche Applikation zur Verfügung gestellt, sondern solche, die im Rahmen der Paarprogrammierungssitzung selbst von Nöten waren (z.B. um Dateien auf einen entfernten Server zu übertragen).
- Testaufrufe gerade modifizierter Batch-Dateien.

[19] Siehe Fußnote 20 auf S. 90.
[20] Siehe Fußnote 217 auf S. 364.
[21] Siehe Fußnote *a* in Tabelle 7.32.
[22] Siehe Fußnote *a* in Tabelle 7.32.
[23] Siehe Fußnote 15 auf S. 87.
[24] Der Applikationsserver JBoss stellt eine JMX Console zur Verfügung, über die alle registrierten Services (MBeans) eingesehen werden können (siehe auch Fußnote 153 in Tabelle 7.35).

– Überprüfung, ob alle (bisher) notwendigen Arbeitsschritte durchgeführt wurden. Derartige *verify_sth*-Phänomene unterscheiden sich von den bisher diskutierten dadurch, dass nicht die Änderungen an und für sich überprüft werden, sondern in gewisser Weise der bisher durchlaufene Prozess.

In Hinsicht auf den Beginn eines *verify_sth* soll im Rahmen der BS folgendes gelten: Wird ein Softwaretest durch „ausprobieren“ vollzogen, so startet eine Kodierung mit diesem Konzept spätestens zu dem Zeitpunkt, an dem der Driver die zu testende Applikation aufruft, auch wenn der Aufruf selbst nur eine Voraussetzung für die Durchführung des Tests und nicht sein eigentliches Ziel darstellt. In allen anderen Fällen wird analog verfahren: Ein Test, wie auch ein Review beginnt mit dem Schaffen der Voraussetzungen für seine Durchführung, also z.B. mit dem Aufruf der Datei, die inspiziert werden soll. Achtung: Das Schreiben von Unit-Tests oder das Erstellen von anderweitigen Testskripten wird nicht als *verify_sth* gewertet, sondern muss im Rahmen der BS mit *write_sth* annotiert werden. Allerdings konnte weder das Erstellen von Unit-Tests noch deren Modifikation im Rahmen der hier analysierten Sitzungen beobachtet werden.

Darüber hinaus sollte bei der Verwendung von *verify_sth*, genauso wie bei den anderen bisher erläuterten HCI/HEI-Konzepten, eine pragmatische Sicht eingenommen werden. In diesem Zusammenhang sei auf folgende Phänomene hingewiesen:

– **Übergang eines *write_sth* in ein *verify_sth***: Editierschritte können fließend in Überprüfungen übergehen. Schließlich wird ein Driver im Normalfall einen Editiervorgang schon bei seiner Durchführung in einem gewissen Maße verifizieren. Solche Formen der Überprüfung wurden im Rahmen der Herleitung der BS/BKM nicht einzeln festgehalten. Es wurde (erst einmal) einzig ein *write_sth* annotiert. Nur wenn sich das Überprüfen als eigenständige Handlung, z.B. als eine, die nach dem Abschluss eines Editierschritts stattfindet, separieren ließ, wurde das Konzept *verify_sth* eingesetzt.Weiterführende Untersuchungen müssen hier evtl. präziser vorgehen und Doppelkodierungen vornehmen.

– **Übergang eines *verify_sth* in ein *write_sth***: Das Durchsehen von Kode oder anderen Artefakten kann direkt in ein Editieren münden, z.B. wenn ein entdeckter Defekt umgehend behoben wird. Solche Behebungsphasen werden im Rahmen der BS als *verify_sth*-unterbrechend gewertet und einzeln als *write_sth* markiert. Einen besonderen Fall stellt hierbei das Kommentieren von Kode dar. Weiterführende Studien müssen entscheiden, ob sie ein solches als zur Überprüfung gehörendes oder als separate Handlung betrachten wollen.

- **Sonstige Übergänge eines *verify_sth* in eine andere HCI/HEI-Tätigkeit (z.B. ein *explore_sth*)/Ende eines *verify_sth***:
  - **Bei Durchsichten**: Im Rahmen einer Durchsicht kann es z.B. dazu kommen, dass ein Entwickler auf Kode aufmerksam wird, der im Verlauf der Sitzung noch nicht verändert wurde. Sollten derartige Phänomene identifiziert werden, ist zu überprüfen, inwieweit es sich bei den mit den Phänomenen in Zusammenhang stehenden Handlungen noch um solche vom Typ *verify_sth* oder um die eines anderen Typs handelt, z.B. um ein *explore_sth*.
  - **Bei Tests**: Bei Testtätigkeiten stellt sich in besonderer Weise die Frage, wann man davon sprechen möchte, dass eine solche als abgeschlossen zu betrachtet ist. Folgende Zeitpunkte kommen dafür z.B. in Frage:
    - Wenn von den Entwicklern (zeitweise) keine diesbezüglichen physischen Aktivitäten mehr vorgenommen werden.
    - Wenn das Programm unter Test oder das verwendete Testwerkzeug die zu Testzwecken gestarteten Funktionalitäten vollständig durchgeführt hat.
    - Wenn einer der Entwickler erkennt, dass der Test bestanden wurde oder fehlgeschlagen ist und diese Erkenntnis verbalisiert.

    Für jedes einzelne dieser Kriterien (sowie auch Kombinationen dieser) sind Situationen vorstellbar, in denen seine (bzw. ihre) Anwendung zu unbefriedigenden Ergebnissen führen würde – z.B. ein *verify_sth* zu früh oder zu spät beendet wird – oder in denen es potentiell schwierig ist, diese zu operationalisieren. Beim letzten Kriterium stellt sich darüber hinaus die Frage, ob bzw. in welchen Situationen die Erkenntnis noch zum *verify_sth* gehört oder schon mit einem anderen Konzept, z.B. einem *explore_sth* kodiert werden sollte. Im Rahmen der Herleitung der BS wurde pragmatisch vorgegangen: In Zweifelsfällen wurde nach dem frühsten Zeitpunkt gesucht, für den sich intersubjektiv nachvollziehbar begründen lässt, dass der handelnde Entwickler nicht mehr dabei ist, die Testtätigkeit durchzuführen. Beispielsweise wurde ein vom Driver durchgeführtes *verify_sth* spätestens zu dem Zeitpunkt als abgeschlossen gewertet, an dem der Driver seine Aufmerksamkeit auf eine andere, nicht die Durchführung des Tests unterstützende HCI/HEI-Aktivität lenkt – zumindest, wenn er nicht kurzfristig zur Testtätigkeit zurück wechselt. Also z.B. dann, wenn der testweise Aufruf einer in Entwicklung befindlichen Applikation zu einem Versagen führt, in Folge dessen das Programm einen Stacktrace ausgibt, der vom Driver detailliert angesehen wird (Übergang zu einem *explore_sth*; siehe z.B. Abbildung 8.2). Allerdings folgt einem *verify_sth* nicht immer direkt eine andere HCI/HEI-Aktivität. In solchen Fällen, also z.B. dann, wenn die physischen Testaktivitäten nahtlos in eine

Diskussion über Resultate übergehen, ohne dass ein Erfolg dieser explizit herausgestellt wird, ist es oft besonders schwierig zu entscheiden, wann man davon sprechen möchte, dass das *verify_sth* als abgeschlossen angesehen werden sollte. In Tabelle 8.5 ist ein solches Phänomen erläutert. Allerdings wird hier ein *explain_finding* und ein Markieren eines Teiles des Resultates mit der Maus als Ende des *verify_sth* gewertet. Man stelle sich aber vor, dass diese Handlungen nicht stattgefunden hätten. Wann würde man dann davon sprechen wollen, dass das *verify_sth* abgeschlossen ist? Die BS überlässt die Antwort auf diese Frage weiterführenden Untersuchungen, z.B. solchen, die sich mit Verifikationsphänomenen bei der PP auseinandersetzen. Denn erst nähere Erkenntnisse über Verifikationstätigkeiten können hier zu letztendlich zweckmäßigen Kodierungen führen. Außerdem muss an dieser Stelle noch einmal darauf hingewiesen werden, dass es im Rahmen von qualitativen Studien weniger darauf ankommt, wie einzelne Phänomene kodiert werden, als vielmehr darauf, welche Erkenntnisse beim Kodieren erlangt werden. Darüber hinaus ist die BS/BKM (wie bereits eingangs erwähnt) keineswegs dahingehend konzipiert, dass zu jedem Zeitpunkt pro Person maximal ein HCI/HEI-Konzept einzusetzen ist. Dies gilt selbst dann, wenn man von Kodierungen mit *show_sth* absieht. Diese Entscheidung liegt darin begründet, dass eine Person durchaus zwei (oder mehr), durch die BKM separat zu markierende Handlungsziele gleichzeitig verfolgen kann. Ein diesbezügliches Beispiel ist in Tabelle 8.6 zu finden.

- **Tests, die in Durchsichten münden**: Tests von Applikationen können, insbesondere dann, wenn im Rahmen des Tests ein Versagen festgestellt wird, direkt in eine Durchsicht münden. Im Rahmen der Herleitung der BS/BKM wurden die beiden Überprüfungsschritte separat konzeptualisiert.
- **Durchsichten ohne physische Aktivitäten/Durchsichten des Observers**: Auf S. 382 wurde erörtert, dass Erkundungen (*explore_sth*) ohne physische Aktivitäten stattfinden können. Für Durchsichten gilt dasselbe, z.B. dann, wenn sie vom Observer durchgeführt werden. In solchen Fällen ist es oft schwierig nachzuweisen, ob es sich tatsächlich um eine Durchsicht handelt. Weiterführende Untersuchungen, insbesondere solche, die sich mit der Behauptung auseinandersetzen, dass die PP zu einem kontinuierlichen Reviewprozess führen kann (siehe z.B. [219]), müssen Regeln entwickeln, anhand derer Review-Phänomene ohne physische Handlungen, speziell solche des Observers, zu identifizieren sind.

Die Erläuterungen zum Konzept *verify_sth* machen noch einmal deutlich, auf welcher „Ebene“ sich die HHI/HEI-Konzepte bewegen: Mit einem *verify_sth* wird die Umsetzung eines Teilziels adressiert, das mit einer Handlung im Rahmen

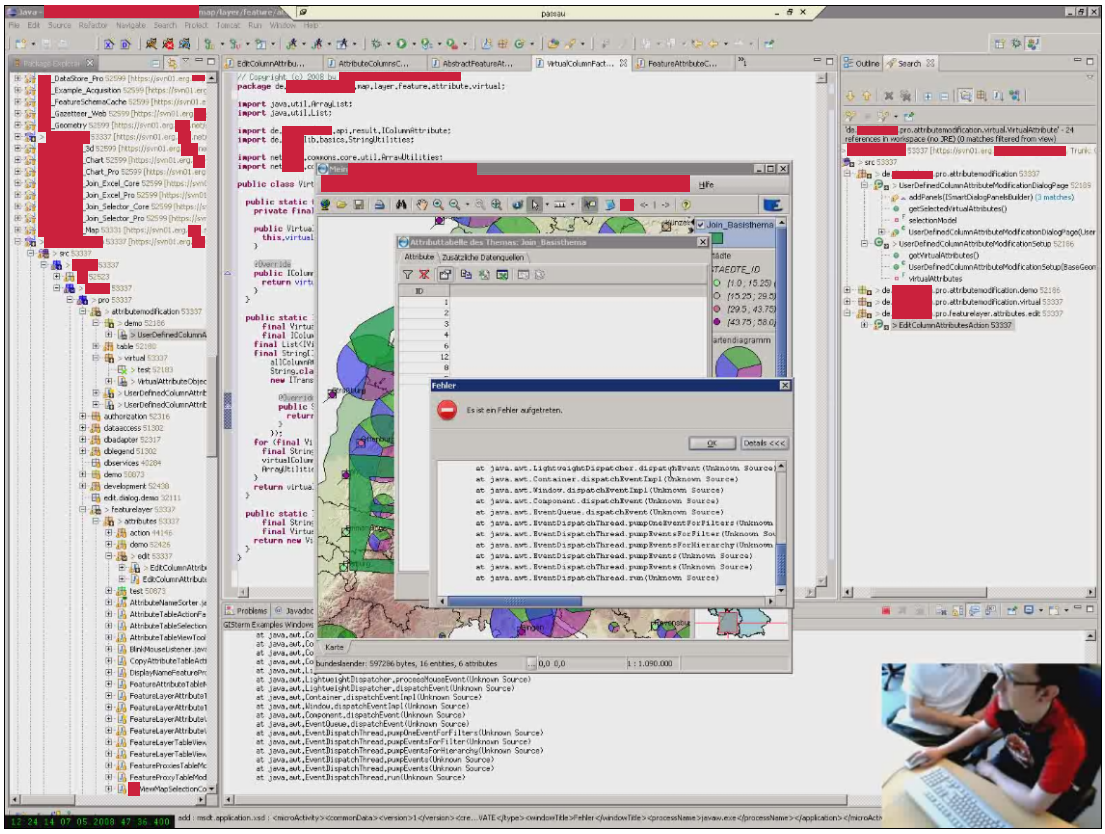


Abb. 8.2: Screenshot aus Sitzung PR2.1 (12:24:17). Es ist der Zeitpunkt dargestellt, zu dem der Driver von einem *verify_sth* zu einem *explore_sth* wechselt. Vier Fenster sind zu sehen: Im Vordergrund (und im Fokus) eine gerade erschienene Fehlermeldung der sich im Test durch Ausprobieren befindlichen Applikation, direkt dahinter ein Anzeige- und Eingabefenster dieser Applikation, dahinter das Hauptfester der Applikation und im Hintergrund die IDE Eclipse. In der unteren rechten Ecke ist das Bild der Videokamera eingeblendet, mit welcher die beiden Entwickler beobachtet wurden. Die Sitzung läuft seit 47 Minuten und 36 Sekunden. Folgendes hat sich gerade abgespielt: Während der Driver (rechts) dabei war, die in Entwicklung befindliche Applikation bez. zuvor vorgenommener Änderungen zu testen (*verify_sth*), wurde ein Versagen („Es ist ein Fehler aufgetreten“) in einem separaten Fenster gemeldet. Daraufhin brach der Driver seinen Test ab und und begann damit, sich den Stacktrace der Fehlermeldung anzusehen (*explore_sth*). Das Erkennen der Fehlermeldung als eine solche markiert hier die Grenze zwischen dem *verify_sth* und dem *explore_sth*. An derartigen Stellen sollten weiterführende Untersuchungen zwischen *verify_sth*-abbrechenden oder -unterbrechenden Erkundungen und solchen, die sozusagen en passant – und in der Regel zur Unterstützung der Testtätigkeit – vorgenommen werden und nicht einzeln zu kodieren sind, unterscheiden. Bei letzteren handelt es sich in der Regel um Erkundungen, die natürlicherweise zu einem *verify_sth* gehören - insbesondere zu einem vom Typ Test. Schließlich stellt es keine Besonderheit dar, dass Driver beim Durchführen von Tests auf Rückmeldungen, also auf temporäre Artefakte, achten, ohne das Testen wirklich zu unterbrechen.

#	Kodes [Dauer]	Äußerung	Kontext/Kommentar
1	(*P1.propose_step*/ Start eines *P1.think aloud_activity*/ Start eines *P1.verify_sth*) || (*P2.propose_step*/ Start eines *P2.verify_sth*) [ca. 1 sek.]	„<*P1:*> Können‿ wir‿mal‿ausprobieren, oder?“ || „<*P2:*> Shoot!“	Beide Entwickler äußern parallel den Vorschlag, das Skript zu testen, also es im Browser aufzurufen. Während der Driver *P1* spricht, holt er den Browser in den Fokus. Das *think aloud_activity* beginnt also kurz vor der zugehörigen HCI-Aktivität.[25]
2	*P1.ask_knowledge* [ca. 1 sek.]	„O.K.,‿und‿was‿hatten‿wir‿jetzt‿gesagt?“	Nach einem Zögern von ca. einer Sekunde fragt der Driver *P1* noch einmal nach, welches Ergebnis beim Aufruf der Seite zu erwarten ist.[26]
3	*P2.explain_knowledge* [ca. 1 sek.]	„Wir kriegen was.“	*P2* antwortet unmittelbar und stellt fest, dass sie die Bedingung und die Werte[27] so gesetzt haben, dass zu erwarten ist, dass bei einem Aufruf der Seite bzw. des Skriptes Datensätze aus der Datenbank angezeigt werden.
4	*P1.disagree_knowledge* [ca. 1 sek.]	„Echt?“	Ohne einen Moment zu warten, zieht *P1* die Erklärung seines Partners in Zweifel.[28]
5	[ca. 1 sek.]		Direkt danach klickt er auf die „Reload“-Schaltfläche des Browsers[29] – erst einmal und weniger als eine Sekunde später noch diverse weitere Male. Nach dem ersten Klick ändert sich die angezeigte Seite, nach den weiteren bleibt sie unverändert. In den ersten beiden Zeilen der Ergebnisseite sind Hilfs- bzw. Testausgaben[30] zu sehen. In den nachfolgenden Zeilen jeweils ein Datenbankeintrag (Integers).
			Fortsetzung auf der nächsten Seite.

[25] Die HCI-Kodierung für *P2* wird erst in Abschnitt 8.1.9 erläutert.

[26] Im Rahmen der Herleitung der BKM wurde an dieser Stelle erwogen, ein Konzept *ask_hypothesis* einzuführen. Da es sich bei der Frage aber nicht um die Aufforderung handelt, eine neue Hypothese abzugeben sondern lediglich darum, eine bereits erörterte zu wiederholen, wurde zugunsten einer *ask_knowledge*-Kodierung entschieden.

[27] Bestimmte Werte, die den Ablauf des Skriptes beeinflussen, sind zu Testzwecken im Skript *hard-coded*.

[28] Für Details zur Kodierung siehe auch S. 326. Hinweis: Es konnte nicht ermittelt werden, ob *P1* das erläuterte Verhalten an und für sich oder die Behauptung, dass genau dies Verhalten im Vorfeld erwartet wurde, anzweifelt.

[29] In der Adresszeile des Browsers steht bereits die URL des aufzurufenden Skriptes.

[30] `Last request: 20.06.2007 15:52:27` und `Last change: 20.06.2007 15:54:07`

			Fortsetzung der Tabelle der vorherigen Seite.
7	*P1.explain_finding/* Ende des *P1.verify_sth/* Ende des *P1.think aloud_activity*|| Ende des *P2.verify_sth*[31] [ca. 1 sek.]	„Àhhh.“	Unmittelbar nach dem letzten Klick macht *P1* einen Ausruf, der signalisiert, dass er etwas erkannt hat – aller Wahrscheinlichkeit nach das, was der Observer kurz zuvor prophezeit hatte. In welcher Weise er dies wertet, ist der Äußerung allerdings nicht zu entnehmen.
8	*P1.mumble_sth/* P1.*show_sth*; [ca. 1 sek.]	„(~)“	Direkt danach markiert er einen Teilstring mit der Maus.[32]
9	*P2.ask_knowledge* [ca. 5 sek.]	„Was mich nur wundert, warum wir jetzt immer wieder was kriegen. (.) Weil wenn wir ein Refresh machen, dann ist doch das (.) `MemCache`-Objekt gesetzt?“	Mit seiner unmittelbar nachfolgenden Äußerung bezieht sich der Observer auf die Beobachtung, dass der Driver das Skript nicht nur einmal sondern mehrere Male aufgerufen hat (mehrere Klicks auf die „Reload“-Schaltfläche), sich die im Browser dargestellte Seite aber nur beim ersten Aufruf verändert hat. Seiner Erwartung nach dürften bei den weiteren Aufrufen wohl keine Datenbankeinträge zurückgegeben werden.
10	*P2.explain_finding*; [ca. 3 sek.]	„Ach so, weil wir, weil wir die Zeit nicht ändern.‿Weil `last_change` sich nicht ändert.“	Nach ca. 2 Sekunden, in denen nichts passiert, beantwortet *P2* sich die Frage selbst. Vorher und währenddessen sieht *P1* gebannt auf den Bildschirm
11	*P2.disagree_finding*; [ca. 3 sek.]	„Ne. Moment. Weil (!...!)“	Ca. drei Sekunden später zieht *P2* seine Erklärung wieder zurück. Kurz vor der Äußerung klickt *P1* noch einmal auf die „Reload“-Schaltfläche.
			Fortsetzung auf der nächsten Seite.

[31] Damit *P1* hier zu einer Erkenntnis kommen konnte, musste er wohl vorher das Ergebnis erkundet haben. Solche sehr kurzen Sondierungen wurden im Rahmen der Herleitung der BS/BKM – wie bereits auf S. 383 erläutert – nicht mittels *explore_sth* markiert.

[32] Es handelt sich um die Uhrzeit 15:54. Nähere Erläuterungen zu *show_sth* folgen in Abschnitt 8.1.7.

			Fortsetzung der Tabelle der vorherigen Seite.
12			Nachfolgend diskutiert das Paar weiter darüber, warum das Ergebnis die vorliegende Gestalt hat.

Tab. 8.5: Episode aus Sitzung PR1.1 (Start 15:19:36), die sich um ein *verify_sth* rankt. Sie beginnt, nachdem Änderungen in einem PHP-Skript vorgenommen und diskutiert worden sind. Diese Änderungen betrafen den Bedingungsteil einer `if`-Anweisung, durch den gesteuert wurde, in welchen Situationen das Skript Werte aus einer Datenbank zurück gibt (Chronologie der beiden folgenden Ereignisse: 1. Änderungszeitpunkt eines bestimmten Datenbankeintrages; 2. Zeitpunkts der letzten Nachfrage nach diesem Eintrag). Die Episode verdeutlicht, dass nicht immer kanonisch ersichtlich ist, wann ein `verify_sth`-Phänomen vom Typ Test als abgeschlossen betrachtet werden sollte. Im Beispiel wird der Übergang zu anderen HCI-Aktivitäten, einer impliziten Sondierung (siehe Zeile 7) und einem expliziten *show_sth* (siehe Zeile 8), als diesbezügliches Indiz gewertet. Allerdings könnte man auch derart argumentieren, dass das *show_sth* en passant durchgeführt wird. Schließlich äußert sich das Paar zu keinem Zeitpunkt eindeutig dazu, ob es das Ergebnis des Tests als einen Erfolg oder Misserfolg werten will. Vielmehr ist es so, dass die beiden Entwickler diese Frage nachfolgend nicht nur weiter diskutieren, sondern der Driver das Skript einige Sekunden später sogar noch ein weiteres Mal aufruft. Aus welchem Grund er diese Aktion durchführt, lässt sich nicht sicher ermitteln. Im Rahmen der hier dargestellten Kodierung wurde davon ausgegangen, dass es sich um eine Art Übersprunghandlung handelt, alternativ könnte dieses Reload aber auch als zum ursprünglichen *verify_sth* gehörend gewertet werden. Zur Verdeutlichung des zugrundeliegenden Kodierproblems stelle man sich vor, der Driver hätte keine Handlungen vom Typ *explain_finding* und *show_sth* durchgeführt. Dann wäre das Ende des *verify_sth* noch weniger einfach festzumachen, da keine das *verify_sth* begrenzende HCI/HEI-Handlung mehr vorliegen würde. Da es aber – wie bereits mehrfach angesprochen wurde – in erster Linie auf die Erkenntnisse beim Kodieren und weniger auf die Kodierung selbst ankommt, stellen Phänomene dieser Art kein grundsätzliches Kodierproblem dar.

#	v	d	Kodes [Dauer]	Äußerung	Kontext/Kommentar
1			*P2.propose_step*/ *P2.think aloud_activity*/ Start eines *P2.verify_sth* [ca. 1 sek.]	„(Dann woll'n wir mal.)“	Der Driver *P2* schlägt vor, die in Entwicklung befindliche Applikation zu testen (die Intention der Äußerung wird nur aus dem Kontext ersichtlich). Während er spricht, startet er die Applikation, indem er auf die *Run*-Schaltfläche der IDE Eclipse klickt.
2			*P2.do_sth*[33]/ *P2.ask_knowledge*[34]/ *P2.think aloud_activity* [ca. 2 sek.]	„Na, (.) unser <*Applikationsname*>, (.) warum kommen wir nicht dran?“	Direkt nach dem Start der Applikation fährt *P2* mit der Maus zum unterem Bildschirmrand (Dauer: ungefähr 1 sek.). Allem Anschein nach will er auf die Windows-Taskbar zugreifen, die momentan nicht sichtbar ist.[35] Ziel ist es (wird erst aus nachfolgenden Äußerungen ersichtlich) nachzusehen, ob schon eine Instanz der Applikation läuft. Die Taskbar erscheint aber nicht. Daraufhin äußert er seine Frage. Diese Handlung wurde im Rahmen der Herleitung als eigenständig interpretiert und mit *do_sth* annotiert. Sie unterbricht das *verify_sth* nicht, sondern läuft parallel zu diesem.
3			*P1.explain_finding* [ca. 3 sek.]	„(~) Der muss 'n *Rebuild* machen - also der *buildet* gerad'.“	*P1* fällt seinem Partner mit einer Erklärung fast ins Wort (Details wurden bereits auf S. 270 erläutert). Währenddessen erscheint das Startfenster der gerade gestarteten Applikation mittig im Vordergrund (ähnlich wie in Abbildung 8.2 angeordnet) und zeigt an, dass der Startvorgang läuft.
4			*P1.disagree_finding* [ca. 1 sek.]	„(~Das‿hat‿damit‿nichts‿zu‿tun), dass die Taskleiste nicht geht.“	*P2* widerspricht der Äußerung des Partners unmittelbar nachfolgend (Details wurden bereits auf S. 270 erläutert). Parallel öffnet er die Konsole der IDE Eclipse.
					Fortsetzung auf der nächsten Seite.

[33] Das Konzept *show_sth* stellt eine Art Sammelbecken für unterschiedlichste Formen des „Zeigens“ dar und wird in Abschnitt 8.1.8 erläutert.

[34] Letztendlich muss an dieser Stelle offen bleiben, ob es sich wirklich um ein *ask_knowledge* und nicht um eine rhetorische Frage handelt, die mit einem *explain_finding* zu annotieren wäre (siehe auch S. 319).

[35] Die Microsoft Windows Taskbar kann bei vielen Windows-Systemen so eingestellt werden, dass sie nur dann erscheint, wenn sich die Maus am unteren Bildschirmrand befindet (siehe z.B. http://windows.microsoft.com/de-DE/windows7/the-taskbar-overview (Abruf: 10.01.2012)).

				Fortsetzung der Tabelle der vorherigen Seite.
5		*P1.ask_knowledge* [ca. 1 sek.]	„Welche Taskleiste?“	*P1* hatte allem Anschein nach gar nicht verstanden, worum es seinem Partner ging. Jetzt fragt er nach.
6		*P2.mumble_sth* [ca. 1 sek.]	„(~Hmm. Ach.)“	Die Intention dieser unmittelbar nachfolgenden Äußerung konnte nicht ermittelt werden.
7		*P1.mumble_sth* [ca. 1 sek.]	„(~Welche Ta (!!...!!))“	Evtl. wollte *P1* seine Frage wiederholen. Er wird aber durch einen Ausruf des Partners unterbrochen.
8		*P2.mumble_sth* [ca. 1 sek.]	„(~)“	Der Driver fällt seinem Partner mit einem unverständlichen Ausruf ins Wort.
9		*P1.ask_knowledge* [ca. 1 sek.]	„Is', is' unten?“	Unmittelbar nachfolgend stellt der Observer eine Frage. Er bezieht sich wohl auf die Taskleiste.
10		*P2.explain_knowledge* [ca. 2 sek.]	„Ne, ich hab' keine. (.) Siehst' doch.“	Auch *P2* reagiert unmittelbar. Seine Antwort stellt in seinen Augen sicherlich nicht die Formulierung einer neue Erkenntnis dar.
11		*P1.explain_knowledge*/ *P1.think aloud_activity*/ Start eines *P1.do_sth*/ *P1.become_driver*[36] [ca. 2 sek.]	„Bei mir kam sie aber gerad' eben.“	Wieder fällt *P1* seinem Partner fast ins Wort. Dabei greift er nach der Maus, die sein Partner noch in der Hand hält.
12		Ende des *P1.do_sth* [ca. 3 sek.]		*P1* fährt mit der Maus zum unterem Bildschirmrand. Die Taskbar erscheint aber nicht.
				Fortsetzung auf der nächsten Seite.

[36] Das Konzept *become_driver* wird in Abschnitt 8.2 eingeführt.

				Fortsetzung der Tabelle der vorherigen Seite.
13		*P2.explain_finding*|| *P1.mumble_sth* [ca. 2 sek.]	„<*P2:*> Wir haben noch ein <*Applikationsname*> am Laufen. Wir haben noch ein <*Applikationsname*> am Laufen."|| „<*P2:*> (~~)"	*P2* äußert die Erkenntnis, dass noch eine weitere Instanz der Applikation läuft. Es konnte nicht ermittelt werden, wie er zu dieser gekommen ist. Eine parallel vorgenommene Äußerung des Partners ist unverständlich. Währenddessen schließt der Startvorgang der kurz zuvor gestarteten Instanz ab. Die Applikation kann nun getestet werden.
14		Start eines *P1.verify_sth* [ca. 6 sek.]		Der Driver *P1* fährt mit der Maus zu der zu testenden Applikation und ruft dort Funktionalität auf.[37]
15		*P1.explain_finding*/ *P1.think aloud_activity*/ Ende des *P1.verify_sth* [ca. 1 sek.]	„O.K."	*P1* hält fest, dass er erkannt hat, dass die Applikation korrekt funktioniert.
16		*P2.agree_finding*/ *P2.think aloud_activity*/ Ende des *P2.verify_sth* [ca. 1 sek.]	„Gut"	*P2* stimmt unmittelbar nachfolgend zu.
17		*P1.propose_step* [ca. 1 sek.]	„Jetzt können wir einchecken."	*P1* stellt fest, wie fortgefahren werden sollte.

Tab. 8.6: Episode aus Sitzung PR2.1 (12:25:39–12:26:23), in der *P2*, während er eigentlich mit einem Test – also einem *verify_sth* – beschäftigt ist, zeitweise einer anderen HCI-Aktivität nachgeht – und zwar ohne die Testtätigkeit wirklich zu unterbrechen (oder abzubrechen). Die Episode demonstriert, dass Entwickler durchaus zwei verschiedenen HCI/HEI-Handlungen parallel nachgehen können, z.B. dann, wenn sie, wie hier dargestellt, Wartezeiten überbrücken wollen. Die Episode enthält darüber hinaus ein Beispiel für den Ablauf eines Driverwechsels. Details zum Mithandeln von Observern im Rahmen von HCI/HEI-Aktivitäten werden in Abschnitt 8.1.9 erläutert. Die Episode beginnt, nachdem Änderungen am Kode vorgenommen wurden. Die IDE Eclipse ist über die ganze Episode hinweg bildschirmfüllend zu sehen (allerdings nicht immer im Fokus). Legende: In der zweiten Spalte (v) ist der Verlauf von Handlungen vom Typ *verify_sth* visualisiert, in der dritten (d) der von Handlungen vom Typ *do_sth*. Handlungen von *P2* sind hellgrau, solche von *P1* mittelgrau (siehe Zeile 11 und 12) markiert. Wenn beide Entwickler gleichzeitig derselben Handlung nachgehen, so ist dies dunkelgrau markiert (siehe Zeile 14 und 15).

[37] Da die Bildschirmaufzeichnung zu diesem Zeitpunkt nicht flüssig gelaufen ist, kann nicht ausgeschlossen werden, dass *P2* mit dieser Aktion schon kurz zuvor während der Erkenntnisäußerung des Partners begonnen hat.

einer Handlungsentität verfolgt wird, nämlich ein Verifizieren oder Validieren. Es wird nicht festgehalten, welche Mittel dabei eingesetzt werden, z.B. Durchsichten oder Tests. Ähnlich verhält es sich bei allen anderen bisher erläuterten HHI/HEI-Konzepten. Ein *write_sth* adressiert wie gesehen die Umsetzung des Ziels, Kode oder andere Steuerungsbefehle oder -daten zu erzeugen oder zu verändern. Wie dies geschieht, z.B. durch das Tippen von Zeichen, das Kopieren von Elementen oder das Verwenden von Wizards, bleibt außen vor.

8.1.5 Das Konzept *read_sth*

Mit dem HCI/HEI-Konzept *read_sth* werden Teile von Handlungsentitäten kodiert, in denen es dem Handelnden primär darum geht, Inhalte oder Bruchstücke von Inhalten von Artefakten vorzulesen. Für die Verwendung des Konzeptes ist es unerheblich, ob es sich um physisch, also in der Regel in Papierform vorliegende Artefakte, oder nicht physisch vorliegende, in der Regel auf dem Bildschirm dargestellte, handelt. Beispielsweise handelt es sich beim Vorlesen eines in einem Papierdokument stehenden URL, um ein *read_sth*. Ein Vorlesen, welches nur en passant im Rahmen einer anderen HCI/HEI-Aktivität durchgeführt wird bzw. nicht als eigenständig betrachtet werden kann (z.B. ein partielles Mitlesen von Kode im Rahmen eines *verify_sth*) ist hingegen nicht mit *read_sth* zu annotieren.

Aus HHI-Sicht handelt es sich bei *read_sth*-Verbalisierenden oft um *explain_finding*-Äußerungen vom Typ *D* (siehe S. 254).

8.1.6 Das Konzept *sketch_sth*

Das Anfertigen von Skizzen, sei es auf Papier, auf sonstigen physikalischen Medien (z.B. Whiteboards) oder unter Zuhilfenahme von Softwarewerkzeugen am Rechner, wird durch das Konzept *sketch_sth* adressiert, wobei in den hier analysierten Sitzungen nur das Erstellen von Skizzen auf Papier beobachtet werden konnte. Die beiden folgenden Ziele von *sketch_sth*-Handlungen konnten beobachtet werden:

- Dem Partner etwas erläutern: Unterstützung bei der Erläuterung von Sachverhalten oder Problemen.
- Sich selbst etwas erläutern: Verbesserung des eigenen Verständnisses von Sachverhalten oder Problemen.

8.1.7 Das Konzept *show_sth*

Das Zeigen auf bestimmte Teile physischer oder nicht physischer Artefakte, sei es mit der Maus, durch so genanntes *Highlighting* oder mit dem Finger ist im Rahmen der Verwendung der BS/BKM mit dem Konzept *show_sth* zu annotieren. Beispielsweise zeigt *P1* in Sitzung PR2.1, während er ein von ihm im Vorfeld der Sitzung editiertes und nun im Editor sichtbares Interface mit den Worten

> „Also was ich (.) gemacht hab' mit Absprache von <**Entwicklername**>, dass ist dieses `IFeatureAttributeConfiguration` (..) erweitert um 'n `IVirtualColumn`."

erläutert, mit der Maus zuerst auf den Namen des Interfaces (`IFeatureAttributeConfiguration`) und dann auf eines der Attribute (`IVirtualColumn`).[38] Die beiden Zeigeoperationen wurden, da sie eine kontinuierliche Handlung darstellen (*P1* zeigt, was er gemacht hat), zusammen mit einem *show_sth* kodiert.

Im Rahmen der Herleitung der BS/BKM wurde das Konzept *show_sth* allerdings nur dann vergeben, wenn das Zeigen eine eigenständige HCI/HEI-Handlung darstellte. Derartige Handlungen fanden oft parallel zu Äußerungen vom Typ *knowledge* statt oder zu Vorschlägen, insbesondere solchen, in denen Referenzen auf sichtbare[39] Objekte verwendet werden. Dies war z.B. in den folgenden Situationen der Fall:

- *show_sth* im Rahmen eines *propose_design*: „<*PR2.1 *P1* (Driver):*> So wie das momentan ist, können wir den hier <**P1 highlightet* eine Zeile im sichtbaren Kode*> vernachlässigen."

- *show_sth* im Rahmen von *propose_step* und *explain_knowledge*: „<*PR2.1 *P1* (Observer):*> *Pin* doch über Popup <**P1* zeigt mit einem Finger auf eine Stelle auf dem Bildschirm (auf dem nur die IDE Eclipse zu sehen ist)*>. (.) Sonst ist da ein *Pin* <**P1* zeigt noch einmal mit einem Finger auf eine Stelle auf dem Bildschirm (auf dem nur die IDE Eclipse zu sehen ist)*>."

Trat das Zeigen hingegen im Rahmen einer Handlung eines anderen Handlungstyps (derselben Person) auf, z.B. als unterstützende Aktion bei einem *explore_sth*, wurde im Rahmen der Herleitung der BS/BKM auf eine Annotation von *show_sth* verzichtet. Weiterführende Untersuchungen müssen evtl. anders vorgehen, um relevante Ereignisse nicht zu verpassen. Es kann aber notwendig werden, *show_sth*-Annotationen parallel zu anderen HCI/HEI-Annotationen vorzunehmen. Insbesondere kann ein solches Vorgehen vor dem Hintergrund der in Tabelle 8.5 diskutierten Probleme beim Bestimmen der Endpunkte von HCI/HEI-Aktivitäten sinnvoll sein.

8.1.8 Das Konzept *do_sth*

Das HCI/HEI-Konzept *do_sth* fungiert als eine Art „Sammelbecken" für alle HCI/HEI-Handlungen, die nicht von anderen HCI/HEI-Konzepten adressiert werden bzw. nicht im Rahmen solcher stattfinden. *do_sth* stellt also eher einen

[38] Die Handlung folgt dem in Tabelle 7.10 in Zeile 2 erläuterten *propose_step*.

[39] Ein Objekt soll hier auch dann als sichtbar gelten, wenn es erst in Ansicht gebracht wird.

Marker als ein Konzept dar. Die notwendige Ausarbeitung wird explizit weiterführenden Untersuchungen überlassen. Dies betrifft beispielsweise folgende im Rahmen der Herleitung der BS/BKM beobachtete *do_sth*-Phänomene:

- Starten der IDE Eclipse.
- In Ansicht bringen einer neuen/anderen Klassendatei im Editor der IDE Eclipse.
- Aufrufen von Operationen des Versionsverwaltungssystems Subversion.[40]
- Aufruf und Verwendung eines *Issue-Tracking-System*s.
- Konfigurieren der IDE Eclipse.
- Umstrukturieren der Oberfläche der IDE Eclipse (beispielsweise vergrößern bestimmter Teilfenster wie das des *Problems-Views*).
- Aufrufen von Hilfsprogrammen wie z.B. cURL oder anderen Befehlen in einer Konsole.[41]
- Schließen nicht mehr benötigter Fenster (Aufräumen des Desktops).
- Löschen von Dateien (außerhalb der IDE).
- Starten oder Stoppen des Applikationsservers JBoss[42].
- Aufrufen von *Build*-Skripten.[43]

Im Rahmen der Herleitung der BS/BKM wurden unterschiedliche Ausprägungen von *do_sth*-Handlungen, die von einer Person zusammen bzw. hintereinander im Rahmen einer Handlungsentität durchgeführt wurden, zusammen mit einem *do_sth* annotiert. Weiterführende Untersuchungen müssen im Rahmen der notwendig werdenden Aufsplittung von *do_sth* evtl. anders vorgehen. Auch wurde ein Mithandeln des Partners nicht durch Kodierungen expliziert (siehe hierzu auch die Erörterungen im nachfolgenden Abschnitt 8.1.9).

8.1.9 Mithandeln des Partners

In vielen Fällen wird eine HCI/HEI-Aktivität von einem der Entwickler federführend durchgeführt. Oft handelt es sich hierbei um den Driver - z.B. dann, wenn

[40] Siehe Fußnote 66 auf Seite 191.
[41] Siehe Fußnote *a* auf S. 317.
[42] Siehe Fußnote 15 auf S. 87.
[43] Siehe Fußnote *a* auf S. 276.

die HCI/HEI-Aktivität physische Aktivitäten am Rechner erfordert (siehe Beispiel zu *write_sth* in Tabelle 8.2).[44] Dies bedeutet aber nicht, dass der Partner (und evtl. gleichzeitig auch Observer), welcher in vielen Situationen erst einmal an das Handeln seines Kollegen gebunden ist, nicht gleichzeitig dieselbe Aktivität durchführt, also z.B. während sein Kollege Kode durchsieht, dies nicht auch tut. Allerdings ist seine Aktivität oft nicht leicht als eine solche zu identifizieren, da sie in der Regel mit keinerlei (anhand der hier verwendeten Daten) erkennbaren physischen Aktivitäten einhergeht. An dieser Stelle muss der Forscher festlegen, welche eine solche Aktivität anzeigenden Indikatoren zugelassen werden sollen. Hierbei kann es sich beispielsweise um folgende handeln:

- Der Partner/Observer sieht in Richtung Monitor. Dieser Indikator ist allerdings höchst unzuverlässig. Denn weder ist – zumindest mit den hier propagierten Methoden – wirklich auszumachen, wo der Partner genau hinsieht, noch kann man aus der Blickrichtung ablesen, womit sich der Entwickler gerade beschäftigt.
- Der Partner hatte vorgeschlagen, diese Handlung durchzuführen, oder war zumindest am Erarbeiten des Vorschlags beteiligt.
- Der Partner verbalisiert Teile der Handlung, stellt handlungsspezifische Fragen oder gibt handlungsspezifische Antworten.

So ist beispielsweise der Observer *P2* an einem Vorschlag beteiligt, der zu dem in Tabelle 8.5 erläuterten Test (*verify*) führt. Außerdem antwortet er auf eine handlungsspezifische Frage, die der Driver im Laufe des Tests stellt, und sieht über die ganze Episode hinweg mit leicht vorgebeugtem Oberkörper in Richtung Bildschirm. Obwohl er keine physischen Tätigkeiten durchführt, scheint also eine Kodierung mit *P2.verify_sth* angebracht. Allerdings ist nicht offensichtlich, wann der Entwickler diese Tätigkeit beendet, bzw. zu welchem Zeitpunkt man davon sprechen möchte, dass er dies tut. Schließlich geht der Test mehr oder weniger fließend in eine Diskussion über. Im Rahmen der Herleitung der BS/BKM wurde hier der frühestmögliche plausible Zeitpunkt gewählt, nämlich derjenige, an dem auch das *P1.verify_sth* endet. Ein weiteres Beispiel ist in Tabelle 8.3 zu finden. Hier startet *P1* die im Zentrum der dargestellten Episode stehende Eclipse-Suche, eine Äußerung von *P2* führt aber dazu, dass *P1* den Suchvorgang abbricht und mit veränderten Parametern neu startet. Da *P2* darüber hinaus fast durchgehend in Richtung Monitor sieht, wurde die Episode auch als *P2.search_sth* kodiert.

Ohne weiter im Detail auf Mithandlungen eingehen zu wollen (dies soll weiterführenden Untersuchungen überlassen werden), kann folgendes festgehalten werden:

[44] Achtung: Dass ein Entwickler die Rolle des Drivers einnimmt, heißt keineswegs automatisch, dass er federführend agiert. So konnten Situationen beobachtet werden, in denen der Observer die Handlungen des Drives steuert, indem er ihm sagt, was er zu tun hat. Darüber hinaus ist bei weitem nicht in jeder Situation klar, welcher Entwickler die Rolle des Drivers und welcher die des Observers einnimmt.

- Um ein aktives oder passives Mithandeln kodieren zu können, muss in besonderer Weise nach Indizien gesucht werden. Aller Voraussicht nach wird es sich in der Regel um handlungstypspezifische Indizien handeln.
- Damit das Mithandeln konsistent erfasst werden kann, muss (sukzessive) ein Katalog dieser Indizien erstellt werden.
- Eine besondere Schwierigkeit besteht darin, Kriterien für den Anfang oder das Ende nichtphysischen Mithandelns festzulegen. Hier sollte im Auge behalten werden, dass es – wie schon mehrfach erläutert – nicht unbedingt auf eine hundertprozentig korrekte Kodierung ankommt. Evtl. gibt es eine solche sogar nicht (Stichwort: fließender Übergang). Man beachte in diesem Zusammenhang auch die Anmerkungen in Abschnitt 9.3.6 zum Umgang mit Start- und Endpunkten einzelner Kodierungen.

8.2 Ergänzende Konzepte

Die BKM enthält neben den HHI- und den HCI/HEI-Konzepten eine Reihe deskriptiver Kodes, mit denen weitere potentiell interessante Ereignisse festgehalten werden können. Sie wurden im Rahmen der Herleitung der BS/BKM nicht detailliert betrachtet. Im Einzelnen handelt es sich um die folgenden:

- *become_driver*: Dieser Kode adressiert diejenigen Zeitpunkte, zu denen ein Entwickler Maus und Tastatur übernimmt (siehe Beispiel in Tabelle 8.6). Ein teilweises Übernehmen, z.B. nur der Maus, wird im Rahmen der BS nicht adressiert.
- *work in parallel_sth*: Mit diesem Kode können Intervalle markiert werden, in denen die beiden Entwickler getrennt voneinander, in der Regel an unterschiedlichen Aufgabenteilen oder Problemen, arbeiten. Dieser Kode muss simultan als *P1.work in parallel_sth* und *P2.work in parallel_sth* annotiert werden. Alternativ kann festgelegt werden, nur *work in parallel_sth* zu verwenden.
- *work alone_sth*: Dieser Kode dient dazu, Intervalle zu markieren, in denen einer der beiden Entwickler den Arbeitsplatz verlassen hat, der andere aber trotzdem weiter arbeitet.
- *wait for_sth*: Mit diesem Kode können Phasen markiert werden, in denen ein Nutzer aktiv wartet, z.B. auf das Ergebnis einer Eclipse-Suche. Er wird in der Regel zusätzlich zu anderen HCI/HEI-Konzepten vergeben (siehe z.B. S. 378).
- *interrupt*: Dieser Kode erlaubt es, Arbeitsunterbrechungen festzuhalten, die durch äußere Einflüsse ausgelöst werden – z.B. durch die Frage eines Kollegen oder durch einen Telefonanruf. Dieser Kode entspricht nicht der für Basiskonzepte festgelegten Syntax.

Kapitel

9 Zentrale Aspekte beim Einsatz der BS/BKM

> Eins nach dem anderen, saubere, akkurate Schnitte dazwischen, bloß keine Verschmelzungen.
> Aber so ist es eben nicht.
> Christoph Schlingensief, Ich weiß, ich war's

Nachdem in den beiden vorangegangenen Kapiteln die einzelnen Konzepte bzw. Konzeptklassen weitgehend losgelöst voneinander erörtert worden sind, fokussiert sich das vorliegende Kapitel, nachdem noch einmal auf Zusammenhänge zwischen der BS und der Sprechakttheorie (Abschnitt 9.1) und auf die BS im Kontext der linguistischen Gesprächsanalyse (Abschnitt 9.2) eingegangen worden ist, auf den Umgang mit solchen „Phänomenen“, die weitgehend unabhängig von speziellen Konzepten oder Konzeptklassen auftreten (Abschnitt 9.3). Hierbei werden die Regeln der BS, die in Kapitel 7 und 8 vornehmlich im Zusammenhang einzelner Konzeptklassen erläutert wurden, noch einmal allgemeiner diskutiert – z.B. solche, die die Segmentierung der Daten oder Doppelkodierungen betreffen. Bei den Erörterungen dieser Regeln sollte im Hinterkopf behalten werden, dass es sich weniger um feststehende Gesetzmäßigkeiten, als mehr um Leitlinien handelt, die im Verlauf der Herleitung der BKM erarbeitet wurden, bzw. ohne die eine Strukturierung der beobachteten Phänomene nicht möglich gewesen wäre. So wird im Weiteren noch einmal darauf eingegangen, warum der Umgang mit diesen Regeln flexibel, das heißt auf das jeweilige Untersuchungsziel abgestimmt, erfolgen sollte. Ein weiterer Fokus der nachfolgenden Darstellungen ist darauf gerichtet, aufzuzeigen, welche „Phänomene“ nicht durch die BS/BKM erfasst werden können.

9.1 Die BS im Kontext der Sprechakttheorie

Im vorliegenden Abschnitt werden Parallelen zwischen der BS (bzw. der Ausrichtung der Basiskonzepte) und der Sprechakttheorie (nach Searle [189]) erörtert. Hierbei wird das Ziel verfolgt, deutlich zu machen, welche Aspekte einzelner Äußerungen in welcher Weise von der BKM adressiert werden. Um den Rahmen der vorliegenden Arbeit nicht zu sprengen, muss allerdings weit hinter dem zurückgeblieben werden, was eine ausführliche Erörterung der BS vor dem Hintergrund der Sprechakttheorie möglicherweise zu leisten vermag.

Achtung: Wie bereits angesprochen haben die hier dargestellten Überlegungen zur Sprechakttheorie den Analyseprozess nicht beeinflusst, da sie erst im Anschluss an die Herleitung der BS vorgenommen wurden.

9.1.1 Die HHI-Verben im Kontext der Sprechakttheorie

Wie schon mehrfach erläutert (siehe z.B. S. 143) wird mittels der meisten HHI-Verben auf eine gewisse Weise die (primäre) Illokution einer Sprechhandlung[1] adressiert – beispielsweise referenziert das BKM-Verb *ask* die Illokution „Frage“.[2] Allerdings existiert nicht zu jeder denkbaren (primären) Illokution ein HHI-Verb, welches genau diese explizit adressiert. Vielmehr ist es noch nicht einmal so, dass die BKM für alle bisher im Kontext von Paarprogrammierungssitzungen beobachteten Illokutionen ein eigenes HHI-Verb zur Verfügung stellt. Schließlich beschränkt sich die BKM auf eine überschaubare Menge von HHI-Konzepten. Zusammenfassend gilt hier:

1. Dem Grundprinzip der GTM folgend werden nicht alle irgendwie denkbaren Illokutionen in allen Zusammenhängen/Situationen von der BS/BKM adressiert, sondern nur diejenigen, die tatsächlich beobachtet wurden. Außerdem werden Illokutionen, die sich nicht im herausgearbeiteten Fokus der BS befinden, nicht explizit adressiert, sondern sind in der Regel unter *explain_knowledge* (oder anderen *knowledge*-Konzepten) einzusortieren (siehe z.B. S. 316). In diesem Zusammenhang gilt es zu beachten, dass die HHI-Konzepte der BS/BKM hauptsächlich *repräsentative* und *direktive* Sprechhandlungen adressieren. Diese Begrifflichkeiten gehen auf Searle zurück und adressieren Sprechhandlungen, „mit denen ein Wahrheitsanspruch erhoben wird“ bzw. Sprechhandlungen, mit denen „Forderungen an den Hörer gerichtet [werden]“ [95, S. 238].

2. In der Regel werden unter einem HHI-Verb eine ganze Reihe von Illokutionen subsumiert. Beispielsweise sind in Tabelle 7.5 die unterschiedlichen Arten von Äußerungen erläutert, die durch *propose_design* (bzw. *propose*) adressiert werden. Aus der Tabelle wird ersichtlich, dass vom Verb *propose* primäre Illokutionen wie z.B. Vorschlagen, Auffordern oder Bitten erfasst werden. Allerdings adressiert das Verb *propose* nicht beliebige Äußerungen, in denen solche Sprechhandlungsaspekte ausgemacht werden können, sondern nur solche, in denen z.B. ein Designaspekt oder ein Arbeitsschritt, also z.B. etwas vom Typ *design* oder *step*, neu ins Spiel gebracht wird. Die Auswahl eines geeigneten HCI-Verbs hängt also in einem gewissen Maße auch vom zu verwendenden Objekt und nicht nur von der identifizierten Illokution ab. Weitere Beispiele hierzu sind in Abbildung 9.1 zu finden.

[1] An dieser Stelle sei darauf hingewiesen, dass es keine Eins-zu-eins-Beziehung zwischen Satz und Sprechhandlung gibt. Siehe hierzu z.B. [95, S. 237].

[2] Zur Erinnerung: Nicht jede in Frageform getätigte Äußerung ist im Rahmen der BS als Frage, also über das Verb *ask*, zu klassifizieren. Beispielsweise kann es sein, dass eine Frage zum Design, die eine eigene Designidee enthält als Vorschlag gewertet und somit mit *propose_design* annotiert werden muss (siehe Beispiel 2 in Tabelle 7.4).

Bei der Bestimmung der Illokution kann es helfen, die im Fokus stehende Äußerung zu paraphrasieren. Als Beispiel betrachte man folgende, bereits auf S. 333 angesprochene Äußerung aus Sitzung PR1.1:

> „Speichern wir beide Werte oder speichern wir (!...!). (..) Oder reicht uns `last_change`? (.) Brauchen wir eigentlich `last_request`?“

Es stellt sich die Frage, ob es sich hier um einen Vorschlag zum Design (also ein *propose_design*) oder um eine Frage (also ein *ask_knowledge*) handelt – oder gar um beides, nämlich eine Frage gefolgt von einem Vorschlag? Hierzu sollte man den Kontext näher betrachten – hier unter anderem die Tatsache, dass es um den zuvor ins Spiel gebrachten Vorschlag geht, den Wert `last_request` nicht an das Skript übergeben zu lassen, sondern lokal zu speichern. Nachfolgend sollte dann versucht werden, zu einer passenden/angemessenen Paraphrasierung der Äußerung zu kommen – z.B.:

> „Ich verstehe nicht, warum wir `last_change` *und* `last_request` auf unserer Seite speichern sollten.“

Mittels dieser Form der Äußerung lässt sich erkennen, dass eine Kodierung mit *ask_knowledge* angebracht ist und nicht etwa eine mit *propose_design* oder beiden Konzepten. Allerdings sollte im Auge behalten werden, dass es (im Kontext der BS) nicht in jedem Fall möglich (oder erwünscht) ist, nur die (primäre) Illokution einer Äußerung zu erfassen.[3] So gibt es Situationen, in denen es notwendig bzw. in Hinsicht auf das Erreichen des Analyseziels zumindest hilfreich erscheint, unter Verwendung von Basiskonzepten weitere Phänomene festzuhalten, also Doppelkodierungen vorzunehmen. Hierzu zählen z.B. Wissensäußerungen vom Typ *explain_knowledge*, mit denen der Sprecher in gewisser Weise etwas zuvor Geäußertes abgelehnt. Im Abschnitt 9.3.1 wird hierauf genauer eingegangen.

9.1.2 Die HHI-Objekte im Kontext der Sprechakttheorie

Betrachtet man Beispiele wie

> „Sondern wir müssen das `FeatureProxy` benutzen, um die Werte abzuholen“ (siehe auch S. 158)

kann man erkennen, dass entgegen der ersten Intention bei der Verwendung von HHI-Objekten in vielen Fällen nicht ausschließlich in kategorisierender Weise Bezug auf den propositionalen Gehalt bzw. den propositionalen Akt (siehe

[3] Der Beantwortung der Frage, ob bzw. inwieweit man überhaupt pauschal von *der eindeutigen* (primären) Illokution von Äußerungen sprechen sollte, kann im Rahmen der vorliegenden Arbeit nicht nachgegangen werden.

Exkurs 13) einer Äußerung genommen wird. Schließlich spielt bei der Konzeptualisierung einer Äußerung A_0 als eine vom Typ *design* auch ihre Illokution oder zumindest die Illokution einer Äußerung, auf die sich A_0 bezieht, eine Rolle. Denn die BS erlaubt die Verwendung des Terminus *design* nur dann, wenn ein Designaspekt in Form von Vorschlägen oder Vorschlagsmodifikationen geäußert bzw. referenziert wird, oder wenn Vorschläge bzw. Vorschlagsmodifikationen, in denen es um einen geäußerten Designaspekt geht, bewertet werden.[4] Der Begriff *design* adressiert also keineswegs direkt etwas, was man als Klassifizierung des propositionalen Gehaltes einer Äußerung bezeichnen könnte.[5] Ein weiteres Beispiel für den Einfluss der Illokution auf die Wahl eines passenden Objektes ist in Abbildung 9.1 zu finden.

9.1.3 Die BS/BKM und perlokutionäre Akte

Durch die BS/BKM werden perlokutionäre Akte (siehe Exkurs 13) nicht – zumindest nicht zielgerichtet – erfasst. Hierzu betrachte man beispielsweise das BKM-Verb *agree*. Mit ihm werden Reaktionen in Form von Zustimmungen markiert. Der perlokutionäre Akt, der sich hierbei widerspiegelt, z.B. ein Überzeugen oder Überreden, wird aber nicht festgehalten.[6]

[4] Von *ask_design* sei hier einmal abgesehen.

[5] Es sei in diesem Zusammenhang noch einmal darauf hingewiesen, dass bei der Erläuterung des Objektes *design* nicht von Design, sondern von Designoptionen gesprochen wird (siehe z.B. Tabelle 7.1). Analoges gilt zumindest auch für die *P&P*-Klassen *step* und *strategy* sowie mit Einschränkungen auch für die Klasse *requirement*.

[6] Man beachte hier auch Fußnote *c* in Exkurs 13

Exkurs 13: *Sprechakttheorie: Propositionaler und perlokutionärer Akt*

Im Exkurs 6 wurde in die Sprechaktheorie eingeführt und näher auf den illokutionären Akt eingegangen. Hierbei wurde sowohl der propositionale wie auch der perlokutionäre Akt erwähnt, allerdings ohne sie zu erläutern. Dies soll nun nachgeholt werden.

Der Begriff *propositionaler Akt* geht auf Searle zurück. Searle unterscheidet „zwischen dem illokutionären Akt und dem propositionalen Gehalt des illokutionären Aktes“ [189, S. 49], wobei man unter dem propositionalen Gehalt vereinfacht ausgedrückt den eigentlichen Inhalt („die jeweilige Wirklichkeit“ [226, S. 26]) einer Äußerung verstehen kann. Searle bezeichnet das zum Ausdruck bringen einer Proposition bzw. eines propositionalen Gehaltes[a] als propositionalen Akt [189, S. 48].[b] Er weist darauf hin, dass es zwei propositionale Akte gibt: „Das Referieren auf und das Prädizieren über eine außersprachliche Entität“ [27]. Mit dem *Referenzakt* „*verweist* [ein] Sprecher *auf* ein bestimmtes Objekt [...] oder erwähnt und bezeichnet es“ [189, S. 39], mit dem *Prädikationsakt* prädiziert er es, weist ihm also eine Eigenschaft zu. Dabei sollte beachtet werden, dass die charakteristische grammatische Form des propositionalen Aktes Teile von Sätzen sind: „beim Akt der Prädikation grammatische Prädikate, beim Akt der Referenz Eigennamen, Pronomen und bestimmte andere Arten von Nominalausdrücken“ [189, S. 42]. Mit verschiedenen Äußerungen, insbesondere Äußerungen, mit denen unterschiedliche illokutionäre Akte vollzogen werden, können dieselben Referenz- und Prädikationsakte einhergehen. Als Beispiele verweist Searle [189, S. 39] u.a. auf die Äußerungen „Sam raucht gewohnheitsmäßig“ und „Raucht Sam gewohnheitsmäßig?“ In beiden Fällen referenziert der Sprecher das Objekt „Sam“ und prädiziert es als „raucht gewohnheitsmäßig“. Im ersten Fall handelt es sich aber um eine Behauptung, im zweiten um eine Frage. Achtung [189, S. 49]: „Natürlich haben nicht alle illokutionären Akte einen propositionalen Gehalt – Äußerungen wie z.B. »Hurra!« oder »Au!« haben keinen.“

Das Konzept des *perlokutionären Aktes* stammt von Austin [7] und wurde von Searle übernommen [189, S. 42]: „Eng verbunden mit dem Begriff der illokutionären Akte sind die Konsequenzen oder Wirkungen, die solche Akte auf die Handlungen, Gedanken, Anschauungen usw. der Zuhörer haben. Zum Beispiel kann ich jemanden durch Argumentieren *überreden* oder *überzeugen*, durch Warnen *erschrecken* oder *alarmieren*, [...] durch Informieren *überzeugen* (*aufklären*, *belehren*, [...] *dazu bringen, etwas zu begreifen*). Die in dieser Aufzählung kursiv gedruckten Ausdrücke bezeichnen perlokutionäre Akte.“ Perlokutionäre Akte erzielt man also eher, als dass man sie vollzieht.[c] Wunderlich [226, S. 40] weist darauf hin, dass „perlokative Sprechhandlungen, d.h. das Zustandebringen von Wirkungen dieser Art beim Zuhörer vom Agenten niemals vollständig kontrollierbar [sind]“.

[a] Die Begriffe Proposition und propositionaler Gehalt werden oft nicht gleichbedeutend verwendet. Hierauf kann in der vorliegenden Arbeit aber nicht eingegangen werden.

[b] Genau genommen sagt Searle [189, S. 48f]: „Aussagen und Behaupten sind Akte, Propositionen dagegen nicht. Eine Proposition ist etwas, das im Akt des Behauptens behauptet, in dem des Aussagens ausgesagt wird. [...] Der Ausdruck einer Proposition ist ein propositionaler Akt, kein illokutionärer. Und [...] propositionale Akte [können] nicht selbstständig vorkommen. Man kann nicht einfach eine Proposition ausdrücken und, ohne gleichzeitig noch etwas anderes zu tun, damit einen vollständigen Sprechakt vollziehen.“

[c] Nach Austin [7] ist eigentlich der perlokutionäre Akt vom perlokutionären Effekt zu unterscheiden.

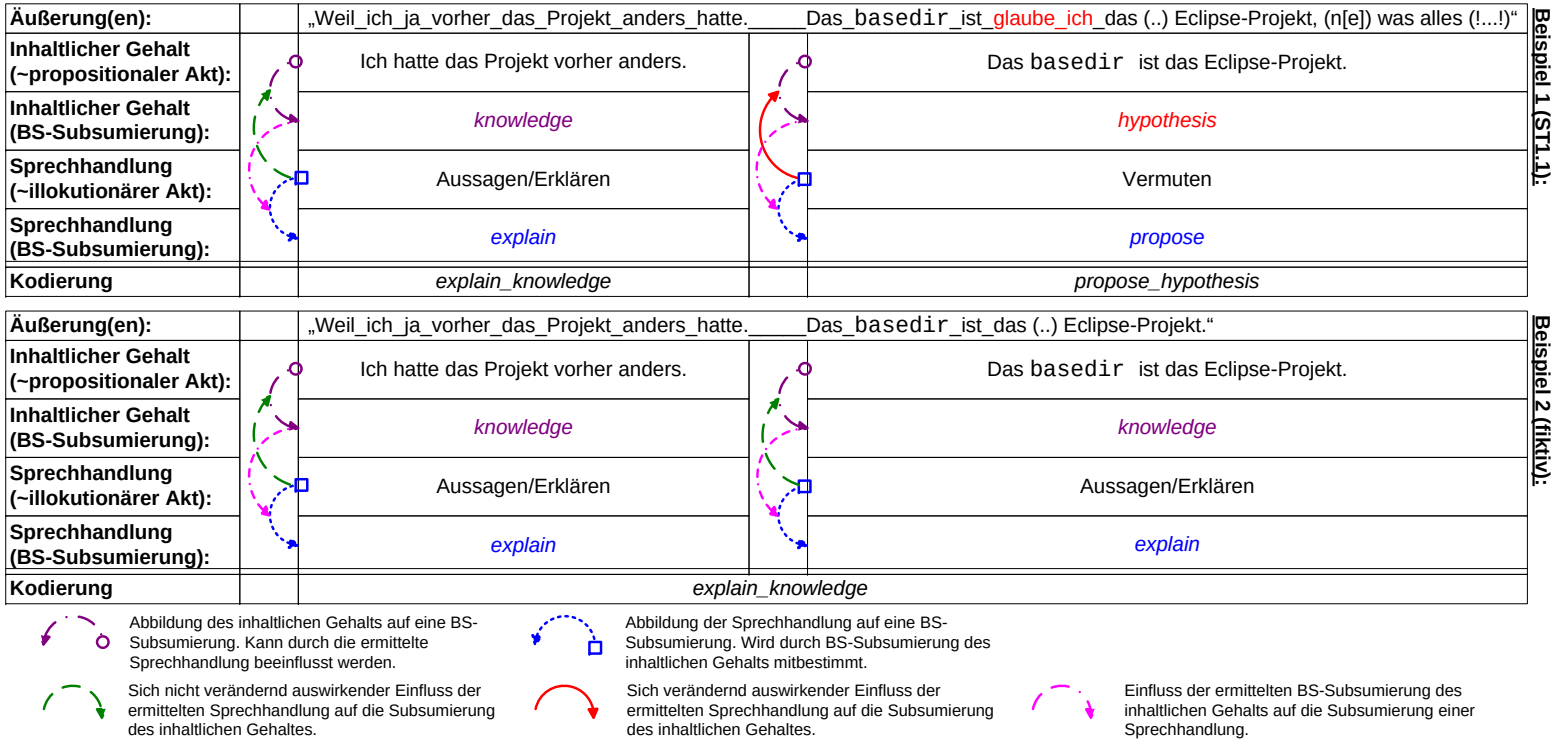


Abb. 9.1: Darstellung des Einflusses des propositionalen und illokutiven Aktes von Äußerungen auf ihre BS-Kodierung anhand von zwei Beispielen. Beispiel 1 stammt aus Sitzung ST1.1 (siehe auch Tabelle 7.37). Beispiel 2 ist fiktiv. Es wurde aus Beispiel 1 gewonnen. In Beispiel 1 ist zu erkennen, das der inhaltliche Gehalt des zweiten Teils der Äußerung nicht unter *knowledge*, sondern unter *hypothesis* zu subsumieren ist, da im Rahmen der BS/BKM die ermittelte Sprechhandlung die Subsumierung des inhaltlichen Gehaltes mitbestimmt. Ferner ist zu erkennen, dass die somit erhaltene Subsumierung des inhaltlichen Gehaltes Einfluss auf die Subsumierung der Sprechhandlung hat. Sie muss im Falle von *hypothesis* unter *propose* subsumiert werden. Nicht dargestellt ist der Einfluss des Kontextes, insbesondere die Konsequenzen daraus, dass zwischen initiativen oder reaktiven (Teil-)Äußerungen unterschieden werden muss. Darüber hinaus kann man in Beispiel 2 erkennen, dass inhaltlich zusammengehörige (Teil-)Äußerungen zusammengefasst und zusammen kodiert werden, wenn jeder einzelnen ohnehin das gleiche Konzept zugewiesen werden würde (siehe auch S. 320).

9.2 Die BS im Kontext der linguistischen Gesprächsanalyse

Nach Brinker und Sager [31, S. 19] sieht es die (linguistische) Gesprächsanalyse „als ihre zentrale Aufgabe an, die Bedingungen und Regeln systematisch zu erforschen, die die ‚natürliche' Gesprächskommunikation, d.h. dialogisches sprachliches Handeln in verschiedenen gesellschaftlichen Bereichen (Alltag, Institutionen, Medien usw.), bestimmen". Die Zielobjekte der Gesprächsanalyse sind also – zumindest in Teilen – mit denen der BS/BKM identisch. Somit ist es naheliegend, die BS mit der Gesprächsanalyse in Hinsicht auf den Umgang mit sprachlichen Einheiten und Strukturen zu vergleichen, um auf diese Weise die Regeln und (gewollten) Beschränkungen der BS/BKM aus einem neuen Blickwinkel beleuchten zu können. Um den Rahmen der vorliegenden Arbeit nicht zu sprengen, geschieht dies nur in drei Bereichen:

1. **Umgang mit Hörersignalen**: Unter einem *Gespräch* verstehen Brinker und Sager „eine begrenzte Folge von sprachlichen Äußerungen[7], die dialogisch ausgerichtet ist und eine thematische Orientierung aufweist" [31, S. 12].[8] Die Grundeinheit eines Gesprächs bildet der *Gesprächsschritt*. Darüber hinaus werden in der (linguistischen) Gesprächsanalyse auch komplexere Einheiten wie *Gesprächssequenzen* und *Gesprächsphasen* betrachtet. Sie „lassen sich als spezifische Abfolge bzw. Kombinationen von Gesprächsschritten" charakterisieren [31, S. 57] (Details zu Geprächsphasen folgen weiter unten).

   Wie schon in Fußnote 48 auf Seite 175 dargestellt wird ein Gesprächsschritt von Goffmann [80, S. 201] als „alles das, was ein Individuum tut und sagt, während es an der Reihe ist", definiert.[9] Brinker und Sager weisen allerdings darauf hin, dass sich Gesprächspartner „nicht nur dann, wenn sie ‚an der Reihe' sind", äußern, sondern z.B. auch in Form von so genannten *Hörersignalen*, „kurzen sprachlichen und nicht-sprachlichen Äußerungen des Hörers", mit denen „in erster Linie Aufmerksamkeit, eventuell noch Zustimmung oder Ablehnung" signalisiert wird [31, S. 58 f].[10] Die linguistische Gesprächsforschung unterscheidet also zwischen zwei verschiedenen Formen von Äußerungen, dem Gesprächsschritt und dem Hörersignal (sie werden unter dem Begriff des *Gesprächsbeitrages* zusammengefasst) [31, S. 58]. Im Rahmen der Erörterung der BS/BKM wird zwar auch kurz auf derartige Unterschiede eingegangen (siehe S.175), es werden aber keine Mittel zur Verfügung gestellt, um diese Differenzierung in den Analyseprozess

[7] Siehe Exkurs 4.

[8] Da die hier gegebene Definition zumindest weit verbreitet ist, sei dahingestellt, ob es – in der Linguistik oder in anderen Bereichen – wirklich ein einheitliches Verständnis darüber gibt, was genau als ein Gespräch gelten soll.

[9] Laut Brinker und Sager [31, S. 58].

[10] Siehe auch die Definition des Begriffs Back-channel auf S. 175.

einzubringen. Dies wird vielmehr explizit nachfolgenden Studien zu überlassen.

2. **Handlungsbedeutung vs. Gesprächsfunktion**: In der linguistischen Gesprächsanalyse wird laut Brinker und Sager [31, S. 64] zwischen zwei Aspekten eines Gesprächsschritts unterschieden: Auf der einen Seite gibt es die Handlungsbedeutung (im Sinne der Sprechakttheorie), die auch als *Basisfunktion* eines Gesprächsschritts bezeichnet wird, und auf der anderen Seite die Gesprächsfunktion, die „konkrete kontextuelle Bedeutung des Gesprächsschritts im Hinblick auf den ‚Stand der Kommunikation', d.h. vor dem Hintergrund der unmittelbar vorangegangenen Gesprächsbeiträge sowie bestimmter Bedingungen und Gegebenheiten der Gesprächssituation (insbesondere der Beziehungskonstellation). Zwischen der Basisfunktion und der Gesprächsfunktion eines Gesprächsschrittes besteht zwar ein Zusammenhang, keinesfalls aber eine 1:1-Beziehung". Als Beispiel betrachte man die zweite Äußerung aus der in Tabelle 7.48 dargestellten Episode:

   „Dann hast Du das umgeändert."

   Als Basisfunktion kann hier „Behauptung" ausgemacht werden. Bei der Gesprächsfunktion handelt es sich hingegen um eine „Disqualifikation des Partners" nach dem Motto: „Andere hätten ein solches Tastaturkürzel sicherlich nicht definiert." Dies kann unter Einbeziehung des Kontextes erschlossen werden. Schließlich lässt der Sprecher schon vorher durch kritische Anmerkungen und eine entsprechende Mimik und Körperhaltung erkennen, dass er dem Partner, der schon im Vorfeld am betroffenen Kode gearbeitet hatte, nicht das Feld überlassen will. Als Mittel dieses zu erreichen, betreibt er ein kontinuierliches Infragestellen der Kompetenz des Partners. Während die Basisfunktion von der BS/BKM durch ein *explain* subsumierend adressiert wird, wird die Gesprächsfunktion hier nicht erfasst. Das Ermöglichen diesbezüglicher Konzeptualisierungen überlässt die BS/BKM in weiten Teilen nachfolgenden Untersuchungen. Bei diesen Untersuchungen muss evtl. auch in Betracht gezogen werden, dass die Gesprächsfunktion die Basisfunktion dominieren kann (siehe hierzu auch Brinker und Sager [31, S. 65]).

3. **Gesprächsphasen**: In der linguistischen Gesprächsforschung werden unterschiedliche Gesprächsphasen (z.B. Eröffnungsphase, Kernphase und Beendigungsphase) – oft separat – betrachtet. Derartige Unterscheidungen werden von der BS/BKM nicht vorgenommen. Anders ausgedrückt: Die Elemente der BKM können unabhängig davon verwendet werden, in welcher Phase der Sitzung sich das Paar befindet – zumindest wenn man davon absieht, dass einige Konzepte eher dann zum Einsatz kommen können, wenn eine neue Phase startet (z.B. *propose_strategy*) und andere eher, wenn eine Phase endet (z.B. *propose_state*). Es ist davon auszugehen, dass weiterführende Untersuchungen diese Vereinfachung nicht beibehalten können.

Schließlich kann angenommen werden, dass Phänomene eines bestimmten BKM-Typs phasenabhängige Eigenschaften besitzen (man beachte auch den Kontext im paradigmatischen Modell).

9.3 Allgemeine Regeln für den Einsatz

Im Weiteren werden Kodierregeln, die in den vorangegangenen Kapiteln nur im Zusammenhang mit einzelnen Konzepten oder Konzeptklassen erläutert wurden, noch einmal weitgehend losgelöst von diesen erörtert. Hiermit wird das Ziel verfolgt, den Einstieg in die Verwendung der BS/BKM zu erleichtern.

9.3.1 Angemessene HHI-Konzepte auswählen

Wie bereits erläutert ist nicht für jede Äußerung unmittelbar ersichtlich, welches HHI-Konzept ihr zugewiesen werden muss oder ob überhaupt ein HHI-Konzept existiert, welches verwendet werden kann.[11] Dies hängt oft weniger mit der BS/BKM zusammen, als dass es eine Eigenschaft von Sprechhandlungen an und für sich ist. Helbig [95, S. 241] hält diesbezüglich fest, dass „die Frage, um welche [Sprechhandlung] es sich bei einer gegebenen Äußerung handelt, nicht immer eindeutig zu beantworten und [darüber hinaus] in der Regel [...] von der aktuellen Kommunikationssituation“ abhängig ist.

Um diesem Problem systematisch zu begegnen, sollte beim Kodieren wie folgt vorgegangen werden:

1. HHI-Kodierungen sind keine Konzeptualisierungen von einzelnen alleinstehenden Äußerungen, sondern Konzeptualisierungen der Bedeutungen von Äußerungen im Kontext einer PP-Session (siehe Exkurs 4 und Beispiel auf S. 201). Unter Berücksichtigung des Kontextes und Beachtung des grundlegenden Prinzips der Kodierung sprachlicher Bezüge (siehe S. 161) sollte versucht werden, zu einer eindeutigen Klassifizierung mit dem spezialisiertesten Konzept zu kommen. Hierbei sollte in der Regel (zumindest in gewisser Weise – siehe Abschnitt 9.1.1 und Abschnitt. 9.1.2) der primäre illokutionäre Akt erfasst werden. Ausnahmen hiervon wurden bereits im Zusammenhang mit den Erläuterungen der einzelnen HHI-Konzepte dargelegt – siehe z.B. S. 316. Im nächsten Punkt wird noch einmal auf diese eingegangen.

2. Sollte – zumindest allen Anschein nach – eine zu kodierende Äußerung durch die existierenden Elemente der BKM nicht adressierbar sein, das heißt kein Konzept existieren, mit dem die Äußerung in Hinsicht auf die eruierte, den

[11] Achtung: Hier geht es nicht darum, dass es vom Beobachter identifizierte sprachliche Phänomene in den Daten gibt, die von der BKM nicht adressiert werden. Solche gibt es natürlich. Es wird vielmehr das Problem angesprochen, dass es sinnvoll, aber evtl. nicht sofort ersichtlich sein kann, eine Äußerung mit mehreren HHI-Konzepten zu annotieren oder dass für eine Äußerung allem Anschein nach kein HHI-Konzept existiert, mit dem diese konzeptualisiert werden könnte.

aktuellen Kontext einbeziehende Illokution angemessen annotiert werden kann, muss wie folgt vorgegangen werden:

(a) Zuerst sollte (noch einmal) überprüft werden, ob die ermittelte Illokution nicht von einem Element der BKM subsumierend adressiert wird. Das heißt, es muss (falls dies nicht schon geschehen ist) der Frage nachgegangen werden, ob ein Konzept zur Verfügung steht, welches das identifizierte Phänomen in gewisser Weise mit adressiert. Falls dies zutrifft, sollte mit diesem Konzept gearbeitet werden. Details hierzu wurden bereits in Abschnitt 9.1.1 erörtert.

(b) Falls kein subsumierendes Konzept ermittelt werden kann, sollte überprüft werden, ob die Illokution und der Bezug der Äußerung, also das Objekt, nicht derart interpretiert werden können, dass die Äußerung durch eines der „Sammelbecken“ der BKM (adäquat) erfasst werden kann. Falls dies möglich ist, sollte mit dem entsprechenden „Sammelbecken“-Konzept gearbeitet werden. In der Regel wird es hierbei darauf hinauslaufen, dass ein *knowledge*-Konzept, z.B. *explain_knowledge*, eingesetzt wird (siehe das Beispiel auf S. 318, bei dem eine Verteidigung als *explain_knowledge* kodiert wird).[12]

(c) Erst wenn die beiden oberen Möglichkeiten ausgeschlossen wurden, sollte geklärt werden, ob ein bereits festgelegtes Verb und ein bereits festgelegtes Objekt zu einem neuen Konzept kombiniert werden können, so dass die Äußerung in adäquater Weise annotierbar wird (z.B. durch ein *disagree* und ein *todo*, was zu dem bisher nicht beschriebenen Konzept *disagree_todo* führen würde). Falls dies möglich ist, sollte ein solches Konzept verwendet werden.

(d) Wenn klar ist, dass keiner der drei beschriebenen Wege beschritten werden kann, sollte über die Einführung neuer Verben und Objekte nachgedacht werden.

3. Ist eine eindeutige Klassifizierung nicht möglich, müssen mehrere Konzepte annotiert werden. Dies kann in unterschiedlichen Situationen notwendig werden. So wurden z.B. im Rahmen der Herleitung der BS Dialoge identifizierte, in denen

   – Vorschläge in Wissensäußerungen des Partners oder umgekehrt
   – Wissensäußerungen in Vorschläge des Partners

   mündeten und die Reaktionen implizite Ablehnungen oder Zustimmungen enthielten. In Abbildung 9.2 sind sie zusammenfassend dargestellt. Da es sich bei den dargestellten Erwiderungen um indirekte Sprechakte handelte,

[12] Hier kann es sinnvoll sein, entsprechende Memos zu verfassen, um beispielsweise später die Beschreibung der BS zu ergänzen.

deren primärer wie auch sekundärer illokutionärer Akt (in weiteren Untersuchungen) von Interesse sein kann, wurden Doppelkodierungen vorgenommen. Weitere Gründe für Doppelkodierungen sind die Folgenden:

- Äußerungen, die sich derart paraphrasieren lassen, dass mehrere, unterschiedlich zu konzeptualisierende (Teil-)Äußerungen entstehen (siehe Beispiel auf S. 205 (*disagree_step* vs. *explain_knowledge*/*explain_finding*)).
- Abgrenzungsprobleme, die darauf zurückzuführen sind, dass sich, wie nachfolgend noch detaillierter erläutert werden wird, Phänomene aus der realen Welt nicht immer auf eindeutig unterschiedene Kategorien verteilen lassen. Als Beispiel betrachte man das *propose_step* in Tabelle 7.11. Hier ist nicht wirklich ersichtlich, ob man die Äußerung als *propose_step* oder *propose_strategy* lesen sollte. Es kann also sinnvoll sein, (vorerst) beide Konzepte zu annotieren.
- Designvorschläge, die im Rahmen von Strategievorschlägen gemacht werden (siehe Beispiel auf S. 219 bzw. Erörterung auf S. 231).
- Implizierte Handlungsankündigungen wie z.B. auf S. 204 beschrieben. Hierauf wird im Abschnitt 9.3.4 detaillierter eingegangen.

Man beachte, dass Doppelkodierungen evtl. auch die Doppelkodierung nachfolgender Äußerungen zur Folge haben können.

4. In den Punkten 1 bis 3 wurde davon ausgegangen, dass ausschließlich mit Elementen der BKM gearbeitet wird bzw. dass nur dann neue Konzepte einzuführen sind, wenn dies im Rahmen des Blickwinkels der BS/BKM (siehe Abschnitt 6.1) nötig ist. Dies ist aber keineswegs der Standard- sondern vielmehr eher der Ausnahmefall. In der Regel wird die BS/BKM „nur“ einen Ausgangspunkt für weiterführende Studien mit einem eigenen Blickwinkel bilden. Solche Untersuchungen werden neue und andersartige Regeln bez. des Umgangs mit Mehrfachkodierungen benötigen, z.B. dann, wenn sie spezielle Eigenschaften von einzelnen Konzepten betrachten oder wenn sie – um einen Spezialfall herauszuheben – den Fokus auf den Erkenntnisprozess legen. Denn um wirklich alle Erkenntnisse zu erfassen, müssen evtl. *P&P*-Äußerungen (z.B. *design*-Äußerungen) doppelt kodiert oder das Konzept *explain_finding* in eine Eigenschaft von *P&P*-Konzepten umgewandelt werden. Außerdem: Wenn nicht ausschließlich mit Elementen der BKM kodiert wird, muss es keineswegs so sein, dass bei der Betrachtung von Äußerungen die primäre oder sekundäre Illokution an und für sich im Mittelpunkt steht.

In diesem Zusammenhang sei (noch einmal) auf zwei grundlegende Probleme eingegangen:

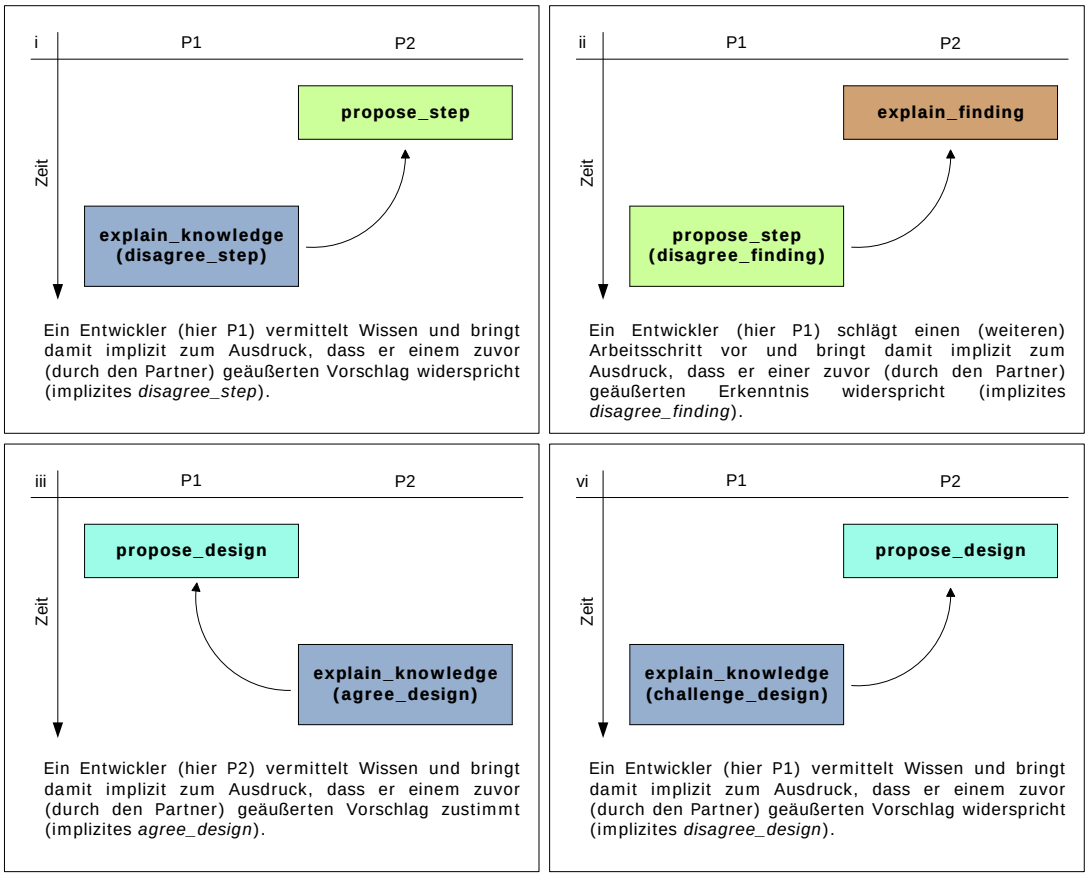


Abb. 9.2: Implizite Zustimmungen und Ablehnungen: Wie bereits im Rahmen der Klassen *step* (siehe S. 205 (Teilabbildung i)), *finding* (siehe Tabelle 7.33 (Teilabbildung ii)) und *knowledge* (siehe S. 338 (Teilabbildung iii)) angesprochen, kommt es vor, dass Zustimmugen oder Widersprüche nicht explizit geäußert, sondern „lediglich“ implizit mit artikuliert werden. Die BS/BKM legt fest, dass wenn es sich dabei um Übergänge zwischen mit *P&P*-Konzepten und *UK* zu annotierenden Äußerungen handelt, Doppelkodierungen vorgenommen werden sollten. Das Beispiel iv basiert auf einem Phänomen aus Sitzung PR2.4. Es wurde folgender Dialog geführt: „Wahrscheinlich hätte ich erst mal einfach ins Debug-Menü oder so (...) 'nen Menüpunkt gepackt, der dann einfach startet“ (*P2.propose_design*); „Also wenn ich die Erfassung richtig in Erinnerung hab', sollt's 'n Leichtes sein, da 'n Menüpunkt direkt auch an die Stelle hinzukriegen, wo's hin soll“ (*P1.explain_knowledge*/*P1.challenge_design*).

– **Problem 1: Was soll im Rahmen des hier propagierten Verfahrens unter einem Konzept oder einer Kategorie bzw. unter „Grenzen“ zwischen Konzepten bzw. Kategorien verstanden werden?:** Petra Muckel [140, S. 212] verweist darauf, dass nach Gamm [69, S. 11 f] „Skepsis einem Denken gegenüber angebracht [ist], das ‚die Welt im Rahmen eindeutig unterschiedener Kategorien‘ zu begreifen versucht. Kategorien zu benutzen stützt sich auf die Möglichkeit, ‚die Welt in wohldefinierte Grenzen einschließen zu können, obschon der einfache Versuch, eine Grenze *als* Grenze, einen Unterschied *als* Unterschied zu definieren, es notwendig macht, auf *irgend etwas* zurückzugreifen, das sich der Macht der Definition entzieht.‘“[13] Muckel [140, S. 213 f] führt aus, dass es neben diesem von Gamm adressierten „klassischen Verständnis einer Kategorie“, in dem „der Fokus auf der *Grenze* [liegt], die eine Kategorie impliziert“ und wo „es möglich [erscheint], *Kriterien*, die den Ein- oder Ausschluss definieren, *exakt zu bestimmen* [(z.B. bei der Mengenlehre)], zumindest noch ein zweites gibt. Dieses legt den „Fokus auf [die] *Zusammengehörigkeit der Mitglieder* einer Kategorie untereinander, hier scheint es *nicht möglich, Kriterien* für den Ein- oder Ausschluss *exakt zu benennen*“. Als typischen Vertreter betrachtet sie Wittgensteins Ansatz der Familienähnlichkeit.[14] Nach ihrer Einschätzung „steht das Kategorienverständnis der Grounded Theory Methodologie den Familienähnlichkeiten WITTGENSTEINs näher als das Kategorienverständnis der Mengenlehre“ [140, S. 214]. Sie begründet dies damit, dass der Fokus der GTM „auf der Entwicklung der Kategorie [liegt], die eher unscharf und polyphon entfaltet als in Form von Zuordnungskriterien abgegrenzt und festgeschrieben wird. [...] Die Kategorien der Grounded Theory Methodologie bleiben bis zum Abschluss der Theorieentwicklung (und darüber hinaus) *im Prozess und offen* für Veränderungen, sie werden dem Prinzip des permanenten Vergleichs untergeordnet. [...] In der GTM werden *Ähnlichkeiten und Relationen der Daten untereinander* zur sukzessiven Elaboration der Kategorien und ihrer Beziehungen miteinander herangezogen (Analogie des Zuordnungsprozesses zu WITTGENSTEINs Ansatz der Familienähnlichkeit). In der GTM werden [...] durch eine Abfolge von aufeinander abgestimm-

[13] Vergleiche auch mit der in Exkurs 9 erläuterten Aussage von Clausewitz zur Einteilung von „wirklichen Dingen“.

[14] Muckel [140, S. 212 f] bezieht sich darauf, dass Wittgenstein „angesichts der Definitionsschwierigkeiten des Spielbegriffs [...] davon ausgeht, dass es schwierig, wenn nicht gar unmöglich sein kann, ein festes Repertoire von typischen Merkmalen [...] festzulegen, das über die Zugehörigkeit zu einem Begriff entscheidet. [...] WITTGENSTEIN schlägt stattdessen vor, solche Aspekte zu einem Begriff (einer Kategorie) zusammenzufassen, die „Familienähnlichkeit“ aufweisen: Wie die Mitglieder einer Familie Ähnlichkeiten und Unterschiede aufweisen, können unterschiedliche Spiele unter den Begriff des Spiels subsumiert werden. [...] WITTGENSTEINs Überlegungen führen ihn zu der Einschätzung, dass der Begriff des Spiels (sinnvoll) angewendet werden kann, obgleich er keine definierbaren Grenzen aufweist und somit keine Regeln für seine Anwendung abzuleiten sind.“ Vergleiche auch mit [223].

ter Kodierprozeduren *polyphone Kategorien* angestrebt, die Widersprüche zulassen sowie dem *Anspruch der Dichte* gehorchen“ [140, S. 215].

Die in der vorliegenden Arbeit propagierte Methode schließt sich der Auffassung von Muckel an. Ungeachtet davon, ob man die Elemente der BKM nun als Konzepte oder Kategorien bezeichnen möchte (siehe S. 125), können und sollten die meisten der Basiskonzepte – trotz aller Regeln, die die BS zur Verfügung stellt – nicht als Kategorien im Sinne der Mengenlehre betrachtet werden (auch wenn im Rahmen ihrer Erörterungen zum Teil Begrifflichkeiten aus der Mengenlehre herangezogen werden). Vielmehr handelt es sich – wie schon mehrfach erläutert – um Objekte, die weiter auszuarbeiten sind. Dies bedeutet unter anderem auch, dass sogar diejenigen Konzepte, die, vor allem um das Verständnis zu erleichtern, auf einen disjunkten Einsatz hin optimiert wurden, nicht immer disjunkt voneinander eingesetzt werden können und sollten (siehe z.B. die Diskussion von *propose_todo* vs. *amend_design*/*propose_design* auf S. 216). Die Regeln der BS müssen also in weiterführenden Untersuchungen immer wieder hinterfragt werden, z.B. dahingehend, ob sie den Erkenntnisprozess in der aktuellen Situation eher fördern oder eher behindern. Ein Brechen/Anpassen von Regeln wird hierbei aller Wahrscheinlichkeit nach notwendig werden, sollte aber immer mit einer Bewusstmachung darüber, warum man dies tut, einhergehen. Ein „stumpfes“ Anwenden der Konzepte und Regeln ist hingegen nicht im Sinne des hier propagierten Vorgehens bzw. Schemas, bei dem es in erster Linie darum geht, Begrifflichkeiten und vorläufige Strukturierungen zur Verfügung zu stellen, die das Nachdenken und Kommunizieren über die Phänomene bei der PP erleichtern sollen.

– **Problem 2: Wie kann sichergestellt werden, dass die richtige Intention bzw. die richtige Illokution ermittelt wird?:**

  Die einfache Antwort hierauf ist: Es kann nicht sichergestellt werden. Dies stellt aber kein wirkliches Problem dar. Denn folgt man den unter Problem 1 erörterten Gedanken, so wird schnell klar, dass der Versuch, für *jede* mögliche Situation sicherstellen zu wollen, dass anhand von Regeln das „einzig richtige“ (BKM-)Konzept, also auch die „einzig richtige“ Intention, annotiert wird, nicht das zentrale Anliegen einer Untersuchung auf Basis der GTM sein kann. Konzepte sind vielmehr ein Hilfsmittel und dienen im Rahmen der GTM dazu, etwas über Prozesse, an denen Menschen beteiligt sind, im vorliegenden Fall über den Prozess der PP, zu lernen. Trotzdem ist es natürlich nicht nur erlaubt, sondern sogar erwünscht, die Frage danach zu stellen, woran man Intentionen bzw. Illokutionen erkennen kann oder will und wie im Rahmen der BS/BKM damit umgegangen werden soll, dass Gesagtes nicht unbedingt immer gleich Gehörtes ist.[15] Dies sollte im

[15] Um den Umfang der vorliegenden Arbeit nicht zu sprengen, muss auf eine Erörterung der BS/BKM vor dem Hintergrund des Vier-Seiten-Modells von Friedemann Schulz von Thun (siehe z.B. [208]) verzichtet werden.

Rahmen der Darstellung einer GT erörtert werden. Die Praktik der Paarkodierung kann dabei helfen, derartige Probleme frühzeitig im Analyseprozess zu reflektieren.

9.3.2 Angemessene HCI/HEI-Konzepte auswählen

Analog zur Regel, dass HHI-Konzepte, die (auf eine gewisse Weise zu betrachtenden) *Intentionen von Äußerungen* adressieren, gilt für HCI/HEI-Konzepte, dass sie dahingehend konzipiert sind, den (auf eine gewisse Weise katalogisierten) *Zweck von „Nichtsprechhandlungen“* festhalten zu können. Um zu einer angemessenen HCI/HEI-Kodierung zu kommen, muss also ermittelt werden, mit welcher (übergeordneten) Absicht ein Entwickler bestimmte Tätigkeiten durchführt. Die Tätigkeit selbst darf dabei nur als Mittel zum Zweck betrachtet werden. Beispielsweise kann die Tätigkeit des Scrollens ein *verify_sth* oder *search_sth* darstellen, was es zu ermitteln gilt.

9.3.3 Berücksichtigung des Kontextes von Äußerungen

Wie bereits mehrfach diskutiert, spielt der Kontext einer Äußerung eine entscheidende Rolle bei ihrer Konzeptualisierung. Bisher wurde allerdings nicht erörtert, was genau unter dem Kontext einer Äußerung verstanden werden soll.[16] Dies wird nun nachgeholt, indem auf eine Begriffsbestimmung aus den Sprachwissenschaften nach Bußmann [37] zurückgegriffen wird, nach dem zum Kontext der verbale und non-verbale (z.B. mimischen) Kontext, der aktuelle Kontext der Sprechsituation, der soziale Kontext der Beziehung zwischen Sprecher und Hörer, ihr Wissen und ihre Einstellungen gehören.

Die Definition macht deutlich, dass ein Forscher in der Regel nur eingeschränkte Möglichkeiten besitzt, eine Äußerung zu deuten. Denn selbst wenn wie bei den hier diskutierten Sitzungen, Audio- und Videoaufzeichnungen sowie ein Screencast zur Verfügung stehen, kann auf bestimmte Informationen, wie z.B. solche über die Beziehung zwischen Sprecher und Hörer oder ihr Wissen und ihre Einstellungen, bestenfalls eingeschränkt zugegriffen werden. Der Kontext, der einem (externen, wissenschaftlich ausgerichteten) Beobachter zur Verfügung steht – hier als *Beobachterkontext* bezeichnet (siehe auch Exkurs 4) – enthält also in der Regel weniger Informationen oder hat eine andere Struktur, als der des Sprechers oder auch der des Hörers.[17] Allerdings steht einem solchen Beobachter eine Methode zur Verfügung Kontextinformationen einzuholen, die Sprecher und Hörer nicht besitzen. Es ist ihm nämlich möglich „in die Zukunft zu sehen“ – z.B. um dort nach Informationen zu suchen, die ihm selbst fehlen, die Sprecher und Hörer

[16] Es wurde nur darauf eingegangen, wie der Begriff Kontext im Zusammenhang mit dem paradigmatischen Modell der GTM definiert ist (siehe S. 66).

[17] Darüber hinaus gilt, dass jede gängige Form der Aufzeichnung von Interaktion zu einer „Reduktion von Realität“ führt [31, S. 33 f].

aber schon zum Zeitpunkt der Äußerung besessen haben. Bei derartigen Analysen kann es allerdings passieren, dass – ungewollt – auch solche Informationen in die Beurteilung einer Äußerung einfließen, die zum Zeitpunkt der Verbalisierung weder dem Sprecher noch dem Hörer zur Verfügung standen. Dies kann dazu führen, dass unzutreffende Kodierungen vorgenommen werden (siehe beispielsweise die Erörterung in Fußnote 92 auf S. 226). Eine Einbeziehung von Fakten aus dem späteren Verlauf der Sitzung sollte also, zumindest wenn es darum geht, die Illokutionen von Äußerungen zu analysieren, mit Bedacht betrieben werden.

Achtung: Auch in dem der BS/BKM zugrundeliegenden Prinzip des Aufzeigens von sprachlichen Bezügen (siehe S. 161) spiegelt sich das Anliegen wieder, Äußerungen anhand ihres Kontextes zu klassifizieren. Wie bereits in Matrix 7.50 für reaktive Wissens- oder Erkenntnisäußerungen demonstriert, kann diese Art der Betrachtung komplexere Analysen zur Folge haben.

9.3.4 Umgang mit implizierten Handlungsankündigungen

Es konnte beobachtet werden, dass aus einer Frage (oder anderen Form von Äußerung) eines Entwicklers ersichtlich werden kann, dass dieser ein bestimmtes Vorhaben ins Auge gefasst hat, auch wenn er selbst nicht (bewusst) vor hatte, dieses Vorhaben (z.B. im Sinne eines *step*s oder *design*s) zu explizieren. Ein derartiges Phänomen wurde bereits im Rahmen der Erörterungen der Klasse *step* auf S. 204 diskutiert: Der Entwickler stellt hier eine Frage zu einer Funktionalität der IDE Eclipse. Aus dieser wird, bezieht man den Kontext ein, ersichtlich, welchen Arbeitsschritt (*step*) er als nächstes durchzuführen gedenkt. Solche Phänomene werden im Rahmen der BS, wie bereits auf S. 204 ausgeführt, *implizierte Handlungsankündigungen* genannt. Sie sind weniger einfach zu identifizieren als explizit geäußerte Handlungsankündigungen, da in der Regel die inhaltliche Bedeutung solcher Äußerungen durchdrungen werden muss, um erkennen zu können, worauf diese wohl zwangsläufig hinauslaufen. Falls solche Phänomene entdeckt werden, sollten sie aber, obwohl die Sprecher weder direkt noch indirekt den Sprechakt des Vorschlagens vollziehen, mit *propose*-Konzepten annotiert werden. Schließlich ist es nicht ausgeschlossen, dass auch der Partner implizierte Handlungsankündigungen erkennt (vgl. auch hier mit dem Beispiel auf S. 204) und diese somit Auswirkungen auf den Prozessverlauf haben können. Weiterführende Untersuchungen müssen festlegen, wie viel Aufwand vor dem Hintergrund des anvisierten Analyseziels gerechtfertigt ist, um nach solche Phänomenen zu suchen.

9.3.5 Segmentierung von Äußerungen

Zu Beginn der Untersuchungen war nicht ersichtlich, auf welche Weise eine Segmentierung der Daten idealerweise vorgenommen werden sollte. Eine „Segmentierungsstrategie“ wurde vielmehr als eines der Ergebnisse der Herleitung der BS angesehen (siehe Abschnitt 6.1). Im Laufe des Untersuchungsprozesses stellte sich

dann heraus, dass vor allem die verbalen Interaktionen zwischen den Entwicklern kodiert werden sollten und dass in diesem Zusammenhang auf Äußerungsebene zu arbeiten ist. Hierbei wurde eine Äußerung grob als eine „sinnvoll-abgeschlossene sprachlich-kommunikative Einheit“ betrachtet (siehe Exkurs 4). Da sich des Weiteren zeigte (bzw. festgelegt wurde), dass die HHI-Konzepte in gewisser Weise auf Basis des illokutiven und präpositionalen Aktes – jeweils subsumiert in herauszuarbeitenden Klassen – vergeben werden sollten, orientiert sich das, was als „sinnvoll-abgeschlossene sprachlich-kommunikative Einheit“, also Äußerung gesehen werden soll, im Rahmen der BS zumindest in großen Teilen an diesen beiden Akten und an der Art, diese zu subsumieren.[18] Bei einer *BS-Segmentierung* der Verbalisierungen von Entwicklern (zumindest was die Vergabe von Konzepten betrifft, die nicht in die Klasse der Fassadenkonzepte fallen) steht also zuerst einmal im Fokus, diese derart in Teile zu zerlegen, dass die vorkommenden, nach Regeln der BS/BKM zu subsumierenden „Informationseinheiten“ in Hinsicht auf ihre „Verwendung“ im Rahmen von subsummierenden BS/BKM-Handlungszielen separiert werden. Darüber hinaus spielt bei der Segmentierung eine Rolle, wie die ins Auge gefassten Äußerungen oder Segmente im Kontext sowie in Hinsicht auf eine potentielle Einzelkodierung zu interpretieren wären. Schließlich sollten direkt hintereinander formulierte Äußerungen desselben Typs in der Regel zusammen kodiert werden (siehe z.B. Anmerkungen auf S. 267). In Abbildung 9.1 wird das Vorgehen mit zwei Beispielen bebildert. Es ist dargestellt, wie eine (zumindest vorläufig) gültige Segmentierung im Wechselspiel mit Kodierungsüberlegungen festgelegt wird. Bei der Verwendung der BKM sollten die dort dargestellten einzelnen Schritte – um nicht in ein starres Kodieren zu verfallen – in der Regel nicht sequenziell, sondern mehr oder weniger gleichzeitig durchlaufen werden (ganzheitliche Betrachtung).

9.3.6 Umgang mit Start- und Endpunkten einzelner Phänomenkodierungen

Wie bereits in den vorangegangenen Kapiteln diskutiert wurde, lässt sich nicht immer zweifelsfrei bestimmen, wann ein in den Videodaten ausgemachtes Phänomen tatsächlich genau beginnt oder endet. Dies gilt insbesondere für Phänomene vom Typ HCI/HEI, z.B. wenn das Mithandeln eines Observers kodiert werden soll (siehe Abschnitt 8.1.9). Derartige Probleme bilden einen der Gründe dafür, dass nicht nur Annotationen von Datensegmenten als vorläufig angesehen werden müssen, sondern auch die Datensegmente selbst. Nur wenn ein Forscher – z.B. im Rahmen des ständigen Vergleichens – im Hinterkopf behält, dass es möglich ist, dass Grenzen von Datensegmenten evtl. in einer anderen Weise neu gesetzt werden müssen, ist sichergestellt, dass Phänomene letztendlich adäquat betrach-

[18] Wie schon in Fußnote 7 auf S. 124 ausgeführt, sprengt eine Diskussion darüber, inwieweit das initiale Festlegen einer Syntax für die Konzeptnamen Einfluss auf das Ergebnis, also z.B. die Art der Phänomene, die von der BS/BKM adressiert werden, genommen hat, den Rahmen der vorliegenden Arbeit.

tet werden. In diesem Zusammenhang kann es helfen, entsprechend benannte Memos zu annotieren, um potentiell interessante Abgrenzungen im Auge behalten zu können. Dies kann auch dazu beitragen, dass, falls eine QDA-Software eingesetzt wird, die es ermöglicht, Datensegmente in Videos direkt anzuspringen und abzuspielen – wie z.B. ATLAS.ti[19] – nicht mehr in Erwägung gezogen wird, Segmente nach ihrer Festlegung noch einmal zu vergrößern.

9.3.7 Kodierung von Äußerungswiederholungen

Die BS unterscheidet zwischen zwei Formen von, nicht notwendigerweise wörtlichen, Wiederholungen von Äußerungen durch dieselbe Person:

1. *Zeitnahe Wiederholungen:* Wiederholungen, durch die sichergestellt werden soll, dass der Partner die ursprüngliche Äußerung bzw. das ursprüngliche Anliegen, z.B. einen Transfer von Wissen/Erkenntnissen oder einen Vorschlag, registriert. Sie erfolgen – soweit dies bisher beobachtet werden konnte – innerhalb von wenigen Sekunden oder Minuten.[20]

2. *Nicht zeitnahe Wiederholungen*

Zeitnahe Wiederholungen werden im Rahmen der BS mit demselben Konzept annotiert wie die ursprüngliche Äußerung. Die BS markiert solche Wiederholungen also nicht als solche. Gleiches gilt auch für die nicht zeitnahen Wiederholungen – mit einer, bereits auf S. 268 erläuterten Ausnahme: Sollte eine Erkenntnis später noch einmal artikuliert werden, wird diese dann nicht durch die Klasse *finding*, sondern durch die Klasse *knowledge* konzeptualisiert. Es wird – zumindest wenn keine Indizien vorliegen, die etwas anderes nahelegen – davon ausgegangen, dass die Erkenntnis zum Teil des Bestandswissens geworden ist. Ein derartiges Vorgehen verlangt vom Forscher in der Regel eine hohe Aufmerksamkeit. Schließlich muss er sich daran erinnern, dass genau dieses Wissen schon einmal geäußert wurde und nun keine neue Erkenntnis mehr darstellt (zumindest dann nicht, wenn der Sprecher nicht erkennen lässt, dass er dieses Wissen wieder vergessen hat).

9.3.8 Umgang mit eingeschränkten Zustimmungen und Ablehnungen

In unterschiedlichen Zusammenhängen ist schon darauf eingegangen worden, dass Zustimmungen und Ablehnungen nicht immer derart formuliert sind, dass davon ausgegangen werden kann, dass es sich um weitgehend uneingeschränkte Urteile des Sprechers handelt (siehe z.B. S. 176 (*disagree_design*) oder S. 226 (*agree_strategy*)).[21] Dies kann in einzelnen Fällen sogar dazu führen, das es

[19] Siehe Fußnote 5 auf S. 122.

[20] Siehe auch Fußnote 118 auf S. 252.

[21] Zur Erinnerung: Auch Vorschläge können derart formuliert sein, dass zu erkennen ist, dass der Sprecher Zweifel an ihrer Richtigkeit/Angemessenheit hat. Sie werden im Rahmen der BS trotzdem der Klasse *propose* zugerechnet (siehe z.B. S. 169).

schwierig wird, überhaupt zu entscheiden, ob es sich bei einem Urteil um eine Zustimmung (*agree_**) oder um eine Ablehnung (*disagree_**) handelt. Bisher wurden zwei Typen von Phänomenen identifiziert, die zu derartigen Problemen führen:

1. Der Sprecher ist sich seines Urteils nicht hundertprozentig sicher und äußert dies mehr oder weniger eindeutig.

2. Der Sprecher kann nicht im vollen Umfang oder in allen Teilen zustimmen und expliziert dies in gewisser Weise.

In beiden Fällen sollte versucht werden, herauszufinden, ob die Anmerkung „eher“ als Ablehnung oder als Zustimmung zu deuten ist und es sollten – zumindest vorläufig – entsprechende Kodierungen vorgenommen werden. Darüber hinaus sollte das Phänomen, zumindest wenn es für das Untersuchungsziel potentiell relevant zu sein scheint, mit einem entsprechenden Memo – oder vielleicht sogar mittels einer Eigenschaft – markiert werden. Außerdem muss – zumindest bei komplexen Objekten wie z.B. bestimmten Strategien – in Betracht gezogen werden, ein *challenge*-Konzept einzusetzen (siehe z.B. S. 223). Alle diese Maßnahmen sollen es erleichtern, im weiteren Verlauf der Analyse herauszufinden, was an derartigen Stellen genau passiert. Letztendlich wird der konkrete Umgang mit derartigen Phänomenen aber weiterführenden Untersuchungen überlassen.

9.3.9 Umgang mit Eigenkorrekturen

Die reaktiven Basiskonzepte, insbesondere die der Klassen *challenge* und *amend*, sind derart konzipiert, dass sie nicht nur zur Kodierung von verbalen Reaktionen eines Entwicklers auf Äußerungen des Partners eingesetzt werden können, sondern auch dann, wenn sich ein Entwickler auf eine von ihm selbst hervorgebrachte Äußerung bezieht (siehe z.B. S. 175 (bez. *design*), S. 196 (bez. *step*) oder S. 269 (bez. *finding*)). Dies liegt darin begründet, dass, wie an vielen anderen Stellen auch, dem Aufzeigen von sprachlichen Bezügen eine hohe Priorität eingeräumt wird (siehe S. 161). Der Aspekt der Selbstkorrektur wird von der BS/BKM auf diese Weise allerdings nur indirekt adressiert. Weiterführende Studien sollten derartige Phänomene, wenn sie dann das Untersuchungsziel berühren, expliziter herausstellen.

9.3.10 Kodierung von Begründungen

Auf den ersten Blick mag es erstaunlich sein, dass mit der BKM Begründungen nicht explizit, das heißt durch eine eigene Objektklasse wie z.B. *rationale*, als solche markiert werden können. Diese Entscheidung hat folgenden Grund: Durch die einzelnen Objekte der BKM wird Wissen nicht danach klassifiziert, zu welchem Zweck bzw. mit welcher Intention es im Rahmen einer Verbalisierung eingesetzt wird – zumindest nicht primär. Der Einsatzzweck wird vielmehr erst

durch das Verb kenntlich gemacht. Hierbei wird eine Begründung unter *explain* subsumiert. Wenn in weiterführenden Untersuchungen ein besonderes Interesse am Umgang mit Begründungen besteht, kann diesem z.B. unter Verwendung von Eigenschaften nachgegangen werden. Äußerungen vom Typ *explain_knowledge* oder *explain_finding* müssten dann mit einer entsprechenden Eigenschaft ausdifferenziert werden.

9.3.11 Geprächsabläufe als azyklischen gerichteten Graphen betrachten

Schon bei der Erörterung des Konzeptes *strategy* wurde darauf hingewiesen, dass es natürlich nicht so ist, dass sich reaktive Äußerungen immer auf genau eine zuvor gemachte Äußerung beziehen (siehe Abbildung 7.6). Die inhaltlichen Bezüge, die im Rahmen eines Gesprächsablaufes gemacht werden, bilden also keineswegs immer eine gerichtete Kantenfolge, sondern allgemeiner einen azyklischen gerichteten Graphen[22] mit (Teil-)Äußerungen als Knoten und der Relation „bezieht sich inhaltlich auf“[23] als Kanten. Obwohl die BKM dahingehend konzipiert wurde, Bezüge aufzuzeigen (siehe S. 161), berücksichtigt sie diese Tatsache nicht bzw. nicht explizit. Bezüge werden vielmehr nur in Hinsicht auf die chronologisch letzte „passende“ Äußerung erfasst – und das auch nur dann, wenn sich die Äußerungen im Wesentlichen mit demselben Objekt, genau genommen mit demselben Äußerungsgegenstand (siehe S. 161), in einer in der Regel Illokutionen berücksichtigenden Art und Weise befassen – zumindest wenn man von Shift-Kodierungen absieht.

Die Regel, nach der im Rahmen einer BS-Kodierung nur Bezug auf die letzte „passende“ Äußerung genommen wird, erleichtert den Prozess der Kodierung erheblich. Bezüge auf weiter zurück liegende Äußerungen oder solche eines anderen Objekttyps sind aber somit in einer BS-Kodierung in der Regel nicht mehr direkt zu erkennen. So werden beispielsweise Vorschläge und Erklärungen zu diesen

[22] Nach [90, S. 242f] sei eine *gerichtete Kantenfolge* und ein *azyklischer gerichteter Graph* wie folgt definiert: „Ein [...] *gerichteter Graph* G besteht aus einer Menge $V = V(G)$, den *Knoten* von G und aus einer Menge $E = E(G)$ von geordneten Paaren $k = [x, y]$ mit $x, y \in V$, den *gerichteten Kanten* von G. [...] Die Richtung der Kante $k = [x, y]$ zeigt dabei von x nach y. x heißt *Anfangspunkt* und y *Endpunkt* der gerichteten Kante $[x, y]$. [...] Eine *gerichtete Kantenfolge* von x_1 nach x_n ist eine Folge von gerichteten Kanten $[x_1, x_2][x_2, x_3], ..., [x_{n-1}, x_n]$ [...], ein *gerichteter Kreis* ist eine gerichtete Kantenfolge, in der alle Knoten bis auf Anfangs- und Endpunkt verschieden sind. [...] Ein gerichteter Graph, in dem es keinen gerichteten Kreis gibt, heißt *azyklischer gerichteter Graph*.“ Außerdem: Von Äußerungen, die sich aufeinander beziehen, wie es z.B. vorkommen kann, wenn sich die Partner gegenseitig ins Wort fallen, sei hier abgesehen.

[23] In der linguistischen Gesprächsanalyse wird zwischen unterschiedlichen Formen von Bezügen unterschieden. So wird beispielsweise von den Relationen *explizite Wiederaufnahme* (z.B. durch „Wiederholung desselben Substantivs“) und *implizite Wiederaufnahme* (Bezüge ohne Referenzidentität, aber mit einem spezifischen „Kontiguitätsverhältnis, das ontologisch bzw. kulturell begründet ist“) gesprochen [31, S. 73]. Um den Rahmen der vorliegenden Arbeit nicht zu sprengen, wird hier nicht näher auf die unterschiedlichen Formen von Bezügen eingegangen.

Vorschlägen nicht explizit zueinander in Beziehung gesetzt. Solche Bezügen erfassbar und somit explizit untersuchbar zu machen wird weiterführenden Studien überlassen.

Kapitel

10 Die Herleitung der BS/BKM

I'll be on my way hey
I'll be on my way oh
I'll be on my way oh
I'll be on my way

I'll Be On My Way, The Beatles

Das Verfahren der GTM verlangt, dass nachvollziehbar dargestellt wird, wie zu den Ergebnissen einer GTM-Studie gekommen wurde. Dieser Vorgabe wurde bei der Darstellung der einzelnen Konzeptklassen gefolgt. Im vorliegenden Kapitel wird nun noch auf das übergeordnete Vorgehen eingegangen.

Die Entwicklung der BS/BKM durchlief – nachdem der erste Analyseversuch gescheitert war (siehe Abschnitt 5.1) – eine Vielzahl von Iterationen. Diese wurden nicht durch jeweils im Vorfeld definierte Analysezeiträume, sondern durch sukzessive erarbeitete, zum Teil der Methode des theoretischen Samplings (siehe S. 52) folgende Zwischenziele bestimmt. So ging es in der ersten Iteration – die sich über mehrere Monate hinzog – darum, auf Basis einer einzigen PP-Sitzung (ST1.1) eine erste Version – in gewisser Weise einen Prototypen – der BS/BKM herzuleiten und dabei die Methoden der erweiterten GTM (siehe Abschnitt 5.2) auf ihre Nützlichkeit hin zu testen.[1] Weitere Iterationen hatten beispielsweise die Ziele

- die Ergebnisse aus der Untersuchung der studentischen Sitzung ST1.1 mit Vorgängen in einer Sitzung mit professionellen Softwareentwicklern abzugleichen bzw. die BS/BKM auf einen diesbezüglichen Einsatz hin zu erweitern,
- das Zusammenspiel von HHI-Aktivitäten und HCI-Aktivitäten besser zu verstehen oder
- eine zufriedenstellende Abgrenzung zwischen den Klassen *step* und *strategy* zu finden.

In einigen der Iterationen mussten neue Daten herangezogen werden, in anderen reichte es aus, bereits analysierte Daten weiter zu erkunden. Zeitweise wurden Iterationen sogar parallel zu anderen durchgeführt. Beispielsweise diente – wie schon in Abschnitt 4.2.2 angesprochen – eine vom Autor des vorliegenden Dokumentes betreute Masterarbeit, in der es primär darum ging, Beeinflussungen der Aktivitäten eines Entwicklers durch seinen Partner zu untersuchen [231],

[1] In dieser Iteration kam die Praktik des Paarkodierens zum Einsatz.

auch dazu, die Einsatzfähigkeit der BS/BKM zu erproben. Hierbei kam über ein halbes Jahr hinweg eine schon in den meisten Details ausgearbeitete Version der BS/BKM zum Einsatz. Analysiert wurden die Sitzungen ST1.1, PR1.1, PR2.1, PR2.2, PR2.3, PR2.4 und PR3.1. Der Autor des vorliegenden Dokumentes und der Abschlussarbeiter diskutierten Probleme bei der Verwendung der BS zeitnah und verwendeten partiell Paarkodierung. Außerdem wurden neu entdeckte Phänomene mit ähnlichen aus vorher analysierten Sitzungen verglichen. Die Ergebnisse dieser Diskussionen und Vergleiche, also die Einsichten aus den Kodierungen flossen direkt in die Ausarbeitung der BS/BKM ein (ein Beispiel hierfür findet sich in Abbildung 9.2).

Im Laufe der Entwicklung der BS zeigte sich interessanterweise, dass die Struktur des Ergebnisses der ersten Iteration soweit tragfähig war, dass zwar Erweiterungen und kleinere Modifikationen, aber keine grundlegenden Modifikationen vorgenommen werden mussten – auch wenn die erste Version der BS/BKM wegen einer Vielzahl ungeklärter Teilaspekte noch weit davon entfernt war, im angestrebten Sinne einsatzfähig zu sein. Vergleicht man die erste (unveröffentlichte) Version mit der hier vorgestellten finalen, fallen unter anderem folgende Punkte auf:

- Von den HHI-Objektklassen existierte in der ersten Version schon der weitaus überwiegende Teil (*activity*, *completion*, *finding*, *gap in knowledge*, *hypothesis*, *knowledge*, *source of information*, *state*, *step*, *strategy*) – wenn auch vereinzelt unter anderem Namen (*step* hieß beispielsweise *action*).

- Eine Reihe von HHI-Objektklassen adressierten in der ersten Version einen breiteren Bereich als in der aktuellen. Beispielsweise wurden durch den Vorgänger von *step* (*action*) sowohl Arbeitsschritte im jetzigen Sinne wie auch Designaspekte adressiert.

- Viele Abgrenzungen von HHI-Objektklassen waren in der ersten Version noch sehr unscharf. Beispielsweise war zwar offensichtlich, dass es sinnvoll ist, zwischen Arbeitsschritten (etwas Kurzfristigem) und Strategien (etwas Längerfristigem) zu unterscheiden, im Einzelfall machten solche Differenzierungen aber immer wieder Probleme, so dass später nach operationalisierbaren Kriterien für eine Differenzierung gesucht wurde. Insgesamt lässt sich sagen, dass, auch wenn die Aufwände für einzelne Analyseschritte nicht protokolliert wurden, die Beseitigung derartiger „Unschärfen“, z.B. durch das Herausarbeiten primär charakterisierender Eigenschaften wie den Strategietypen (siehe S. 217 f), letztendlich den Hauptaufwand bei der Herleitung der BS/BKM verursachte.

- Eine Gruppierung nach prozessorientierten, produktorientierten und universellen Konzepten und eine damit einhergehende Fokussierung der Konzepte existierte noch nicht. So war zwar der Wissenstransfer als ein zentraler Aspekt aus den Daten extrahiert worden – schließlich gab es schon Objektklassen wie *knowledge* oder *finding* – es war aber noch nicht festgeschrieben,

in welcher nachvollziehbaren Art und Weise er im Rahmen der zu entwickelnden Konzeptmenge von anderen Aspekten der Kommunikation separiert werden soll. So wurde beispielsweise anfänglich nicht zwischen Vorschlägen und Vorschlagsbegründungen unterschieden, wohl aber zwischen Vorschlägen und sonstigem Wissenstransfer. Erst die spätere Einführung der Klasse der universellen Konzepte und die damit einhergehende Festlegung, welche Formen von Wissenstransfer explizit durch die BKM erfasst werden sollen, löste dieses Problem.

– Auch von den HHI-Verbklassen existierte schon der weitaus überwiegende Teil (*amend*, *ask*, *agree*, *challenge*, *decide*, *disagree*, *explain*, *propose*, *remember*, *think aloud*), allerdings nicht immer mit derselben Bedeutung wie in der finalen Version. So wurde beispielsweise ursprünglich versucht, mittels *explain* und *think aloud* zwischen gerichteter und ungerichteter Kommunikation zu unterscheiden – eine Differenzierung, die zunehmend Probleme machte (siehe auch Fußnote 31 auf S. 159). Darüber hinaus existierten in der ersten Version auch Verben wie `ignore` (Ignorieren von Vorschlägen) oder *detect* (Erkennen von Wissenslücken), mit denen – mehr oder weniger bewusst – versucht wurde, zusätzlich Phänomene „auf anderen Ebenen“ zu adressieren. Dieses Vorhaben wurde aber zugunsten der Übersichtlichkeit der BKM abgebrochen, bevor die Gestalt dieser Ebenen herausgearbeitet war.

– Die in der initialen Version vorhandenen Abgrenzungsprobleme zwischen Objekten sowie zwischen Verben führten zwangsläufig dazu, dass sich bei der Verwendung der Basiskonzepte unterschiedlich geartete Probleme ergeben konnten. Diese wurden, wenn sie sich allem Anschein nach nicht ad hoc oder mit überschaubarem Aufwand lösen ließen, erst einmal in Form von Memos notiert. Beispielsweise ließ sich in bestimmten Fällen, wenn eine Äußerung in Form einer Frage formuliert worden war, nicht zufriedenstellend entscheiden, ob ein *propose_action* oder ein *ask_knowledge* zu annotieren ist. Dies lag unter anderem daran, dass zu diesem Zeitpunkt *action* als Wissenstyp betrachtet wurde. Derartige Probleme wurden in der Regel erst in nachfolgenden Iterationen – nötigenfalls unter Zuhilfenahme einer größeren Datenmenge – explizit angegangen.

– Die HEI/HCI-Konzepte existierten zwar auch weitgehend komplett (*do_sth*, *explore_sth*, *read_sth*, *show_sth*, *verify_sth*, *write_sth*), allerdings konnten sie, da die durch sie adressierten Phänomene noch nicht wirklich ständig miteinander verglichen worden waren, nur als sehr grobe Marker eingesetzt werden. Der Grund für die Vernachlässigung des ständigen Vergleichens der HCI/HEI-Konzepte war die Entscheidung, den Fokus auf die interessanter erscheinenden HHI-Konzepte zu legen.

Zusammenfassend muss also festgehalten werden, dass die erste Version der BKM der finalen auf den ersten Blick sehr ähnlich sieht, dass es sich bei den Kon-

zepten der ersten Version aber doch eher um datennahe Kodes als um Kategorien handelt. Erst die nachfolgenden Iterationen sorgten dafür, dass die BKM bzw. die BS zu einem dichten Rahmenwerk wurde und sich eine theoretische Sättigung einstellte.

Kapitel 11 Einordnung der BS

Dave: Well HAL, I'm damned if I can find anything wrong with it.
HAL: Yes, it's puzzling. I don't think I've ever seen anything quite like this before.
2001: A Space Odyssey

Im vorliegenden Kapitel wird die Einhaltung des ursprünglichen Blickwinkels bzw. der Umgang mit demselben diskutiert (Abschnitt 11.1). Nachfolgend wird die BS/BKM (bzw. ihre Herleitung) anhand von Gütekriterien für qualitative Forschung bewertet (Abschnitt 11.2) und abschließend mit Kodierschemata verglichen, die für ähnliche Zwecke entwickelt, im Allgemeinen aber postuliert und nicht in Daten gegründet wurden (Abschnitt 11.3).

11.1 Einhaltung des ursprünglichen Blickwinkels

In diesem Abschnitt soll diskutiert werden inwieweit der ursprünglich formulierte Blickwinkel (siehe Abschnitt 6.1) eingehalten bzw. wo von diesem aus welchen Gründen abgewichen wurde. Hierzu wird auf die drei Teile des Blickwinkels separat eingegangen:

1. **Ausrichtung des Interesses:** Im Rahmen der Herleitung der BS wurde von dem ursprünglichen Interesse, die grundlegenden Aktivitäten der einzelnen Paarprogrammierer charakterisieren zu können, nicht abgerückt. Da allerdings zu Beginn der Untersuchungen keine „a-priori-Vorgaben" dahingehend gemacht wurden, was unter einer grundlegenden Aktivität (später Basisaktivität) zu verstehen sein soll, musste ein solches Verständnis im Rahmen der Analysen erarbeitet bzw. der Begriff Basisaktivität sukzessive mit Bedeutung gefüllt werden. Betrachtet man nun das Ergebnis, insbesondere die BKM, so kann man erkennen, dass die Konzepte nicht – wie man vielleicht hätte vermuten können – vereinzelt auftretende, „besonders relevante" Aktivitäten im Paarprogrammierungsprozess adressieren, sondern etwas, was man als Basisfunktionen[1] der stattfindenden Handlungen bezeichnen könnte (vergleiche auch mit den Abschnitten 9.1.1 und 9.1.2). Die BKM wurde so konzipiert, dass – zumindest prinzipiell – jeder, auf einer bestimmten Granularitätsstufe separierbaren Handlung ein Basiskonzept zugeordnet werden kann. Den Grund für dieses Design liefert der Blickwinkel.

[1] Nicht zu verwechseln mit dem Begriff Basisfunktion aus der linguistischen Gesprächsanalyse (siehe S. 410) – auch wenn es eine gewisse Verwandtschaft gibt.

Denn wenn er auch an keiner Stelle explizit fordert, dass Sitzungen flächendeckend kodiert werden können, wird doch das Ziel formuliert, „eine extensionale Beschreibung eines auf die PP bezogenen Aktivitätsbegriffes zu erhalten. Dieser soll einen Ausgangspunkt für spezialisierte, theoriebildende Untersuchungen zur Verfügung stellen“ (siehe S. 123).

Die Betrachtung der von der BS/BKM adressierten Phänomene als Basisfunktionen der stattfindenden Handlungen erklärt auch andere Eigenschaften der BS/BKM, beispielsweise warum Wissenstransfer nur eingeschränkt adressiert wird, also z.B. dass dasjenige Wissen, welches nur durch HCI-Handlungen „hervortritt“, nicht explizit erfasst wird. Die Basisfunktion einer HCI-Aktivität besteht eben – zumindest im Allgemeinen – nicht darin, Wissen zu transferieren (man betrachte z.B. Phänomene vom Typ *write_sth*). Diese Einschränkung bei der Betrachtung von Phänomenen bedeutet aber keineswegs, dass der Blick auf Handlungen in weiterführenden Untersuchungen nicht zu verschieben ist, sondern spiegelt nur wider, dass im Rahmen der Herleitung der BS/BKM Entscheidungen getroffen wurden, die mit dem „Ausklammern“ bestimmter Phänomentypen bzw. -aspekte gleichzusetzen sind. Nur diese Entscheidungen machten es aber möglich, zu einer handhabbaren Menge von Konzepten kommen zu können.

2. **Art der zu berücksichtigenden Phänomene:** Im Rahmen der Herleitung der BS wurde der ursprünglichen Vorgabe in Bezug auf die zu berücksichtigenden Phänomene weitgehend gefolgt. So wurden nur direkt beobachtbare Phänomene berücksichtigt. Dies bedeutet insbesondere, dass die handelnden Personen weitgehend als „Black Box“ betrachtet wurden. Allerdings wurde trotzdem versucht, Ergebnisse bestimmter mentaler Prozesse, insbesondere Erkenntnisse, als solche sichtbar machen zu können. Dies geschah vor allem deshalb, weil sich Erkenntnisprozesse im Rahmen der Untersuchungen – sicherlich nicht überraschenderweise – als wichtige Aktivitäten herausstellten. Allerdings wurden nur bestimmte, zumindest indirekt verbalisierte Erkenntnisse (oder Erkenntnisprozesse) als zu markierende festgeschrieben. In diesem Zusammenhang wurden Regeln formuliert, die das Markieren (und spätere Einbeziehen in eine Theoriebildung) intersubjektiv nachvollziehbar machen sollen (siehe z.B. Abbildung 7.11).

3. **Ergebnistyp:** Das Ergebnis der Untersuchungen besteht im Kern aus einer Menge von Konzepten (eben den Basiskonzepten) und ihren Beschreibungen, geht aber insgesamt deutlich darüber hinaus:

   – Mit den Konzepten zusammen wurden – obwohl dies durch den Blickwinkel nicht gefordert war – Strukturen auf diesen entwickelt, auf unterster Ebene durch Verb- und Objektklassen, daneben in Form von projektorientierten, prozessorientierten und universellen Konzepten und darüber hinaus durch die Hierarchie zwischen den Fassaden- und den Nicht-Fassaden-Konzepten (siehe S. 343 f). Diese Strukturen

erlauben einen geordneteren Umgang mit Konzepten, als es eine kaum strukturierte Menge zulassen würde.

- Die Basiskonzepte wurden zum Teil explizit durch so genannte primär charakterisierende Eigenschaften (siehe S. 127 f) ausdifferenziert (siehe z.B. S. 170 oder S. 252). Dies war nötig, um garantieren zu können, dass die zugehörigen Phänomene in den zur Analyse herangezogenen Daten identifiziert werden können, erlaubt aber darüber hinaus auch einen direkteren Einstieg in weiterführende, spezialisierte Analysen.
- Die BS umfasst nicht nur Konzepte und ihre Beschreibungen, sondern auch eine große Menge von Kodierregeln. Diese demonstrieren auf der einen Seite, in welcher Weise die Konzepte aus den Daten extrahiert wurden und wie Phänomene im Einzelnen voneinander differenziert werden können, liefern auf der anderen Seite aber auch eine Vielzahl von Anregungen dahingehend, wie über Kodierungen nachgedacht werden kann. Wie schon mehrfach ausgeführt sind die Regeln keineswegs als feststehend zu betrachten. Werden Regeln gebrochen oder geändert, sollte aber beleuchtet werden, aus welchen Gründen dies passiert.

Darüber hinaus sind auch die auf S. 126 f im Zusammenhang mit dem Blickwinkel formulierten Leitlinien eingehalten worden:

1. Die Menge der HHI-Konzepte ist mit ihren 61 im Wesentlichen auf 14 Objektklassen verteilten Konzepten noch so handhabbar, dass eine konsistente Anwendung möglich ist. Die Beschränkung auf eine solche Größe wurde im Wesentlichen durch zwei Vorgehensweisen möglich: Zum einen wurden bestimmte Phänomene ausgeklammert, zum anderen wurden die betrachteten Tätigkeiten wie auch die betrachteten „Dinge“, auf denen diese Tätigkeiten durchgeführt werden, unter nicht mehr als nötig spezialisierten Verben bzw. Objekten subsumiert.

2. Die BKM hat eine gleichförmige Struktur, die in unterschiedlichen Bereichen (Objektklassen) dieselben oder zumindest ähnliche Unterscheidungskriterien verwendet.

11.2 Validierung und Güte

Im Weiteren soll nun die Güte der in den Kapiteln 6 bis 9 präsentierten Ergebnisse erörtert werden. Da diese Ergebnisse ausschließlich unter Verwendung qualitativer Methoden erzielt wurden, können hierbei, wie Mayring [132, S. 140] für qualitative Methoden im allgemeinen bemerkt, nicht einfach die Maßstäbe quantitativer Forschung (im Wesentlichen bez. Validität und Reliabilität) übernommen werden. Mayring betont, dass bei der Beurteilung qualitativer Forschungsergebnisse nicht einfach Kennwerte errechnet werden können, sondern argumentativ vorgegangen werden muss. „Es müssen Belege angeführt und diskutiert werden, die die

Qualität der Forschung erweisen können [...], der Prozess der Begründbarkeit und Verallgemeinbarkeit der Ergebnisse [...] rückt in den Vordergrund.“ Er erläutert in diesem Zusammenhang – unter anderem unter Bezug auf [62; 108; 113] – sechs allgemeine Gütekriterien qualitativer Forschung [132, S. 144 ff]:

- **Verfahrensdokumentation**: Im Gegensatz zu den meisten quantitativen Studien, bei denen in der Regel ein Verweis auf die verwendeten (standardisierten) „Techniken und Messinstrumente“ ausreicht, muss das Vorgehen bei qualitativen Untersuchungen „bis ins Detail dokumentiert werden“.

- **Argumentative Interpretationsabsicherung**: Da Interpretationen auf der einen Seite eine „entscheidende Rolle“ in qualitativen Studien spielen, sich solche aber auf der anderen Seite „nicht beweisen“ lassen, müssen sie „argumentativ begründet werden“. Dabei muss das „Vorverständnis der jeweiligen Interpretationen adäquat sein“ sowie Brüche in der Interpretation erklärt und Alternativdeutungen überprüft werden.

- **Regelgeleitetheit**: Auch wenn qualitative Forschung offen „gegenüber ihrem Gegenstand“ sein muss, darf dies „nicht in ein völlig unsystematisches Vorgehen münden. Auch qualitative Forschung muss sich an bestimmte Verfahrensregeln halten, systematisch ihr Material bearbeiten.“

- **Nähe zum Gegenstand**: In der qualitativen Forschung wird die Gegenstandsangemessenheit dadurch erreicht, „dass man möglichst nahe an der Alltagswelt der betroffenen Subjekte anknüpft. [...] Inwieweit dies gelingt, stellt ein wichtiges Gütekriterium dar.“

- **Kommunikative Validierung**: „Die Gültigkeit der Ergebnisse, der Interpretationen kann man auch dadurch überprüfen, indem man sie den Beforschten nochmals vorlegt, mit ihnen diskutiert.“

- **Triangulierung**: In der qualitativen Forschung kann „die Qualität der Forschung durch die Verbindung mehrerer Analysegänge vergrößert werden“. Dies kann auf unterschiedlichen Ebenen passieren: „Verschiedene Datenquellen können herangezogen werden, unterschiedliche Interpreten, Theorieansätze oder Methoden. Triangulierung meint immer, dass man versucht, für die Fragestellung unterschiedliche Lösungswege zu finden und die Ergebnisse zu vergleichen.“

Betrachtet man die (Herleitung der) BS/BKM bzw. ihre Darstellung im vorliegenden Dokument in Hinsicht auf diese Kriterien, so kann man Folgendes festhalten:

- Das Analyseverfahren ist regelgeleitet und genau dokumentiert. Diese Dokumentation beginnt bei der detaillierten Beschreibung der verwendeten Methode, der GTM (Kapitel 3), sowie den vorgenommenen methodischen

Erweiterungen (Kapitel 5) und führt über die phänomenorientierte Beschreibung von Konzepten[2] (Kapitel 6 bis 9) hin zur beispielhaften Darstellung des sukzessiven Erkenntnisgewinns (Kapitel 10). Flankiert wird diese Beschreibung der Analysevorgänge durch eine detaillierte Erläuterung der verwendeten Daten und Datenerhebungsmethoden (Kapitel 4).

– Im Rahmen der in den Kapiteln 6 bis 9 erläuterten Untersuchungen wurden Interpretationen von Phänomenen in erster Linie dazu eingesetzt, Klassifizierungen durchzuführen. Hierbei wurde darauf geachtet, nicht nur darzulegen, wie etwas als Phänomen eines bestimmten Typs identifiziert wurde, sondern auch, welche Aspekte von Phänomenen nicht berücksichtigt wurden (siehe z.B. die Erläuterungen zu dem Wissen, welches von den UK adressiert wird (S. 239 ff) oder zu den perlokutionären Akten (Abschnitt 9.1.3)).

– Alle bis auf eine der analysierten Sitzungen wurden im Rahmen professioneller Softwareentwicklung in der Industrie durchgeführt. Bei der Ausnahme waren Studierende tätig, die sich am Ende ihrer universitären Ausbildung befanden. Sie bearbeiteten eine Aufgabe, die sich in einen größeren Entwicklungskontext einbettete, in dem sie im Vorfeld schon zusammen gearbeitet hatten. Darüber hinaus stand einem der beiden Entwickler während der Aufzeichnung seine gewohnte Entwicklungsumgebung zur Verfügung. Es kann also auch in diesem Fall von einer eher natürlichen als von einer Experimentumgebung gesprochen werden. Alle Untersuchungen fanden somit nahe an der Alltagswelt der betroffenen Subjekte statt.

– Auf eine kommunikative Validierung musste verzichtet werden, da auf die einzelnen Entwickler nach den aufwändigen, zeitintensiven Analysen nicht mehr zugegriffen werden konnte.

– In Hinsicht auf Triangulierung wurden, der Methode des theoretischen Samplings folgend, unterschiedliche Datenquellen verwendet (Datentriangulation). Außerdem wurden durch den Einsatz der Paarkodierung zumindest teilweise zwei unterschiedliche Interpreten in den Analyseprozess involviert (Forschertriangulation). In gewisser Weise wurde auch eine Form von Theorietriangulation durchgeführt, da die Daten gegen Ende der Untersuchungen auch aus Sicht der Sprechakttheorie betrachtet wurden. Allerdings muss hier hinzugefügt werden, dass dies erst dann geschah, als die

[2] Flick [62] weist darauf hin, dass „die Unsitte, den vollzogenen Interpretationsprozeß lediglich dadurch für interessierte Leser nachvollziehbar und auf seine Stimmigkeit beurteilbar werden zu lassen, daß der Forscher ‚illustrative' Zitate aus Interviews oder Beobachtungsprotokollen einfügt, [...] häufig beklagt worden" ist. Die hier vorgestellte Untersuchung geht allerdings nicht so vor, da sie ein so genanntes „kodifiziertes Verfahren der Datenanalyse" (siehe hierzu [36]), die Methode des ständigen Vergleichens, verwendet und dokumentiert. Beispiele werden in erster Linie dazu herangezogen, die erläuterten Interpretationen und Entscheidungen zu bebildern. Hierbei wird auch auf nichttypische Fälle eingegangen.

GTM-Analyse deutlich erkennen ließ, dass die Anwendung von bestimmten Aspekten dieser Theorien zumindest terminologisch hilfreich sein könnte.

Zu guter Letzt muss in diesem Abschnitt darauf hingewiesen werden, dass sich die hier präsentierten Ergebnisse nicht unbedingt ohne Ergänzungen vollständig lückenlos auf alle Formen von Paarprogrammierungssitzungen anwenden lassen. Schließlich wurden nur solche Entwicklungstätigkeiten betrachtet, in denen es darum ging, bestehende Applikationen zu erweitern. Sitzungen, in denen komplett neue Applikationen entwickelt werden oder solche, in denen es nur um Designentwicklung, Wissenstransfer, „Refactorings“, Durchsichten oder „Bugfixings“ geht, wurden nicht herangezogen. Allerdings gab es in den betrachteten Sitzungen längere Passagen, in denen diese Tätigkeiten praktiziert wurden.

11.3 Vergleich mit anderen Kodierschemata

Auch wenn die BS nicht dahingehend konzipiert wurde, um als (reines) Kodierschema eingesetzt zu werden, wird sie im vorliegenden Abschnitt mit Kodierschemata verglichen. Hierbei geht es weniger darum, die verschiedenen Schemata erschöpfend vorzustellen, als mehr darum, Unterschiede sowie Parallelen zur BS/BKM aufzuzeigen und deutlich zu machen, warum diese Schemata als Startpunkt für die im vorliegenden Dokument beschriebenen Untersuchungen eher ungeeignet gewesen wären. Berücksichtigt werden dabei nur solche Schemata, die in auf die PP fokussierten Untersuchungen zum Einsatz gekommen sind. Eine Ausnahme bildet das Schema von Mayrhauser und Lang („A Flexible Expandable Coding Scheme“ (AFECS), [128]), welches eine gewisse Popularität im Software Engineering insgesamt erlangt hat.[3] Eine Übersicht ist in Tabelle 11.1 zu finden.

Als erstes soll auf das Kodierschema von Mayrhauser und Lang [128] eingegangen werden. Es ist in mehreren Studien im Software Engineering zum Einsatz gekommen (siehe z.B. [29; 158]) und zielt auf Untersuchungen zum Programmverstehen (insbesondere bei der Softwarewartung) unter Verwendung der Protokollanalyse [59]. Hierfür stellt es eine große Anzahl von Konzepten zur Verfügung, die zur Analyse von „Think Aloud“-Protokollen eingesetzt werden können. Die Menge der Konzepte wurde auf Basis einer Reihe von Voruntersuchungen sowie bestehender Theorien entwickelt (siehe z.B. [129]). Sie lässt sich erweitern, da die einzelnen Konzepte aus mehreren Segmenten bestehen, deren Wertebereiche ergänzt werden können oder sogar müssen. Im Gegensatz zur BS/BKM wird Soloprogrammierung (SP) bzw. „Think Aloud“ bei der SP adressiert, also monologische und nicht dialogische Kommunikation. Dies macht dieses Kodierschema

[3] Laut Google Scholar wird der Artikel von Mayrhauser und Lang [128] in 47 anderen Artikeln/Veröffentlichungen zitiert (http://scholar.google.de (Abruf: 05.01.2013)), laut Microsoft Academic Search in 23 (http://academic.research.microsoft.com (Abruf: 05.01.2013)). Dies bedeutet aber keineswegs, dass das Schema von Mayrhauser und Lang tatsächlich in 47 bzw. 23 Studien eingesetzt wurde. Die Anzahl der Studien, in denen das Schema zur Anwendung gekommen ist, dürfte deutlich darunter im einstelligen Bereich liegen.

für den Einsatz im Rahmen der PP eher ungeeignet. So betonen Brinker und Sager [31, S. 9] im Rahmen ihrer Erörterungen zur linguistischen Gesprächsanalyse, dass die Kommunikationsrichtung (monologisch/dialogisch) „nicht einfach ein Kriterium neben anderen [ist], sondern ein Merkmal, das grundlegende Konsequenzen für die Theoriebildung und Analyse hat“. Darüber hinaus adressiert das Schema nur Äußerungen und keine sonstigen Handlungen. Vergleicht man das Schema mit der BKM, kann man trotz der unterschiedlichen Ausrichtungen an einigen Stellen Ähnlichkeiten feststellen. Beispielsweise adressieren beide Schemata den Umgang mit Zielen und Hypothesen. Ein detaillierter Vergleich der beiden Schemata kann evtl. hilfreich sein, um einige der Unterschiede zwischen SP und PP besser zu verstehen. Er wird nachfolgenden Studien überlassen.

Vier weitere Kodierschemata, die im Zusammenhang mit qualitativen bzw. qualtitativ-quantitativen Untersuchungen der PP zum Einsatz gekommen sind, wurden bereits in Abschnitt 2.2 angesprochen. In der Studie von Cao und Xu [41] wurde ein Schema verwendet, mit dessen Hilfe sich Aktivitätsmuster von Paarprogrammierern grob kategorisieren lassen (für Details siehe S. 37 f). Als Startpunkt für die im vorliegenden Dokument beschriebenen Analysen wäre dieses Schema allerdings nur sehr bedingt geeignet gewesen. Denn einerseits basiert es auf anderen (nicht auf die PP spezialisierten) Schemata, wie dem von Lim et al. [121] (“Computer System Learning“) und erlaubt somit keinen vorurteilsfreien Blick auf die (Basisaktivitäten der) PP, und andererseits werden die Konzepte im Artikel nicht detailliert genug erörtert, um sie im Rahmen einer Analyse einsetzen zu können. Darüber hinaus werden von den Elementen des Schemas zum Teil größere Abschnitte, in denen beide Entwickler abwechselnd zu Wort kommen, adressiert. Betrachtet man die Kategorien im einzelnen, so kann man aber gewisse Parallelen zu den Basiskonzepten erkennen. Beispielsweise existiert die Subkategorie „Formulating Strategy/Action“. Sie ähnelt – soweit man dies trotz ihrer knappen Beschreibung sagen kann – den Elementen der Klassen *step* und *strategy*, ohne hier allerdings eine Unterscheidung zu explizieren. Sie ist einer Kategorie mit dem Namen „Leader's activities“ zugeordnet, während die Konzepte der Klassen *step* und *strategy* unabhängig von einer „Leader“-Rolle vergeben werden können bzw. genau genommen sogar müssen. Die BKM ist hier also nicht nur differenzierter, sondern auch allgemeiner. Auch das in der Studie von Xu et al. [228] zur Anwendung gebrachte Kodierschema basiert auf existierenden (nicht auf die PP spezialisierten) Theorien („Constructivist Learning Theory“ [164] und „Bloom's taxonomy of the cognitive domain“ [20]). Es ist auf die Untersuchung von Lernprozessen ausgerichtet und wäre somit als Ausgangspunkt für die Herleitung der BKM zu spezialisiert gewesen (für Details siehe S. 40 f). Dasselbe gilt auch für das 2008 in der Studie von Bryant et al. [34] eingesetzte Kodierschema. In dieser Studie ging es um die Analyse des Rollenkonzepts der PP (für Details zur Studie siehe S. 39 f). Das Schema adressiert auf der Basis von Schemata von Pennington [160] sowie Good und Brna [83] Abstraktionsebenen von Äußerungen. Das von Bryant et al. [34] 2006 publizierte, weitgehend eigenhändig entwickelte Schema adressiert letztendlich die zu absolvierenden Teilaufgaben und nicht die

stattfindenden Aktivitäten. So hat es in gewissen Bereichen zwar Ähnlichkeiten mit der BKM – z.B. kennt es einen Subtask *Agree strategy/conventions*, bei dem es darum geht Entscheidungen zu fällen – adressiert bestimmte Aspekte, die im Zentrum der BKM stehen, aber gar nicht. So wird beispielsweise der gesamte Bereich des Wissens- oder Erkenntnistransfers von Bryant et al. nicht explizit angesprochen. Darüber hinaus ist auch dieses Schema im Artikel nicht so detailliert dargestellt, dass es problemlos eingesetzt werden könnte.

Zusammenfassend kann festgehalten werden, dass – abgesehen davon, dass das unverzerrte Widerspiegeln der wirklichen Vorgänge ein entscheidendes Entwurfsziel war – keines der hier beschriebenen Kodierschemata dafür geeignet gewesen wäre, um auf dessen Basis mit den im vorliegenden Dokument beschriebenen Analysen zu beginnen. Hierfür lassen sich zwei Hauptgründe ausmachen:

1. Die meisten dieser Schemata beruhen auf Theorien, die sich nicht direkt auf die PP beziehen. Somit wären die Analysen potentiell von vornherein in eine bestimmte Richtung getrieben worden.

2. Die meisten der Schemata zielen nicht auf eine Wiederverwendung. Dementsprechend werden sie in den Artikeln nur grob beschrieben.

Nichtsdestotrotz kann es hilfreich sein, diese Schemata bzw. die hinter ihnen stehenden Theorien in weiterführenden Studien über die PP zu beachten – z.B. um die theoretische Sensibilität in bestimmten Bereichen zu verbessern. Dies betrifft natürlich auch Schemata aus anderen Bereichen wie z.B. aus der Forschung zu kognitiven Aktivitäten bei der Softwareentwicklung (siehe beispielsweise [175; 204; 227]). Eine Übersicht über Kodierschemata, die im Rahmen von Untersuchungen im Software Engineering und unter Verwendung der *Verbal Protocol Analysis* durchgeführt worden, findet sich bei Hughes und Parkes [98].

Autoren (Artikel)/ ggf. Name	**Kontext**	**Forschungs-methodischer Hintergrund**	**Basierend auf existierenden Theorien?**	**Anvisierte Art der Phänomene**	**Anzahl der Konzepte/ Kategorien**	**Sonstiges/*Beispiel***
Mayrhauser und Lang [128]/ AFECS	Programmverstehen/ Mentale Modelle/ Softwarewartung	Protokollanalyse	Ja	„Think Aloud“ bei der SP	>61 (erweiterbar/ reduzierbar)	Sprechende Konzeptnamen (partiell); Kodes bestehen aus Segmenten/ *OPP.hyp.end.com*
Cao und Xu [41]	Aktivitätsmuster bei der PP	Protokollanalyse	Ja	Dialog bei der PP	5 (mit 12 Unterkategorien)	Sprechende Konzeptnamen, Konzepte werden aber nur grob beschrieben/ *Ask for opinion*
Xu et al. [228]/ Self Directed Learning Theory	Lernprozesse bei der PP	Protokollanalyse	Ja	Dialog bei der PP	4 * 6	Es werden zwei Schemata, die in Summe „Self Directed Learning Theory“ genannt werden, nacheinander angewendet/ *Analysis+Absorption*
Bryant et al. [33]	Kollaboration bei der PP	*Verbal Protocol Analysis*	Nein	Dialog bei der PP	12	Das Schema adressiert Teilaufgaben, nicht Typen von Kollaboration/ *Configure environment*
Bryant et al. [34]	Rollen im Paarprogrammierungsprozess/ Abstraktionsebenen von Äußerungen	*Verbal Protocol Analysis*	Ja	Dialog bei der PP	6	/ *RW* („Real world or problem domain“)
BS/BKM	PP allgemein	GTM	Nein	Dialog und Handlungen bei der PP	74 (erweiterbar/ reduzierbar)	Sprechende Konzeptnamen, Kodes bestehen aus Segmenten

Tab. 11.1: Übersicht über Kodierschemata, die bei Untersuchungen zur PP zum Einsatz gekommen sind (im Vergleich zur BS/BKM). Darüber hinaus ist das Schema von Mayrhauser und Lang [128] aufgeführt, da es eine gewisse Popularität im Software Engineering erlangt hat.

Kapitel

12 Beispiele für die Anwendung der BS/BKM

Beziehungen schaffen, am liebsten
zwischen allen Dingen der Welt.
Kurt Schwitters

Wie zu Beginn der Arbeit bereits erörtert, geht es in den dargestellten Untersuchungen darum, eine Grundlage für die konsolidierte, qualitative Erforschung des Prozesses der PP zu legen. Theoriebildende Analysen einzelner Aspekte der PP sollen explizit außen vor bleiben. Von dieser Festlegung soll auch im vorliegenden kurzen Kapitel, in dem es um Beispiele für die Anwendung der BS geht, nicht abgewichen werden. Es werden also keine Theorien oder Theoriefragmente vorgestellt, sondern es wird an einzelnen, isolierten Beispielen illustriert, in welcher Weise der Einsatz der BS dabei helfen kann, Blickwinkeln zu folgen bzw. Theorien zu erarbeiten. Dabei wird nicht das Ziel verfolgt, alle Bereiche der BS/BKM zu adressieren. Außerdem werden die Blickwinkel nur soweit ausgeführt, wie es für das Verständnis der Beispiele notwendig ist. Die Summe der Beispiele soll deutlich machen, in welcher Weise die BS/BKM dabei helfen kann, zu einer effektiven, konsolidierten Erforschung der PP zu kommen.

12.1 Beispiel: Wissenstransfer

Ein viel diskutierter Aspekt der Paarprogrammierung ist der durch die Verwendung der Praktik erreichbare Wissenstransfer (siehe Einleitung zu Kapitel 2). Er wird derzeit im Rahmen eines Promotionsvorhabens von Franz Zieris am Institut für Informatik der Freien Universität Berlin unter Verwendung der BS/BKM untersucht.[1] Die diesbezüglichen qualitativen Analysen stehen noch am Anfang, es kann aber bereits beispielhaft gezeigt werden, wie die BS/BKM im Rahmen der Untersuchungen zum Einsatz kommt. So wurde zu Beginn der Studie im Rahmen der Festlegung bzw. nachfolgenden Arretierung des Blickwinkels entschieden, sich in einem ersten Analyseschritt auf den primären illokutionären Akt (siehe Exkurs 6) des Erklärens und auf Bestandswissen zu konzentrieren, genaugenommen auf Phänomene, die in die Klasse *knowledge* (siehe Abschnitt 7.6.5) fallen. Danach wurde in der Sitzung PR3.1 nach Episoden gesucht, die in diese Klasse fallen,

[1] Diese Untersuchung findet in der Arbeitsgruppe Software Engineering im Rahmen eines DFG-geförderten Projektes (Deutsche Forschungsgemeinschaft; http://www.dfg.de (Abruf: 18.01.2013)) mit dem Titel „Prozesscharakterisierung der lokalen und der erweiterten verteilten Paarprogrammierung“ statt.

dass heißt nach inhaltlich zusammenhängenden Folgen von Phänomenen, die mit Elementen der Klasse *knowledge* zu annotieren sind. Diese wurden dann näher analysiert. In Abbildung 12.1 ist eine solche Episode inklusive erster, vorläufiger Analyseergebnisse dargestellt. Die Episode demonstriert drei zusammenhängende Wirkungsweisen der BS/BKM:

- Die BS/BKM ermöglicht es, direkt in die Kodierung interessierender Phänomene einzusteigen.[2] Dies führt zu einer schnellen Orientierung in den Videodaten.

- Die interessierenden Phänomene können dann zügig ausdifferenziert werden.

- Dadurch, dass die BS/BKM die nähere Differenzierung von Phänomenen weiterführenden Studien überlässt, besteht die Möglichkeit, Schwerpunkte selbst festzulegen/auszuwählen bzw. zu entdecken. So führten die im hier diskutierten Beispiel beleuchteten Eigenschaften dazu, dass der Aspekt „Steuerung des Wissenstransfers“ (zumindest vorläufig) in den Fokus der Untersuchung rückte. Franz Zieris beschreibt seinen Erkenntnisprozess an dieser Stelle wie folgt (Kontextinformationen finden sich in Abbildung 12.1):

  > „Bei der Äußerung ‚Das ist Swing‘ von *P2* war ich mir zunächst sicher, dass es sich um ein *explain_standard of knowledge* handelt (Typ *AK*: ‚Paraphrasieren von übermitteltem Wissen‘ – und zwar der Spezialfall, dass man die Äußerung des Partners zu Ende führt). Dann dachte ich mir, dass sich *P2* ja gar nicht sicher ist, und seine Äußerung eher eine Vermutung ist und kam auf *propose_hypothesis* (siehe Vorfahrtsregeln). Und gleich darauf kam mir dann der Fall ‚Fragen, die Antwortmöglichkeiten enthalten‘ in den Sinn, was die Äußerung als ein *ask_knowledge* qualifizierte. Drei Kodierungen waren mir nun aber doch zu viel und so suchte und fand ich die Abgrenzung zwischen *ask_knowledge* und *propose_hypothesis*: ‚Geht der Sprecher davon aus, dass sein Partner die gestellte Frage derzeit zuverlässig beantworten kann?‘ – ich finde ‚ja‘. Deswegen war *propose_hypothesis* aus dem Rennen. Eine Unterscheidung zwischen *explain_standard of knowledge* und *ask_knowledge* ist mir dann nicht mehr gelungen und bei der Suche, was diese Äußerung nun so besonders macht, stieß ich darauf, dass der Sprecher womöglich mit der Art der Beantwortung durch *P1* nicht zufrieden war. Er wollte schließlich die spezifische Technologie in Erfahrung bringen. Konsequenz für mich:

[2] Dies gilt vorrangig für Untersuchungen solcher Aspekte, die direkt aus der BS/BKM abgeleitet werden und für die die BKM unmittelbar erste geeignete Konzepte zur Verfügung stellt (siehe hierzu auch das Beispiel zum Erkenntnistransfer (Abschnitt 12.2)).

> Das Steuern oder Formen des Wissenstransfers ist hier ein spannendes Phänomen.“[3]

Der letzte Punkt der Liste demonstriert auch, wie die (nicht verpflichtenden) Kodierregeln dazu führen können, dass über bestimmte Phänomene genauer nachgedacht wird.

Achtung: Zu Beginn der Untersuchungen wurde die Praktik des Paarkodierens eingesetzt. Hiermit wurde z.B. erreicht, dass der anfänglich noch breit angelegt Blickwinkel (Wissentransfer allgemein) erst einmal wie oben beschrieben eingeschränkt werden konnte. Da die Untersuchungen wie erläutert im Rahmen eines Promotionsvorhabens stattfinden und dieses eine eigenständige Leistung darstellen sollte, wurde das Paarkodieren danach – zumindest vorerst – eingestellt.

12.2 Beispiel: Erkenntnistransfer

Die BS/BKM macht unter anderem darauf aufmerksam, dass es im Rahmen von Paarprogrammierungssitzungen neben dem oben angesprochenen Transfer von Bestandswissen zumindest eine zweite Form von Wissenstransfer gibt. Es handelt sich um die Übermittlung von Erkenntnissen, die einzelne Entwickler im Laufe einer Paarprogrammierungssitzung erlangen. Es ist plausibel anzunehmen, dass hierin ein Schlüssel dafür zu finden ist, warum in vielen Paarprogrammierungssitzungen schneller bessere Ergebnisse erzielt werden als in Solositzungen. Eine Untersuchung, die sich mit dem Transfer von Erkenntnissen beschäftigt, scheint also vielversprechend zu sein. Die BS/BKM stellt für eine derartige Untersuchung einen direkten Einstieg zur Verfügung, indem sie die Explizierung von Erkenntnissen – zumindest in Teilen – direkt (durch die Klasse *finding*) adressiert und in diesem Zusammenhang zwischen verschiedenen Erkenntnistypen unterscheidet (P, D, T und ihre Untertypen (siehe S. 252 ff)). Die Erkenntnistypen werfen darüber hinaus Fragen auf – z.B.: Welche Auswirkungen ergeben sich aus der im Vergleich zur SP potentiell erweiterten Wahrnehmungsfähigkeit von Ereignissen? Es könnte also zuerst einmal ein Blick auf Ereignisse vom Typ P (*perceived event*, siehe S. 252) geworfen werden. Bei einer solchen Untersuchung würde dann auch eine wie in Tabelle 8.3 auf S. 380 dargestellte Episode näher untersucht werden. Hier führt eine Erkenntnisäußerung des Observers vom Typ P dazu, dass letztendlich Zeit gespart wird – und dies, obwohl der Observer selbst eine solche Ersparnis allem Anschein nach gar nicht im Sinn hatte. Es könnte also ein neues Konzept oder eine Eigenschaft eingeführt werden, welches die Zeitersparnis adressiert. Bei der Untersuchung anderer Erkenntnisäußerungen könnte dann (zumindest vorerst) mit untersucht werden, in welcher Weise Zeitersparnis eine Rolle spielt.

[3] Eine nachträgliche, gemeinsame Analyse der Sitzung (Paarkodierung) führte dazu, dass letztendlich nur *explain_standard of knowledge* annotiert wurde.

Das Beispiel verdeutlicht einen weiteren Nutzen: Die BS/BKM bietet die Möglichkeit, potentiell interessante, direkt mit dem Prozess der PP zusammenhängende Aspekte zu Analysezwecken auszuwählen. Außerdem zeigen die hier dargestellten Überlegungen noch einmal, dass die BS/BKM nicht als Kodierschema konzipiert wurde. Sie hilft vielmehr dabei, zu für qualitative Analysen geeigneten Konzepten zu kommen.

12.3 Beispiel: Entscheidungsprozesse

Die BKM beschreibt unter anderem, welche unterschiedlichen Formen von Entscheidungsprozessen im Rahmen von Paarprogrammierungssitzungen durchlaufen werden (*design*, *step*, *strategy* etc.). Eine Untersuchung darüber, ob und in welcher Weise sich diese Entscheidungsprozesse bei verschiedenen Arten von Paarungen (z.B. Paarungen von Anfängern, Paarungen von erfahrenen Paarprogrammierern etc.) unterscheiden, könnte dabei helfen, besonders geeignete Einsatzszenarien für die PP zu finden bzw. Entwickler in Hinsicht auf einen effektiveren Einsatz der PP zu schulen. Einen ersten Ansatzpunkt könnten hierbei diejenigen Dialoge bilden, die im Zusammenhang mit taktischen Entscheidungen stattfinden, also solche, die um Äußerungen der Klasse *step* kreisen. Derartige Prozesse spielen nämlich – zumindest soweit dies bisher beobachtet werden konnte (siehe S. 188) – eine große Rolle bei der PP. Hierbei könnte z.B. folgenden Fragestellungen nachgegangen werden:

- Finden die Entscheidungsprozesse wirklich zu großen Teilen auf *step*-Ebene statt? Wenn ja, warum?
- Wie laufen diese Entscheidungsprozesse ab?

Wie schon bei den anderen Beispielen gesehen, eröffnet die BS/BKM die Möglichkeit, direkt in das Kodieren von Phänomenen, hier von Entscheidungsprozessen auf taktischer Ebene, einzusteigen. Insbesondere stellt sie Möglichkeiten zur Verfügung, Vorschläge oder vorschlagsähnliche Äußerungen unterschiedlicher Art zu identifizieren und näher zu untersuchen. So handelt es sich beispielsweise im in Tabelle 7.11 erläuterten Fall („Die Frage ist, ob ich mich da jetzt rantrauen will, ob wir uns da jetzt rantrauen wollen?“) um ein *propose_step* vom Typ *SO* (Orientierung suchen; siehe Tabelle 7.5). Das Ziel der Äußerung besteht allem Anschein nach weniger darin, einen Vorschlag zur Diskussion zu stellen, als Absicherung zu erlangen. In wie weit ein somit in den Daten gegründetes Konzept wie *assure* Bedeutung in Entscheidungsprozessen hat, könnte eine der Stoßrichtungen sein, denen bei der weiteren Untersuchung gefolgt wird – z.B. falls derartige Phänomene häufiger beobachtet werden können.

Genauso wie das Beispiel in Abschnitt 12.2 verdeutlicht auch dieses, dass die BS/BKM den direkten Einstieg in die Analyse eines – das Bild der PP vervollständigenden – Aspektes der PP ermöglicht und zeigt – zumindest beispielhaft

– auf welche Weise zu Ausdifferenzierungen oder Kategorisierungen gekommen werden kann. Darüber hinaus stellt die BS/BKM Mittel zur Verfügung, um auch solche Aspekte zu erfassen, die nicht im Fokus stehen, aber eng mit den im Fokus stehenden Phänomenen zusammenhängen. Beispielsweise werden Konzepte zur Verfügung gestellt, um einen den Entscheidungsprozess unterstützenden Wissenstransfer zu betrachten (siehe z.B. Episode in Tabelle 7.44). An dieser Stelle kann man erkennen, wie das Schichtenmodell (siehe Abbildung 1.1) idealerweise arbeitet: Untersuchungen zu unterschiedlichen Aspekten werden über eine gemeinsame „Sprache“ integrierbar. Hier z.B. Untersuchungen zu Entscheidungsprozessen mit solchen zum Wissenstransfer.

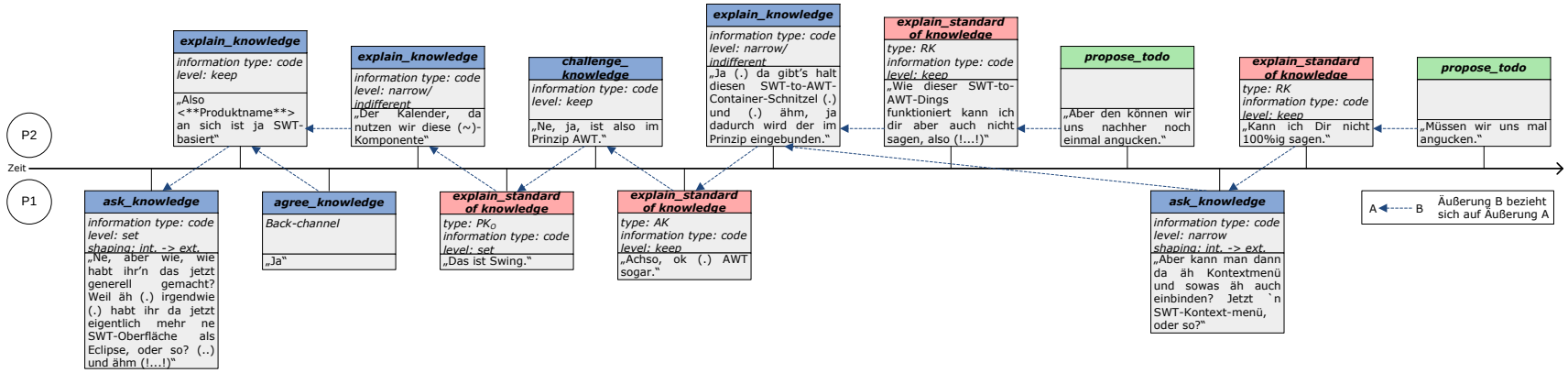


Abb. 12.1: Beispiel aus einer Untersuchung zum Wissenstransfer bei der PP. Die im Detail betrachtete Episode stammt aus den ersten Minuten der Sitzung PR3.1. Sie hat eine Länge von 54 Sekunden. Die dargestellte Kodierung wurde in den ersten Wochen der Analyse vorgenommen, gibt also den Untersuchungsstand zu einem sehr frühen Zeitpunkt wieder. In einem ersten Schritt wurden die Äußerungen mit Basiskonzepten annotiert. Danach wurde damit begonnen, die Annotationen mit Eigenschaften bzw. Werten von Eigenschaften auszudifferenzieren. Bei den Äußerungen vom Typ *explain_standard of knowledge* ist hierbei auf eine durch die BS/BKM zur Verfügung gestellte Klassifizierung (hier als *type* bezeichnet) zurückgegriffen worden (siehe Kapitel 7.6.3). Darüber hinaus wurden neue Differenzierungen eingeführt: Bei der durch die Eigenschaft *information type* vorgenommenen Differenzierung handelt es sich um eine Klassifizierung des übermittelten Wissens. Sie kann sowohl bei Fragen/Nachfrage wie auch bei Antworten verwendet werden. So bedeutet der Wert *code*, dass es sich um Wissen über den vorhandenen Programmkode (Struktur, verwendete Bibliotheken, etc.) sowie Wissen über das Fehlen von Kode handelt. Die Eigenschaft *level* adressiert das *relative* Abstraktionsniveau des Wissenstransfers und kann ebenfalls sowohl bei Fragen/Nachfrage wie auch bei Antworten verwendet werden. Die Werte von *level* bedeuten im Einzelnen: *set*: Niveau vorgeben oder erneut setzen; *keep*: Niveau beibehalten; *narrow*: auf einen Teilaspekt eingehen/eingrenzen; *expand*: auf einen größeren Themenkreis eingehen (z.B. eine Frage allgemeiner beantworten). Die Eigenschaft *shaping* ist bisher noch wenig präzise erfasst. Sie gehört zu Fragen bzw. Nachfragen, bei denen die Art der gewünschten Antwort mitgeteilt wird. So bedeutet der Wert *intensional* (*int.*), dass die Art des weiteren Wissenstransfers durch eine „begrifflich-inhaltliche Angabe“ vorgeben wird, der Wert *extensional* (*ext.*), dass dies durch konkrete Beispiele erfolgt. Der Wert *int. -> ext.* markiert einen entsprechenden Übergang. Die Entdeckung der Eigenschaften führte dazu, dass eine neue Fragestellung (bzw. ein neues Konzept) in den Fokus des Interesses rückte: *Steuerung des Wissenstransfers*. Eine ähnliche Abbildung wurde von Franz Zieris in einem Vortrag präsentiert.

Kapitel

13 Fazit und Ausblick

Wozu diese Vergeudung?
Matthäus 26, 8

Den Ausgangspunkt für die vorliegende Arbeit bildetet die Beobachtung, dass bisher zwar eine Vielzahl, meist quantitativer, den Prozess als „Black Box“ betrachtenden Untersuchungen der PP durchgeführt wurden, dass die Ergebnisse dieser Studien aber ein in vielen Punkten heterogenes Bild ergeben. Erklärungsmodelle für Effekte oder Diskrepanzen zwischen Ergebnissen werden kaum zur Verfügung gestellt. Auch wurde in keiner der bisherigen Studien versucht, eine allgemeine Theorie der PP zu erarbeiten oder Grundlagen für eine solche zu legen.

An dieser Stelle setzten die hier beschriebenen Untersuchungen an: Anstatt sich vorschnell auf bestimmte Teilbereiche (Effekte, Einsatzszenarien etc.) der PP zu fokussieren, sollte die Basis für ein breites Feld spezialisierter Untersuchungen geschaffen werden. Es sollten Methoden und Werkzeuge für effektive, konsolidierte, qualitative Analysen des Mikroprozesses der PP zur Verfügung gestellt werden, um in nachfolgenden Untersuchungen zu einem holistischen Verständnis der Praktik kommen zu können. Somit sollte es letztendlich möglich gemacht werden, die den Erfolg der PP beeinflussenden Faktoren zu identifizieren. Die Konsolidierung einzelner Ergebnisse sollte hierbei nicht nur dadurch erleichtert werden, dass alle Studien auf einer gemeinsamen Basis aufbauen, sondern auch dadurch, dass sie derart in so genannten Schichten organisiert werden, dass sie schon während des Analyseprozesses voneinander profitieren können (siehe Abbildung 1.1). Die zu entwickelnde Basis, das so genannte Rahmenwerk, sollte aus zwei Teilen bestehen (siehe S. 22 f),

- zum einen aus einer auf die Analyse von Aufzeichnungen von Paarprogrammierungsprozessen zugeschnittenen Untersuchungsmethode,
- zum anderen aus einem adaptiven System von Konzepten und Anwendungsregeln, um die den Prozess der PP formenden grundlegenden Aktivitäten der einzelnen Paarmitglieder konzeptualisieren zu können.

Das System von Konzepten, später Basiskonzeptmenge (BKM) genannt, sollte allerdings nicht ein Kodierschema im klassischen Sinn bilden. Vielmehr war es das Ziel, die BKM so zu konzipieren, dass sie in nachfolgenden spezialisierteren Untersuchungen – also solchen, die einzelne Aspekte wie zum Beispiel den Wissenstransfer bei der PP ins Visier nehmen – ausdifferenziert, verallgemeinert, erweitert, beschnitten, ergänzt oder zum Zweck der Verbesserung der theoretischen Sensibilität eingesetzt werden kann.

Die Ergebnisse der vorliegenden Arbeit lassen sich wie folgt zusammenfassen:

1. Es wurde eine auf der GTM nach Strauss und Corbin basierende qualitative Analysemethode für Videoaufzeichnungen von Paarprogrammierungssitzungen[1] entwickelt. Hierbei wurde die ursprüngliche Methode von Strauss und Corbin um vier Praktiken erweitert, ohne sie im Kern zu verändern (siehe Abschnitt 5.2). Diese vier zusätzlichen Praktiken sind darauf ausgerichtet, Problemen zu begegnen, die mit der Reichhaltigkeit von Videodaten zusammenhängen. Die Praktiken, insbesondere der Blickwinkel auf die Daten (siehe Abschnitt 5.2.1) und die Paarkodierung (siehe Abschnitt 5.2.4), haben sich nicht nur bei der Herleitung der BKM, sondern darüber hinaus bereits in weiterführenden Untersuchungen des Mikroprozesses der PP (also bei der Analyse weiterer Schichten) als hilfreich erwiesen (siehe z.B. S. 439) und sollten auch bei nachfolgenden Studien verwendet werden. Da sie nicht direkt auf die Analyse der PP ausgerichtet sind, haben sie das Potential, auch bei Untersuchungen in anderen Themenfeldern gewinnbringend eingesetzt zu werden – insbesondere bei solchen, die mit komplexem Videomaterial arbeiten.

2. Mit der Basisschicht (BS), die aus

   - der BKM (im Wesentlichen 69 Konzepte[2], verteilt auf zwei Hauptklassen: HHI (*human-human interaction*; siehe Abschnitt 6.5.1 und Kapitel 7) und HCI/HEI (*human-computer interaction/human-environment interaction*; siehe Abschnitt 6.5.2 und Abschnitt 8.1) sowie
   - einer großen Anzahl von Anwendungsregeln besteht,

   wurde ein Grundgerüst entwickelt, um direkt in die qualitative Analyse spezifischer Aspekte der PP einsteigen zu können, ohne Gefahr zu laufen, den Daten ungeeignete Konzepte oder gar Theorien „überzustülpen“. Der Hauptgrund für diese Eigenschaft liegt in der Verwendung der GTM bei der Herleitung der BS/BKM. Durch diese Maßnahme wurde sichergestellt, dass nur solche Konzepte in die BKM eingehen konnten, die zu Paarprogrammierungsdaten passen und relevante Phänomene beschreiben. Darüber hinaus zeichnet sich die BS/BKM durch folgende Eigenschaften aus:

   - Sie ist in erster Linie auf die Analyse verbaler Äußerungen ausgerichtet. Hier wird der Blick in vielen Situationen auf die illokutionären Akte von Sprechhandlungen (siehe Exkurs 6) gelenkt. Unterschiedliche Typen von Akten werden dabei unter einem Begriff subsumiert (siehe z.B. Tabelle 7.5). Nur so war es möglich, zu einer handhabbaren Menge von Basiskonzepten zu kommen.

[1] Ton + Webcam-Aufzeichnung der Probanden + Desktopaufzeichnung (siehe Abbildung 4.2)
[2] Über diese Konzepte hinaus enthält die BS/BKM noch fünf deskriptive Kodes (siehe Abschnitt 8.2).

- Beim Erfassen von Sprechhandlungen fokussieren sich die HHI-Konzepte besonders auf die Aspekte Programmdesign/Artefaktgestaltung, Sitzungssteuerung sowie Wissens- und Erkenntnistransfer (siehe Abbildung 7.3).
- Mit den HHI-Konzepten wird auch das Ziel verfolgt, gewisse Zusammenhänge zwischen Äußerungen festhalten zu können (siehe auch S. 161).
- Die BKM stellt mit den so genannten *activity*-Konzepten (siehe Abschnitt 7.7) eine Möglichkeit zur Verfügung, Äußerungen zu adressieren, die sich direkt auf HCI/HEI-Handlungen beziehen.
- Mit den HCI/HEI-Konzepten werden in der Regel nicht einzelne Aktionen, sondern inhaltlich zusammengehörige, hintereinander ausgeführte Tätigkeiten adressiert. So werden beispielsweise mit dem Konzept *write_sth* nicht Eingaben separater Zeichen oder Zeichenketten, sondern hintereinander ausgeführte, auf ein bestimmtes taktisches Ziel ausgerichtete Editiervorgänge erfasst (siehe auch S. 369).
- Die BS stellt eine große Anzahl von Kodierregeln zur Verfügung. Hiermit werden zwei Ziele verfolgt: Zum einen soll nachvollziehbar gemacht werden, welchen Regeln bei der Herleitung der BKM gefolgt wurde. Zum anderen geht es darum, herauszustellen, dass es sich bei der Anwendung der Basiskonzepte nicht in jeder Situation um einen kanonischen Prozess handelt bzw. vom Untersuchenden Entscheidungen gefällt werden müssen, wie mit bestimmten Phänomenen nutzbringend umgegangen werden soll. Die durch die BS vorgegebenen „Regeln“ geben hierbei Leitlinien für unterschiedlichste Situationen vor. Diese sollten aber in weiterführenden Untersuchungen in Hinsicht auf ihre jeweilige Angemessenheit hinterfragt werden.
- Viele Punkte werden von der BS/BKM bewusst offen gelassen. Dies geschieht vor allem dann, wenn Details erarbeitet werden müssten, deren Gestalt potentiell nicht unerheblich mit der Ausrichtung weiterführender Studien zusammenhängt oder die aller Voraussicht nach nur in wenigen Studien von Interesse sein werden. So ist beispielsweise der Umgang mit den so genannten Back-channels (siehe z.B. S. 176) oder der Umgang mit Bestandswissen, welches im Rahmen von Erkenntnissen mit artikuliert wird (siehe z.B. S. 270), nicht für jede zukünftige Studie gleichermaßen von Interesse.
- Bestimmte Punkte werden von der BS/BKM nicht adressiert – zum Teil aufgrund des behavioristischen Ansatzes der Untersuchungen (siehe S. 124), zum Teil zur Beschränkung des Umfangs. Dies betrifft nicht nur nähere Ausdifferenzierungen von Phänomenen. So werden beispielsweise implizite Vorschläge – hiermit sind solche gemeint, die nicht verbal artikuliert werden, sondern bestenfalls aus HCI/HEI-Handlungen abgelesen werden können – nicht erfasst. Das gleiche gilt

für impliziten Wissenstransfer, also solchen, der nicht über Äußerungen (oder zum Zwecke des Erklärens angefertigte Zeichnungen und Skizzen) von statten geht. Auch hier wird es weiterführenden Studien überlassen, Ergänzungen vorzunehmen.

Dass die BS/BKM ihre „Feuerprobe“ hinter sich gebracht hat, zeigen die Masterarbeit von Franz Zieris [231] sowie die ersten Ergebnisse der Untersuchungen zum Wissenstransfer (siehe Abschnitt 12.1). Der Weg für eine effektive, konsolidierte, qualitative Untersuchung des Mikroprozesses der PP ist also frei.

Anhang

A Zusammenfassung

Aufgrund des hohen Interesses an agilen Entwicklungsmethoden ist die *Paarprogrammierung* (PP), bei der sich zwei Softwareentwickler einen Computer und somit in der Regel auch eine Tastatur/Maus teilen, um zeitgleich zusammen Artefakte zu bearbeiten, zu einem beachteten Forschungsthema geworden. Der weitaus überwiegende Teil der bisher durchgeführten empirischen Studien verwendete allerdings quantitative Methoden, in denen die PP als „Black Box“ betrachtet wird. So liefern diese Untersuchungen zwar Zahlen, z.B. bez. der Performance der Entwickler, können aber weder deren Zustandekommen noch Diskrepanzen zu anderen vergleichbaren Studien erklären. Um diese große Lücke schließen zu können, müsste die interne Prozessstruktur, der Mikroprozess der PP, analysiert werden, was bisher nur in wenigen Fällen versucht wurde. Hier setzt die vorliegenden Arbeit an. Anstatt sich auf bestimmte Teilbereiche der PP zu fokussieren, wird die Basis für ein breites Feld spezialisierter, qualitativer Untersuchungen geschaffen. Mit dem sogenannten Rahmenwerk werden Methoden und Werkzeuge für effektive, konsolidierte, qualitative Analysen des Mikroprozesses der PP zur Verfügung gestellt. Im Wesentlichen besteht es aus zwei Teilen. Zum einen wird eine auf die Analyse von Videoaufzeichnungen von Paarprogrammierungssitzungen zugeschnittene qualitative Untersuchungsmethode bereitgestellt. Diese basiert auf der Grounded Theory Methodology nach Strauss und Corbin, die um vier Praktiken erweitert wurde, ohne sie im Kern zu verändern. Zum anderen besteht es aus der sogenannten Basisschicht, einem adaptiven System von 74 Konzepten und zahlreichen Anwendungsregeln, um die den Prozess der PP formenden grundlegenden Aktivitäten der einzelnen Paarmitglieder konzeptionell erfassen zu können. Das System von Konzepten bildet kein Kodierschema im klassischen Sinn. Vielmehr ist es derart konzipiert, dass es in nachfolgenden spezialisierten Untersuchungen – also solchen, die einzelne Aspekte wie zum Beispiel den Wissenstransfer bei der PP ins Visier nehmen – ausdifferenziert, verallgemeinert, erweitert, beschnitten, ergänzt oder zum Zweck der Verbesserung der theoretischen Sensibilität eingesetzt werden kann. Es adressiert sowohl sprachliche Handlungen – zum Beispiel prozesssteuernde oder den Entwurf betreffende – wie auch sonstige Aktionen der Entwickler. Auch die Basisschicht selbst wurde unter Zuhilfenahme der erweiterten GTM entwickelt. Das Rahmenwerk bildet den ersten entscheidenden Schritt hin zu einem holistischen Verständnis der PP.

Anhang B

Transkriptionsregeln

Transkribieren ist „die Verschriftlichung menschlicher Kommunikation, meist auf Grundlage von Tonband- oder anderen Aufzeichnungen“ [123]. Die Basis hierfür bildet ein Transkriptionssystem, also ein Regelwerk, welches genau festlegt, „wie gesprochene Sprache in eine fixierte Form übertragen wird“ [111, S. 43]. Es muss, anders als Schrift bzw. ein Schriftsystem, „nicht für kommunikative, sondern analytische Zwecke ausgebildet“ sein [171, S. 1038]. Beim Transkribieren soll, obwohl das Produkt des Vorgehens der „Situationsentbindung und Überlieferung im Wissenschaftsprozeß“ dient, „der fixierte Diskurs selbst [...] in seinen originalen, situationsgebundenen Mündlichkeitsmerkmalen so vollständig wie möglich durchsichtig bleiben. Präsentationsform und Inhalt treten auseinander und sind in eben diesem Auseinandertreten auf qualitativ höherer Stufe in eine ganzheitlich rezipierbare Einheit zu bringen“ [171, S. 1038].

Damit diese grundlegende Anforderung nicht bei jeder Untersuchung aufs Neue durch eigene Vereinbarungen ad hoc erfüllt werden muss und eine „größtmögliche Regelhaftigkeit und Zuverlässigkeit des Transkripts“ [172, S. 3] gewährleistet werden kann, wurden eine Reihe „konventionalisierter Transkriptionssysteme“ [171, S. 1038] entwickelt.[1] Neben der Verwendung verschiedener Notationen unterscheiden sie sich im Wesentlichen in ihrer Genauigkeit, das heißt darin, ob sie die verschiedenen Äußerungsmerkmale, wie z.B. Intonation oder nonverbale Kommunikation, überhaupt im Transkript berücksichtigen [111, S. 43].

Im deutschsprachigen Raum, insbesondere bei linguistischen Untersuchungen von Kommunikation, ist die „Halbinterpretative Arbeitstranskription“ (HIAT) weit verbreitet. Sie wurde von Ehrlich und Rehbein in den siebziger Jahren des letzten Jahrhunderts vorgestellt [55; 56; 57] und später u.a. von Rehbein für die computergestützte Analyse weiterentwickelt [172]. Die HIAT bildet den Ausgangspunkt für das im Rahmen der vorliegenden Arbeit verwendete Transkriptionssystem und hat folgende Eigenschaften:

– **Die Synchronizität der Ereignisse bleibt erhalten:** HIAT ist ein Partitursystem. Sprechakte, Kommentare etc. werden wie bei einer Partitur in der Musik in untereinander stehenden, Sprechern zugeordneten Endloszeilen (Spuren mit unterschiedlichen Spurtypen) angeordnet. Erst nachgeord-

[1] Von Transkriptionsstandards kann an dieser Stelle allerdings nicht gesprochen werden [111, S. 43].

net werden auch Segmente (im Allgemeinen Äußerungen) identifiziert und markiert.

- **Es wird die literarische Umschrift verwendet:** In HIAT orientiert sich die Verschriftung an der Standardorthographie der in der Aufnahme gesprochenen Sprache, erfasst aber auch viele Phänomene (Aussprachebesonderheiten), die in dieser nicht berücksichtigt werden [172, S. 11]. Hierzu zählen z.B. das Verschlucken von Silben oder das Zusammenziehen von Wörtern, zumindest wenn diese Phänomene vereinzelt auftreten.

  Mit der literarischen Umschrift wird ein Mittelweg zwischen der so genannten phonetischen und einer streng orthographischen Transkription eingeschlagen. „Abweichungen von der Standardorthographie werden dann wiedergegeben, wenn sie vereinzelt auftreten und für die Interpretation des Transkriptes eine Rolle spielen könnten. [...] Die literarische Umschrift folgt selbst wieder bestimmten Regeln, die eingehalten werden sollten, damit abweichende Wortformen dennoch – z.B. für eine computergestützte Suche – möglichst vorhersagbar bleiben [172, S. 11]."

- **Es werden weiterentwickelbare Arbeitsversionen erstellt:** HIAT zielt, wie es der Name schon andeutet, darauf, dass Artefakte erzeugt werden, die über die Zeit in grundsätzlich unterschiedlichen Arbeitsschritten – Verfeinerungen ebenso wie Bereinigungen von Kontingenzen – oder bezüglich verschiedener Fragestellungen zahlreiche Korrekturfassungen aufweisen können, „ohne Verlust an Information und ohne einen systematischen Verfahrenswechsel [171, S. 1047]".

- **Der Transkribent muss eine gewisse hermeneutische Leistung erbringen:** Im Laufe des Transkriptionsprozesses werden Einheiten ermittelt, das heißt das Gehörte wird nach Wörtern und Äußerungen segmentiert. Bei dieser Tätigkeit, besonders beim Identifizieren einzelner Äußerungen, handelt es sich nicht um einen schematischen, sondern um einen interpretativen Vorgang, „bei dem TranskribentInnen ihr Wissen über kommunikative Strukturen auf den zu transkribierenden Diskurs anwenden [172, S. 20]".

Während Transkripte in vielen Untersuchungen die Grundlage für die angestrebten Analysen bilden, hatten sie im Rahmen der hier geschilderten Studien nur eine nachgeordnete, unterstützende Funktion. Dies liegt daran, dass frühzeitig erkannt wurde, dass es schwierig, wenn nicht sogar unmöglich sein würde, ein Transkript zu erstellen, welches detailliert genug ist, um den Analyseanforderungen in jeder Hinsicht gerecht werden zu können. So schien es beispielsweise kaum praktikabel zu sein, alle Bildschirminhalte bzw. alle Veränderungen von Bildschirminhalten über die gesamte Sitzungszeit hinweg hinreichend vollständig und nachvollziehbar zu erfassen. Als Konsequenz aus dieser Einschätzung wurde beschlossen, die Analyse direkt auf dem Videomaterial durchzuführen (siehe

Kapitel 5) und Transkripte einzelner Äußerungen oder Episoden nur noch gezielt für folgende Zwecke anzufertigen:

1. Als unterstützende Maßnahme bei der Klärung spezieller Fragen, wie z.B. bei der Konzeptualisierung schwierig zu deutender oder besonders komplexer bzw. langer Dialoge.

2. Als Beispiele für die schriftliche Erläuterung von Konzepten und Konzeptklassen.

Unter besonderer Berücksichtigung dieser Einsatzszenarien wurde ein eigenes Transkriptionssystem, die so genannte pragmatische halbinterpretative Transkription (PHIT), entwickelt, welche sich an die HIAT anlehnt, allerdings einige entscheidende Unterschiede aufweist:

1. PHIT ist kein Partitursystem. Vielmehr werden Segmente (Äußerungen) im Zeilenformat dargestellt (siehe z.B. Tabelle 7.15), gelegentlich auch im Spaltenformat (siehe z.B. Tabelle 7.54). Hierdurch kann es passieren, dass „die faktischen Verhältnisse der kommunikativen Kontinuität und Synchronizität selegierend verdeckt [werden], ohne zurückverfolgbar zu sein (abgesehen vom Hineinhören in den O-Ton [...]) [171, S.1047]“.

   Der Grund für den Verzicht auf Spuren ist der Aufwand, den eine Partiturtranskription potentiell verursacht.[2] Dieser steht in keinem Verhältnis zum Nutzen, der im Rahmen der Entwicklung der BS/BKM aus einer solchen gezogen werden kann.[3]

2. Bei einer Transkription mit der PHIT wird, wie bei einer mit der HIAT, eine literarische Umschrift verwendet. Die PHIT ist hier allerdings pragmatischer als die HIAT. So wird in der Regel nicht zwischen vereinzelten oder regelmäßigen Abweichungen von der Standardorthographie unterschieden, sondern versucht, die Äußerungen genauso wiederzugeben, wie sie gemacht wurden (z.B. auch inkl. des Nichtsprechens des Schwas oder des Übersprechens von Wortgrenzen). Auf so genannte Sprechertabellen [172, S. 13] mit Angaben zu durchgehenden phonetischen Charakteristika wird hingegen verzichtet. Das durchgehende Auftreten einzelner Charakteristika eines Dialektes wird allerdings nicht berücksichtigt.

   Zwei Gründe sprechen für ein solches Vorgehen:

[2] Nach Kuckartz [111, S. 42] hängt der Aufwand für Transkriptionen in erster Linie vom gewählten Genauigkeitsgrad ab. „Auch für einfache Transkriptionen beträgt der benötigte Zeitaufwand etwa das fünf- bis zehnfache der Interviewzeit“. Redder beurteilt einen Aufwand von 1:60 bei der Anwendung von HIAT als gut [171, S. 1047f], wobei sie für die Erstellung von Rohtranskripten von 1:30 ausgeht.

[3] Während der Herleitung der BS/BKM entstand nur wenige Male der Bedarf, komplizierte „Verhältnisse der kommunikativen Kontinuität und Synchroniziät“ explizit (schriftlich) sichtbar zu machen.

(a) Potentiell können einzelne Auffälligkeiten und Abweichungen auf für den PP-Prozess relevante Eigenschaften von Äußerungen hinweisen. Eine streng orthographische Kodierung würde diese verdecken.

(b) Bei der Illustration der Elemente der BKM vermittelt nur eine möglichst genaue literarische Umschrift einen realistischen Eindruck davon, wie leicht oder schwierig es ist, einzelne Äußerungen „herauszuschneiden“ und zu konzeptualisieren.

3. In der Regel besteht die PHIT-Transkription einer Äußerung aus drei Teilen, der Nennung des Sprechers[4], der literarischen Umschrift des Gesagten und einem Kommentar in Textform, welcher ggf. Kontextinformationen[5] und weitere Angaben bzgl. Inhalt und Formulierung der Äußerung enthält. Der Kommentar kann auch dazu genutzt werden, Informationen bzgl. verdeckter Äußerungsüberlappungen oder Pausen zu liefern.

   Die Nennung des Sprechers erfolgt in der Regel, zumindest wenn es um die Erörterung von Beispielen geht, in Form eines Konzeptes bzw. Konzeptnamens (vgl. Abschnitt 5.2.2).

   Bei der Darstellung von Episoden werden die einzelnen Äußerungen tabellarisch untereinander dargestellt (siehe z.B. Tabelle 7.15), gelegentlich auch in Spalten nebeneinander (siehe z.B. Tabelle 7.54). Hierbei werden zum Teil auch konzeptuelle Angaben zu HCI/HEI-Aktivitäten gemacht (siehe z.B. Tabelle 8.3).

   Achtung: In den Beispielen dieser Arbeit werden zum Teil statt einem Konzept zwei (oder sogar drei und mehr) angegeben. Derart wird vorgegangen, wenn z.B. aus Platzgründen zwei (verschiedenartige) Äußerungen derselben Person zusammen erörtert werden müssen, wenn eine Doppelkodierung (bzw. Mehrfachkodierung) einer Äußerung notwendig ist oder sich die beiden Entwickler zeitgleich äußern. Im ersten Fall steht zwischen den Konzepten das Zeichen „+“ (siehe z.B. Tabelle 7.3), im zweiten das Zeichen „/“ (siehe z.B. Tabelle 7.19) und im dritten die Zeichenfolge „||“ (siehe z.B. Tabelle 8.5).[6] Kommen mehrere dieser Symbole zusammen zum Einsatz, werden Teilausdrücke zwecks Verbesserung des Verständnisses geklammert (siehe auch hier Tabelle 8.5). Außerdem kann es nötig werden, Kodierungen zu notieren, die sich über mehrere Tabellenzeilen (ggf. auch Tabellenspalten) erstrecken. Falls dieser Fall eintritt, wird in der Zeile (oder Spalte), in der die Kodierung beginnt, „Start <*Konzeptname*>“ und in der, in der die

[4] Die Sprecher werden anonymisiert. Falls nur eine einzelne Äußerung betrachtet wird, kann auf die Angabe des Sprechers verzichtet werden.

[5] Während die Kontextbeschreibung bei der Analyse oft nur eine untergeordnete Rolle spielt – schließlich steht ja auch das Video zur Verfügung –, ist sie im Rahmen einer beispielhaften Illustration von Konzepten und Konzeptklassen unerlässlich.

[6] Diese Symbole werden nicht nur im Zusammenhang mit Äußerungen (HHI), sondern auch bei Beschreibungen sonstiger Handlungen (HCI/HEI) verwendet.

Kodierung endet, „Ende <*Konzeptname*>“ notiert (siehe z.B. Tabelle 8.4) oder diese Phänomene werden in den entsprechenden Zeilen (oder Spalten) genauer beschrieben (siehe z.B. Tabelle 8.2).

4. Wie durch die HIAT können auch durch die PHIT Merkmale wie

   – Versprecher und Verbalisierungsschwierigkeiten, inkl. Reihungen, Repetitionen und Reparaturen,
   – Äußerungsüberlappungen, Pausen zwischen Äußerungen,
   – Sprechpausen und ihre Länge,
   – nicht vollständig ausgesprochene Worte und Verschlucken von Silben oder Buchstaben, insbesondere Tilgungen oder Veränderungen von Flexionsmerkmalen („ich hab“ vs. „ich habe“), Kürzungen im Auslaut („nich“ vs. „nicht“) und Reduktion/Assimilation („wir ham“ vs. „wir haben“),
   – Unverständliches bzw. schwer verständliche Äußerungen bzw. Äußerungsteile (inkl. Vermutungen),
   – uneigentliches Sprechen, wie z.B. Vorlesen von Bildschirminhalten,
   – Zusammenziehen von Wörtern und schneller Anschluss zwischen zwei Äußerungen,
   – Planungsindikatoren wie „äh“ oder „em“,
   – Abbrechen von Äußerungen,
   – Tonverlauf,
   – Modulation,
   – paraverbale Äußerungen,
   – Kontextinformationen, z.B. visuell wahrnehmbare Informationen wie nichtsprachliche Handlungen oder Unterbrechungen und
   – Kommentare des Transkribenten

   erfasst werden, wenn auch zum Teil mit Einschränkungen:

   – Überlappungen von Äußerungen (bzw. Turns) können nur durch Kommentare festgehalten werden.
   – Pausen zwischen Äußerungen (bzw. Turns) können nur durch Kommentare festgehalten werden.
   – Bei Sprechpausen innerhalb einer Äußerung können nur ungefähre Zeitangaben gemacht werden.
   – In Bezug auf die Modulation können nur sehr leise gesprochene Passagen markiert werden.

Explizit nicht berücksichtigt werden durch die PHIT Merkmale wie Sprechgeschwindigkeit (wie z.B. „schnell“), Sprechweise (wie z.B. „stakkato“), besondere Betonung und Dehnung. Falls sich das Auftreten solcher Merkmale in Zukunft als wichtig erweisen sollte, bleibt die Möglichkeit, diese in der Kontextbeschreibung anzuführen.[7]

Zur Markierung von Tonverläufen (Tonbewegungen) greift die PHIT auf die Notationen der HIAT zurück (siehe Tabelle B.2).[8]

5. Aufgrund des Verzichts auf Spuren müssen viele Äußerungsmerkmale bei Verwendung der PHIT anders als bei Verwendung der HIAT notiert werden. Aber auch spezielle, mit der PP zusammenhängende Phänomene, wie z.B. kurze Pausen, die im Zusammenhang mit dem gleichzeitigen Schreiben und Sprechen auftreten können, machen von der HIAT abweichende, in diesem Fall ergänzende Notationen sinnvoll. Außerdem wurden zur Verbesserung der Lesbarkeit in Teilen andere Symbole als bei der HIAT verwendet (z.B. im Zusammenhang mit „uneigentlichem Sprechen“ (siehe z.B. [172, S. 47])). Die Notationsformen der PHIT sind in Tabelle B.1 beschrieben.

6. Im Gegensatz zur HIAT wird die PHIT normalerweise erst dann angewendet, wenn bereits eine erste Segmentierung der Daten – durch Markierung so genannter Quotations im Video (siehe 5) – stattgefunden hat. Es kommt aber, ähnlich wie bei der HIAT [172, S. 20] vor, dass eine solche Segmentierung nachfolgend geändert wird. Dies hängt mit ihrem interpretativen Charakter zusammen. Diesen hat die PHIT mit der HIAT gemein.

   Achtung: Segmentierungen nach PHIT oder HIAT unterscheiden sich häufig. Während bei der PHIT oft mehrere Sätze und Satzfragmente zu einem Segment (Äußerung) zusammengefasst werden, wird bei der Anwendung der HIAT, speziell bei linguistischen Untersuchungen, in der Regel feiner segmentiert (siehe z.B. [172, S. 19f]).

7. Während mit der HIAT erstellte Transkripte auf eine spätere Weiterverwertung zielen, wird die PHIT nur zur Erreichung eines kurzfristigen Ziels eingesetzt. Durch Verwendung der PHIT soll also keine vollständige und auf den weiteren Wissenschaftsprozess zielende Datenrepräsentation zur Verfügung gestellt werden – insbesondere keine, auf der längerfristig weitergearbeitet werden kann. Somit kann auch erst während der Transkription einzelner Äußerungen bzw. Episoden entschieden werden, welche Merkma-

[7] Bei Betonungen und Dehnungen könnte auch auf die Notation der HIAT zurückgegriffen werden, bei Sprechgeschwindigkeit und Sprechweise sowie Modulation geht dies allerdings nicht ohne weiteres. Diese Merkmale werden nämlich bei der HIAT in eigenen Spuren abgebildet.

[8] Tonbewegungen wurden im Rahmen der hier dargestellten Untersuchungen nur in Einzelfällen, z.B. im Zusammenhang mit Ausdrücken wie „Hm“ oder „[A]hm“ markiert.

le im Rahmen des gerade anvisierten Ziels erfasst oder eben nicht erfasst werden sollten.[9]

Es muss festgehalten werden, dass mit der PHIT, so wie sie hier dargestellt ist, bei weitem nicht alle potentiell interessanten Phänomene erfasst werden können. Es ist zwar möglich, das „Kommentarfeld“ dafür zu verwenden, Spracheigenschaften zu beschreiben, die nicht direkt mittels Notationen der PHIT transkribiert werden können, besser ist es aber, die PHIT als erweiterbares System zu betrachten und zu verwenden.

[9] Natürlich darf dies nicht dazu führen, dass entscheidende Merkmale ausgelassen und somit die Ergebnisse der Untersuchung verfälscht werden. In der Regel sollten deshalb möglichst vollständige Transkripte erzeugt bzw. immer auch die Originaldaten, also z.B. die Videoaufzeichnungen, im Auge behalten werden. Dies gilt besonders im Rahmen des Analyseprozesses. Bei Transkriptionen zum Zweck der Illustration können und müssen hingegen momentan nicht im Fokus stehende Aspekte oft vernachlässigt werden. Denn hier würde eine Darstellung aller oder vieler Kontextinformationen meist den zur Verfügung stehenden Rahmen sprengen.

PHIT	Beschreibung	Kommentar/Beispiel
(.), (..), (...)	Kurze (bis zu einer Sekunde), mittlere (mehr als eine und bis zu zwei Sekunden) bzw. längere Sprechpause (mehr als zwei und bis zu drei Sekunden), in denen keine HCI- oder HEI-Aktivität stattfindet (weitere Punkte entsprechend).	In der Regel werden nur Sprechpausen innerhalb einer Äußerung auf diese Weise erfasst. Die verstrichene Zeit wurde im Rahmen der Herleitung der BS geschätzt.
(,), (, ,), (, , ,)	Analog zur vorhergehenden Erläuterung, nur dass der Sprecher HCI- oder HEI-Aktivitäten durchführt bzw. mit solchen fortfährt (weitere Kommata entsprechend).[a]	Siehe vorhergehende Erläuterung.
(*ÄT*)	Sehr leise/„in sich hinein“ gesprochener *ÄT*.	
<*paraverbale Äußerung*>	Paraverbale Äußerung.	<Lachen>
<**K**>	Ergänzungen/Kommentare (*K*) durch den Transkriptor.	Dient der Verbesserung der Verständlichkeit der Transkription durch Anreicherung mit Informationen aus dem Kontext.
<***E***>	Ersetzung (*E*) durch den Transkriptor zum Zwecke der Anonymisierung.	<**Entwicklername**>
(~), (~~)	Ein komplett unverständliches Wort bzw. mehrere komplett unverständliche Worte.	
(~*ÄT*)	Wahrscheinlicher Wortlaut eines schwer verständlichen Äußerungsteiles.	
⌣, _	Steht zwischen Worten oder Sätzen, falls der Sprecher diese in auffälliger Weise zusammen zieht bzw. schnell hintereinander ausspricht.	
(!...!)	Der Sprecher bricht mitten im Satz ab, ohne unterbrochen worden zu sein.	Sprechpausen müssen ggf. separat transkribiert werden.
(!!...!!), (!!*ÄT*!!)	Der Sprecher bricht mitten im Satz ab, weil er unterbrochen wird bzw. spricht weiter, obwohl der Partner zu sprechen beginnt.	Ein Spezialfall ist (!!(.)!!). Hier nutzt der Partner eine Sprechpause für einen (kurzen) Einwurf.
(#*ÄT*#)	Uneigentliches Sprechen, wie z.B. Vorlesen von Bildschirminhalten.	Außer dem Mitlesen gerade stattfindender Eingaben.
(##*ÄT*##)	Mitlesen gerade stattfindender Eingaben.	
[*Vokal*]	Stimmlos gesprochener Vokal.	
[...]	Nicht wiedergegebener Teil einer Äußerung.	

[a] Falls der Sprecher nicht sofort HCI- oder HEI-Aktivitäten durchführt, werden auch Mischformen wie (.., , ,) verwendet.

Tab. B.1: Notationen für die Verschriftlichung von Dialogen mittels der PHIT. *ÄT* kürzt den Begriff Äußerungsteil ab. Nicht für alle hier aufgeführten Notationen finden sich Anwendungsfälle in den in dieser Arbeit dargestellten Beispielen.

Notation	Beschreibung
$\acute{s}$	Steigender Tonverlauf in der Silbe *s*.
$\grave{s}$	Fallender Tonverlauf in der Silbe *s*.
$\check{s}$	Fallend-steigender Tonverlauf in der Silbe *s*.
$\hat{s}$	Steigend-fallender Tonverlauf in der Silbe *s*.
$\bar{s}$	Gleichbleibender (progredienter) Tonverlauf in der Silbe *s*.

Tab. B.2: Notationen für die Transkription von Tonbewegungen, übernommen aus der HIAT [172, S. 30f]. Tonbewegungen werden im Rahmen der vorliegenden Arbeit allerdings nur dann angegeben, wenn dies für das richtige Verständnis eines Beispiels unerlässlich scheint, z.B. bei Affirmationen wie „[Â]hm“. Hierbei erfolgt die Notation immer über dem ersten Buchstaben eines Wortes bzw. einer Silbe. Achtung: Nicht für alle hier aufgeführten Notationen finden sich Anwendungsfälle in den in dieser Arbeit dargestellten Beispielen.

Anhang C

Visualisierungen von ECG-Daten

Im Rahmen der Aufzeichnungen von Paarprogrammierungssitzungen wurden mittels des Eclipse-Plugins ElectroCodeoGram (ECG) Mikroereignisse protokolliert, z.B. Fokuswechsel innerhalb der IDE Eclipse, Fokuswechsel auf Applikationsebene oder Editiervorgänge in Eclipse (siehe S. 85). Aus diesen Ereignissen – die immer an ein im GUI dargestelltes Element, z.B. eine in Eclipse geöffnete Datei, einen Eclipseview oder eine Applikation gebunden sind – können in einem Nachbearbeitungsschritt Intervalle generiert werden, z.B. diejenigen Zeiträume. in denen der Fokus auf einem bestimmten View der IDE Eclipse lag. Um zügig einen Überblick über Programmiersitzungen erlangen zu können, wurde vom Autor des vorliegenden Dokumentes ein R-Programm mit dem Namen *stripes* entwickelt, mit welchem sich die derartigen Intervalle als farbig kodierte Rechtecke über einem Zeitstrahl visualisieren lassen. Um die Übersicht zu verbessern werden die Rechtecke hierbei auf folgende Spuren verteilt:

- **Spur 1 (Kommentarspur)**: In dieser Spur werden Sitzungsphasen dargestellt. Diese lassen sich allerdings nicht automatisch aus den ECG-Daten gewinnen, sondern müssen im Rahmen einer Durchsicht des Videomaterials eruiert und in einer CSV-Datei notiert werden. Das Programm *stripes* kann derartige Dateien parallel zu den ECG-Daten einlesen.

- **Spur 2 (`msdt.applicationactive.xsd`)**: In dieser Spur wird angezeigt, zu welchem Zeitpunkt der Fokus auf welcher Applikation lag. Die IDE Eclipse wird in dieser Spur nicht berücksichtigt. Dies geschieht in Spur 3. Achtung: Auch Fokusse auf der Aufnahmesoftware (Camtasia Recorder) werden in dieser Spur nicht angezeigt. Außerdem: Da mit den ersten Versionen des ECG-Plugins nur die IDE Eclipse beobachtet werden konnte, fehlt Spur 2 in der Abbildung C.1 zu Sitzung ST1.1.

- **Spur 3 (`msdt.windowactive.xsd`)**: In dieser Spur wird dargestellt, zu welchem Zeitpunkt der Fokus auf der IDE Eclipse lag. Achtung: Da diese Spur nur Ereignisintervalle eines Typs enthält, wird sie in den zu den Visualisierungen gehörenden Legenden nicht referenziert.

- **Spur 4 (`msdt.partactive.xsd`)**: Hier wird angezeigt, zu welchen Zeitpunkten sich der Fokus auf einem View der IDE Eclipse befand. Fokusse

auf Editoren werden in dieser Spur nicht berücksichtigt, sondern in Spur 5 angezeigt.

– **Spur 5 (`msdt.fileactive.xsd`)**: In der untersten Spur wird angezeigt, zu welchen Zeitpunkten sich welches Editorfenster bzw. welche geöffnete Datei im Fokus befand. In dieser Spur wird darüber hinaus visualisiert, ob und wie stark die Datei in einem Intervall bearbeitet wurde. Je weiter sich ein Rechteck unterhalb der Linie „no editing“ ausdehnt, desto mehr Editierschritte fanden im angezeigten Zeitraum statt. Achtung: Bei der Darstellung handelt es sich um die Angabe von absoluten Werten. Auf eine Achsenbeschriftung wurde aber verzichtet, da die durch das ECG gelieferten Werte nur ein grobes Abbild hinsichtlich der Anzahl von Editiervorgängen liefern (siehe auch S. 85).

Achtung:

– Natürlich kann aus einem Fokus auf eine Datei oder ein Element nicht geschlossen werden, dass sich das Paar nur mit dieser Datei oder diesem Element beschäftigt hat. Vielmehr kann es sogar sein, dass die Datei oder das Element über den gesamten Zeitraum in dem sie bzw. es sich im Fokus befand in keinerlei Weise vom Paar beachtet wurde.

– Fokusse auf Dialogfenster der IDE Eclipse (z.B. auf das Fenster zur Einstellung der „Preferences“ oder auf „Refactoring-Wizards“) werden nicht explizit visualisiert. Es wird lediglich angezeigt, dass sich der Fokus auf der IDE befand (siehe Abbildung C.2 13:45Uhr).

– In den Visualisierungen kann es dazu kommen, dass ein und die selbe Farbe in unterschiedlichen Spuren zur Bezeichnung unterschiedlicher Elemente oder Elementgruppen verwendet wird. Aus dem Kontext lässt sich ermitteln, um welches Element es sich handelt.

In den folgenden Abschnitten sind Visualisierungen der drei im Zentrum der Herleitung der BS stehenden Paarprogrammiersitzungen zu finden.

C.1 ST1.1: Experiment im Rahmen der Lehrveranstaltung „Bau betrieblicher Informationssysteme mit Java2 Enterprise Edition (J2EE)

In Abschnitt 4.2.1 wurde bereits auf die Sitzung ST1.1 (Experiment im Rahmen der Lehrveranstaltung „Bau betrieblicher Informationssysteme mit Java2 Enterprise Edition (J2EE)) eingegangen. Eine Visualisierung der aus den aufgezeichneten ECG-Ereignissen gewonnenen Episoden wird nun in Abbildung C.1

nachgereicht. Bezüglich der in der Abbildung dargestellten Phasen gilt das folgende:

1. **Vorbereitung (in Abbildung C.1 nicht dargestellt)**: Die Entwickler studieren jeder für sich allein die herausgegebene Aufgabe/das herausgegebene Aufgabenblatt. Danach sprechen sie kurz über eine erste Vorgehensstrategie (erst XPetStore modifizieren, danach das externe Programm schreiben) und diskutieren inhaltliche Fragen.

2. **Jk: JBoss konfigurieren (inkl. Erkunden von Dateien und Absprachen)**: Das Paar versucht herauszubekommen an welcher Stelle bzw. in welcher Weise der Applikation XPetStore das benötigte `Topic` (siehe Aufgabenblatt auf S. 500) zur Verfügung gestellt werden kann. Achtung: "On-the-fly" wird eine im Aufgabenblatt erwähnte Änderung am Kode vorgenommen (siehe Hinweis 2 im Aufgabenblatt (S. 501 im Anhang D)).

3. **FIe: Fehlermeldungen der IDE erörtern**: In der IDE Eclipse erscheinen fortwährend Fehlermeldungen („An error has occurred. See error log for more details."). Das Paar unterbricht seine Tätigkeit und wendet sich an den Veranstalter des Experiments. Zusammen wird ermittelt, dass die IDE allem Anschein nach Probleme beim Parsen von XML-Dokumenten hat. Das Paar beschließt, XML-Dokumente nicht mehr in Eclipse, sondern in einem externen Editor zu öffnen.

4. **Jk**: Die Entwickler schließen an die Tätigkeit aus Phase 2 an und beginnen den Applikationsserver JBoss zu konfigurieren - die XML-Konfigurationsdatei `jbossmq-destinations-service.xml` wird in einem externen Editor bearbeitet.

5. **FIe**: Die IDE Eclipse gibt weiterhin die schon bekannten Fehlermeldungen aus. Das Paar beschließt Eclipse neu zu starten. Dieses Vorgehen löst das Problem.

6. **JKu: JBoss Konfigurationsänderungen überprüfen**: Die Entwickler schließen an die Tätigkeit aus Phase 4 an und testen nun, ob die Konfigurationsänderungen zum gewünschten Ergebnis führen („Build" des Projektes und Inspektion des Ergebnisses). Es stellt sich heraus, dass dem nicht so ist.

7. **Jk**: Die Entwickler fahren fort, den Applikationsserver JBoss zu konfigurieren. Letztendlich wird die eben geänderte XML-Datei weiter editiert.

8. **JKu**: Die Entwickler testen ein zweites mal die Konfiguration (Ergebnis: Bisherige Konfigurationsänderungen führen nicht zum Erfolg).

9. **Jk**: Die Entwickler versuchen erneut herauszufinden, welche Änderungen sie wo vornehmen müssen, um zur gewünschten Konfiguration des Applikationsservers JBoss zu kommen. Die XML-Datei wird ein weiteres mal editiert. Dies geschieht diesmal nur, um festzustellen, ob derartige Änderungen überhaupt geeignete Auswirkungen haben können.

10. **JKu**: Die Entwickler testen die Konfiguration erneut (Ergebnis: Keine Auswirkungen zu erkennen.).

11. **Jk**: Die Entwickler fahren fort, zu eruieren, wie bzw. an welcher Stelle sie den Applikationsserver JBoss zu konfigurieren haben. Es werden weitere Konfigurationsdateien im externen wie auch im Eclipse-Editor geöffnet. Im Eclipse-Editor werden Änderungen an einer Properties-Datei (`build.properties`) vorgenommen.

12. **JKu**: Erneutes Testen der Konfiguration (Ergebnis: Konfigurationsänderungen führen zu Fehlermeldung beim „Build“ des Projektes).

13. **Jk**: Wieder eruieren die Entwickler, wie bzw. an welcher Stelle sie den Applikationsserver JBoss zu konfigurieren haben. Nun werden testweise Konfigurationsdateien aus dem Dateisystem gelöscht.

14. **JKu**: Wieder wird die Konfiguration getestet (Ergebnis: Keine verwertbaren neuen Erkenntnisse).

15. **Jk**: Ein weiteres mal eruieren die Entwickler, wie bzw. an welcher Stelle sie den Applikationsserver JBoss zu konfigurieren haben. Testweise kopieren sie die geänderte XML-Datei, die ihrer Meinung nach normalerweise während eines „Builds“ mittels XDoclet transferiert wird, händisch.

16. **JKu**: Wieder wird die Konfiguration getestet. Diesmal durch Neustart des JBoss (Ergebnis: Dem Paar wird klar, dass die Datei derzeit nicht kopiert wird. Unklar bleibt allerdings, warum dies nicht passiert.).

17. **Jk**: Noch einmal eruieren und erörtern die Entwickler, wie bzw. an welcher Stelle sie den Applikationsserver JBoss zu konfigurieren haben. Es wird eine Diskussion mit konträren Ansichten über grundlegende Eigenschaften von J2EE-Applikationen geführt. Die Meinungen werden anhand von Dokumentationen überprüft. Danach glaubt das Paar die Lösung zu kennen: Ihre bisherige Annahme, dass die von ihnen editierte XML-Datei im Laufe des „Build“-Prozesses zu kopieren ist, ist falsch. Vielmehr muss eine andere Inkarnation der Datei editiert werden. Die andere Inkarnation wird editiert (mittels eines externen Editors).

18. **JKu**: Die neue Konfiguration wird getestet. Wieder durch Neustart des JBoss (Ergebnis: JBoss ist wie gewünscht konfiguriert (ein `Topic` steht zur Verfügung)).

19. **Xa: XPetStore anpassen und Anpassungen überprüfen (inkl. Erkunden von Dateien und Absprachen)**: Es werde Änderungen an XPetStore (z.B. in der Klasse `MailerMDB`) vorgenommen. Hierbei wird auch darüber diskutiert, was an welcher Stelle zu ändern ist. In diesem Zusammenhang wird bestehender Kode erkundet.

20. **He: Hilfsapplikation auf Basis von `XPetStoreEmailSpy` entwickeln**: Um beim Testen nicht immer über die Weboberfläche des Shops gehen zu müssen, wird beschlossen, dass stattdessen die ursprüngliche Version der Applikation `XPetStoreEmailSpy` (siehe S. 500 im Anhang D) eingesetzt werden soll. Es wird eine Kopie der Klasse `XPetStoreEmailSpy` mit dem Namen `XpetstoreEmailSpySendToQueue` angelegt. `XPetStoreEmailSpy` wird dahingehend modifiziert, dass es mittels des XPetStore-`Topics` Mails versenden kann. Die Modifikation wird getestet. Anmerkung: Die Klasse wird im Weiteren nicht im angestrebten Sinn verwendet.

21. **Xa**: Es werden weitere Änderungen an XPetStore (z.B. in der Klasse `OMSUtil`) vorgenommen. Es wird versucht die Änderungen zu testen, indem das Webfrontend aufgerufen wird.

22. **FXn: Fehlermeldung im Webfrontend nachgehen**: Um die vorgenommenen Änderungen an XPetStore im Webfrontend der Applikation testen zu können, muss zuerst über das Webfrontend ein Benutzer angelegt werden. Das Anlegen führt zu einer Fehlermeldung. Dem Problem wird nachgegangen.

23. **U: Unterbrechung (Konsultieren des Veranstalters)**: Da nicht klar ist, wie es zu der Fehlermeldung kommt, wird der Veranstalter des Experiments konsultiert. Einer der Entwickler (*P2*) geht hierzu zum Veranstalter, der in einen Nachbarraum sitzt. Sein Partner wartet.

24. **FXn**: Der Entwickler *P2* kehrt kurz zum Arbeitsplatz zurück um etwas nachzusehen. Es kommt zu einer kurzen Absprache mit dem Partner.

25. **U**: Der Entwickler *P2* konsultiert noch einmal den Veranstalter. Sein Partner wartet. Der Veranstalter gibt dem Paar keinerlei Hinweise.

26. **FXn**: Dem Problem wird weiter nachgegangen. Letztendlich wird es gelöst.

27. **Xa**: Jetzt werden die Änderungen an XPetStore getestet. Der Test verläuft erfolgreich.

28. **EAi: Externe Applikation implementieren**: Auf Basis der Klasse `XPetStoreEmailSpy` wird die eigentlich gefragte Applikation entwickelt und getestet. Vorher wird eine Kopie der Klasse `XPetStoreEmailSpy` mit dem Namen `XpetstoreEmailSpySendToTopic` angelegt. Es wird

Dokumentation über die Verwendung von `Topics` herangezogen. Hierbei kommt es zu häufigen Wechseln zwischen der in einem Webbrowser aufgerufenen Information und der IDE Eclipse.

29. **U**: Da das Paar die IDE Eclipse neu starten will konsultiert der Entwickler `P2` noch einmal den Veranstalter. Sein Partner wartet.

30. **EAi**: Nach einem Neustart der IDE Eclipse wird die eigentlich gefragte Applikation weiter entwickelt und getestet. Es wird wieder Dokumentation über die Verwendung von `Topics` herangezogen. Nach mehreren Testläufen kommt das Paar zu einer korrekten Implementierung und schließt die Aufgabe somit erfolgreich ab.

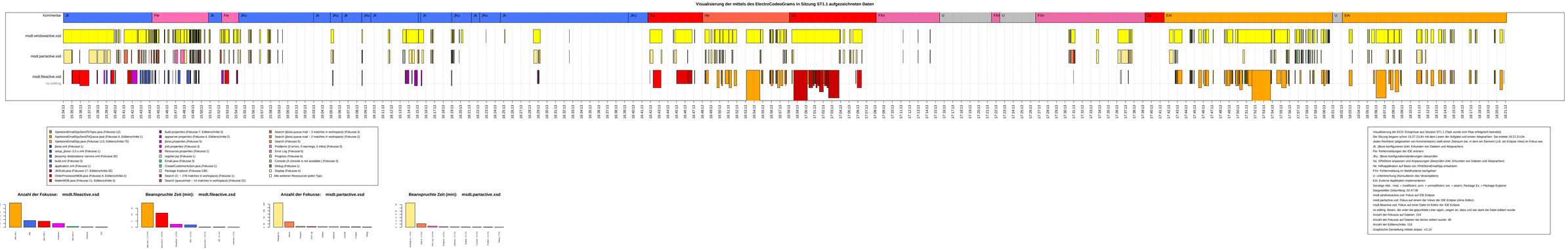


Abb. C.1: Darstellung des Ablaufs der Sitzung ST1.1 anhand der aufgezeichneten ECG-Daten (Erläuterungen zur Darstellung sind in der Einleitung zu Kapitel C zu finden). Die ECG-Daten werden durch Phasenbeschreibungen ergänzt (Kommentarspur – für Details siehe S. 462 ff.). Bezüglich der Spur `msdt.fileactive.xsd` gilt: Mit Orangetönen sind Java-Dateien markiert, die nicht zu XPetStore gehören. Es handelt sich um die zu entwickelnde Applikation `XpetstoreEmailSpy.java` (siehe S. 500) sowie zwei Kopien dieser, die zu unterschiedlichen Zeitpunkten angelegt wurden (`XpetstoreEmailSpySendToQueue.java`, `XpetstoreEmailSpySendToTopic.java`). Mit Rottönen sind Java-Dateien markiert, die zu XPetStore gehören und im Laufe der Sitzung editiert wurden, mit Grüntönen Java-Kassen von XPetStore, die zwar geöffnet aber nicht modifiziert wurden. Achtung: Da während der Sitzung Probleme im Zusammenhang mit der Anzeige von XML-Dokumenten auftraten, beschließt das Paar um 15:50Uhr XML-Dateien nicht mehr in der IDE Eclipse, sondern nur noch in einem externen Editor zu öffnen. Da die verwendete Version des ECGs keine externen Applikationen überwachen konnte, fehlen in der Darstellung somit eine Reihe von Ereignissen auf XML-Dokumenten.

C.2 PR1.1: Paarprogrammierungssitzung in einem Unternehmen im Bereich *Social Media*

In Abschnitt 4.2.2 auf S. 95 wurde bereits detailliert auf die Sitzung PR1.1 eingegangen. Eine Visualisierung der aus den aufgezeichneten ECG-Ereignissen gewonnenen Episoden wird nun in Abbildung C.2 nachgereicht. Bezüglich der in der Abbildung dargestellten Phasen gilt das folgende:

1. **Ad/Ek: Arbeitsvorbereitungen durchführen/Eclipse konfigurieren**: Das Paar passt – um die zu bearbeitende Funktionalität testen zu können – eines der zu modifizierenden PHP-Skripte dahingehend an, dass seine Rückgabewerte direkt in einem Webbrowser zur Anzeige gebracht werden können. Darüber hinaus nimmt das Paar Konfigurationen an der IDE Eclipse vor und verifiziert bestimmte Eigenschaften im Kode eines der zu modifizierenden PHP-Skripte.

2. **KvoFzv: Kode verbessern, ohne Funktionalität zu verändern**: *P1* nimmt, ausgelöst durch Fragen und Anmerkungen von *P2*, Verbesserungen an einem der Skripte vor, ohne die Funktionalität im Kern zu verändern (inkl. Tests (Aufrufe des vorbereiteten Skriptes im Webbrowser)).

3. **Au/InK: Anforderungen umsetzen/Implementierung neuer Kode**: Das Paar beginnt damit, die neuen Anforderungen umzusetzen – inkl. Verifikation von bereits vorhandenem Kode.

4. **U: Unterbrechung**: Unterbrechung durch eine dritte Person.

5. **Au/InK**: Das Paar fährt fort, die neuen Anforderungen umzusetzen – inkl. Verifikation von bereits vorhandenem Kode, explizitem Wissenstransfer, Tests und Kommentierungen im Quellcode. Die Phase wird dominiert von längeren Absprachen in Hinsicht auf Anforderungen und die weitere Gestaltung des Kodes.

6. **Ar: Änderungen rekapitulieren**: Das Paar rekapituliert die gerade am Kode vorgenommenen Änderungen. Dazu wird die *History*-Funktionalität der IDE genutzt.

7. **Au/InK**: Das Paar fährt fort, die neuen Anforderungen umzusetzen – inkl. kürzerer und längerer Diskussionen über Änderungen, Kommentierungen im Quellcode und Tests.

8. **Ek: Eclipse konfigurieren**: *P1* überprüft die Konfiguration von Eclipse. Er stellt fest, dass das Plugin PHPEclipse[1] fehlt.

[1] Siehe Fußnote 126 auf S. 261.

9. **ZaKe: Zu ändernde Kodestellen eruieren**: Das Paar eruiert, an welchen Stellen (Dateien) im bestehenden Applikationskode eine ihrer neu erstellten Funktionen aufgerufen werden muss bzw. Implikationen aus dem Einbau der Funktion beachtet werden müssen (Aufruf des Kommandos `grep` in einer Shell).

10. **Au/InK**: *P1* kommentiert Kode und ändert in Absprache einen Funktionsnamen.

11. **P/O: Pause/Off-Topic**: *P2* verlässt den Raum. Auch sein Partner stellt die Arbeit ein.

12. **Au/InK**: Das Paar (*P1*) fährt fort Kode zu kommentieren. Es schließt diese Tätigkeit mit einem Test ab.

13. **Au/AdNKe: Anforderungen umsetzen/Aufrufe des neuen Kodes einfügen**: Das Paar beginnt Aufrufe der von ihnen neu erstellten Funktion in den bestehenden Kode einzubauen - inkl. diverser Absprachen (auch über Implikationen und evtl. notwendige „Refactorings“).

14. **ZaKe**: Das Paar eruiert noch einmal auf andere Weise (Suche in Eclipse), an welchen Stellen im Kode eine ihrer neu erstellten Funktionen potentiell aufzurufen ist bzw. zu beachtende Implikationen aus dem Aufruf dieser Funktion entstehen könnten.

15. **FKe: Fremden Kode eruieren**: Das Paar versucht einen Kodeabschnitt zu verstehen, der nicht von ihnen selbst stammt.

16. **Au/AdNKe**: Das Paar führt eine Diskussion darüber, auf welche Weise der Aufruf der von ihnen neu erstellten Funktion erfolgen sollte bzw. welche Implikationen sich aus unterschiedlichen Optionen ergeben.

17. **ZaKe**: Das Paar eruiert ein weiteres mal, an welchen Stellen im Kode eine ihrer neu erstellten Funktionen potentiell aufgerufen werden muss (Wiederholung der Suche in Eclipse).

18. **Au/AdNKe**: Das Paar setzt seine Diskussion darüber, auf welche Weise der Aufruf der Funktion erfolgen sollte, welche Implikationen sich ergeben und welche „Refactorings“ evtl. durchgeführt werden sollten, fort.

19. **P/O**: Das Paar unterbricht die Arbeit kurz, um nachzusehen wie viel Zeit schon verstrichen ist.

20. **Au/AdNKe**: Das Paar setzt seine Diskussion darüber, auf welche Weise der Aufruf der Funktion erfolgen sollte bzw. welche „Refactorings“ evtl. durchgeführt werden sollten, fort. Dabei werden kleinere „Refactorings“ durchgeführt.

21. **ZaKe**: Das Paar eruiert ein weiteres mal, an welchen Stellen im Kode eine ihrer neu erstellten Funktionen potentiell aufgerufen werden muss bzw. Implikationen aus dem Einbau entstehen könnten (Wiederholung der Suche in Eclipse).

22. **Au/AdNKe**: Das Paar setzt seine Diskussion darüber fort, auf welche Weise der Aufruf der Funktion erfolgen sollte bzw. welche „Refactorings“ evtl. durchgeführt werden sollten. Die Ergebnisse der Diskussion werden umgesetzt.

23. **ZaKe**: Das Paar überprüft, ob sie alle zu bearbeitenden Stellen (Dateien) wirklich bearbeitet haben (`grep` in Shell und neue Suche in Eclipse).

24. **FT: Finale Tests**: Das Paar testet seine Implementierung final auf dem lokalen System. Hierzu fügen sie Kodezeilen ein, die das Testen erleichtern. Darüber hinaus werden kleinere „Refactorings“ durchgeführt und (in Zukunft vorzunehmende) Verbesserungen diskutiert.

25. **P/O: Pause/Off-Topic**: Das Paar erörtert kurz, ob nun eine günstiger Zeitpunkt wäre, die Aufzeichnung zu beenden.

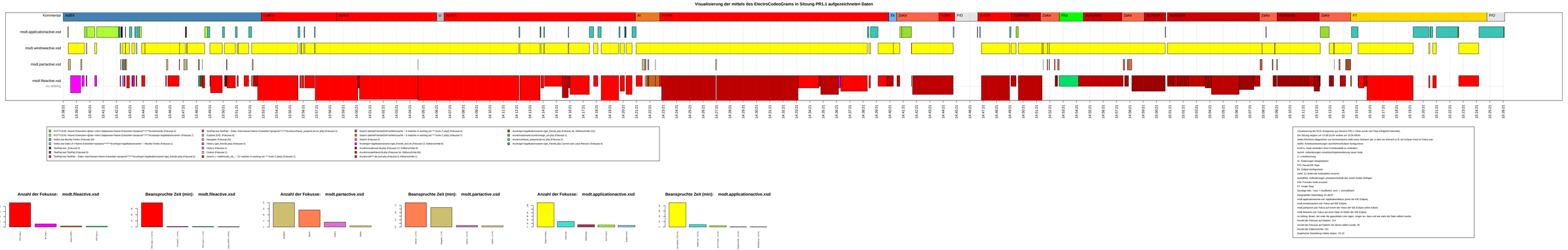


Abb. C.2: Darstellung des Ablaufs der Sitzung PR1.1 anhand der aufgezeichneten ECG-Daten (Erläuterungen zur Darstellung sind in der Einleitung zu Kapitel C zu finden). Die ECG-Daten werden durch Phasenbeschreibungen ergänzt (Kommentarspur – für Details siehe S. 469 ff.). Bezüglich der Spur `msdt.fileactive.xsd` gilt: Mit Rottönen sind PHP-Dateien markiert, die im Rahmen der Sitzung editiert wurden, mit Grüntönen solche, die zwar geöffnet, aber nicht modifiziert wurden. Orangetöne verweisen auf solche PHP-Dateien, die im Rahmen einer Änderungsnachverfolgung geöffnet wurden. Dateinamen bzw. Pfade sind gemäß eines Geheimhaltungsvertrages mit Firma 1 partiell unkenntlich gemacht worden (z.B. durch Verwendung des Platzhalters "`***`").

C.3 PR2.1: Paarprogrammierungssitzung in einem Unternehmen im Bereich Geoinformationssysteme

Auf S. 99 wurde bereits detailliert auf die Sitzung PR2.1 eingegangen. Eine Visualisierung der aus den aufgezeichneten ECG-Ereignissen gewonnenen Episoden wird nun in Abbildung C.3 nachgereicht. Bezüglich der in der Abbildung dargestellten Phasen gilt das folgende:

1. **Ad/S: Arbeitsvorbereitungen durchführen/Synchronisieren SVN/CVS**: Das Paar prüft, ob die vorliegenden Arbeitskopien bez. der „Repositories“ aktuell sind.

2. **Pe: Partner einführen**: *P1* führt seinen Partner in die Kodeänderungen ein, die er im Vorfeld vorgenommen hat. Diese Phase geht dadurch, dass *P2* eine Reihe von Fragen stellt, fließend in die nächste über.

3. **DuSe: Design und Strategie erörtern**: Das Paar erörtert Ideen zum Design sowie zu Strategien, diese Ideen umzusetzen. Die Diskussion verläuft kontrovers.

4. **Pe**: *P1* fährt mit der Einführung in die von ihm vorgenommenen Kodeänderungen fort (inkl. einer eigenständigen Kodedurchsicht der gerade erläuterten Klasse durch *P2*).

5. **KvoFzv: Kode verändern, ohne Funktionalität zu verändern**: *P2* nimmt eine Modifikation an der gerade von ihm inspizierten Klasse vor, ohne die Funktionalität im Kern zu verändern.

6. **Pe**: *P1* fährt mit der Einführung in die von ihm vorgenommenen Kodeänderungen fort. Dabei geht er kritisch auf die von seinem Partner vorgenommene Änderung ein. Danach setzt *P2* seine eigenständige Durchsicht der Klasse fort. Diese Phase geht wieder fließend in die nächste über.

7. **DuSe**: Das Paar setzt die Erörterungen von Ideen zum Design sowie zur Strategie, dieses umzusetzen fort. Auch diese Phase geht fließend in die nächste über.

8. **Pe**: *P1* fährt mit der Einführung in die von ihm vorgenommenen Kodeänderungen fort. Nachfolgend setzt *P2* seine Durchsicht fort. Wieder gibt es einen fließenden Übergang in die nächste Phase.

9. **DuSe**: Das Paar setzt seine Diskussion zum Design sowie zur Strategie, dieses umzusetzen fort. Es wird immer deutlicher, dass *P2* mit den Vorarbeiten des Partners nicht vollständig einverstanden ist. Es wird beschlossen eine Reihe von „Refactorings“ durchzuführen.

10. **KvoFzv**: *P1* führt die erörterten „Refactorings“ durch (anfangs unter Zuhilfenahme der *Refactoring*-Funktionalitäten der IDE, später händisch).[2] Sein Partner verlässt zu Beginn dieser Aktivitäten kurz den Arbeitsplatz. *P1* arbeitet in dieser Zeit weiter. Nach der Rückkehr von *P2* kommt es – während *P1* weiter mit Kodeumstellungen beschäftigt ist – zu einer Reihe von kurzen, kontroversen Diskussionen. Zeitweise sitzt *P2* allerdings auch weitgehend still neben seinem Partner. Dass er dem Geschehen trotzdem folgt, kann daran abgelesen werden, dass er zwischendurch immer wieder Fragen stellt oder Anmerkungen (auch zu Fehlern) macht. *P1* erläutert seine Tätigkeiten nur gelegentlich von sich aus.

11. **Vp: Vorgehen planen**: Das Paar plant die nächsten Arbeitsschritte.

12. **T: Testen**: Das Paar (*P2*) testet die bisher vorgenommenen Änderungen (inkl. einer längeren passiven Wartezeit beim Start der Applikation). Der Test führt zu einer Fehlermeldung.

13. **Db: Defekt beheben**: Das Paar sucht und beseitigt den Defekt, der zu der Fehlermeldung führte.

14. **T**: Das Paar (*P1*) testet noch einmal (diesmal erfolgreich).

15. **Ci: „Check in“**: Das Paar (*P2*) führt ein „SVN-Commit“[3] durch. Da das Paar die ID der gerade in Bearbeitung befindlichen Aufgabe nicht kennt, müssen sie das *Issue-Tracking-System* (Web-Applikation) aufrufen. Da sie darüber hinaus nicht wissen, in welchem Format die ID im SVN anzugeben ist, verlässt *P2* den Arbeitsplatz um an seinem Arbeitsplatz nachzusehen.

16. **Au/InK: Anforderungen umsetzen/Implementierung neuer Kode**: Das Paar (*P2*) beginnt mit der Implementierung von neuen Funktionalitäten (inkl. Erkundung von bestehendem Kode durch *P2*). Eine Reihe von Kodestellen werden allerdings erst einmal nur mit `TODO_NOW` markiert.

17. **T**: Das Paar (*P2*) testet die vorgenommenen Änderungen. Ergebnis: Die Funktionalität arbeitet nicht wie erwartet.

18. **Db**: Das Paar sucht den Defekt im Kode. Dann startet *P2* die Applikation im „Debug-Modus“. Die noch laufende Instanz der Applikation beendet er erst nach dem Start des „Debug-Moduses“. Das Paar ermittelt die Ursache für das Versagen und *P2* korrigiert den Kode.

19. **T**: Das Paar (*P2*) testet die vorgenommenen Änderungen erneut. Diesmal verläuft der Test erfolgreich.

[2] Ob hierbei nicht auch funktionsverändernde Modifikationen vorgenommen wurden, kann letztendlich nicht ausgeschlossen werden.

[3] Siehe Fußnote 66 auf S. 191.

20. **Ci**: Das Paar (*P2*) führt erneut ein „SVN-Commit“ durch.

21. **DuSe**: Das Paar erörtert die im Weiteren vorzunehmenden Änderungen. Hierbei werden unterschiedliche Klassen durchgesehen und Design- sowie Strategieoptionen diskutiert. Zu einer Umsetzung kommt es aber nicht.

22. **Pf: Paarprogrammierungssitzung finalisieren**: Da ein „Stand-up meeting“ ansteht, muss das Paar die Sitzung beenden. Sie sprechen ab, wann sie die Bearbeitung der Aufgabe fortsetzen wollen.

Abb. C.3: Darstellung des Ablaufs der Sitzung PR2.1 anhand der aufgezeichneten ECG-Daten (Erläuterungen zur Darstellung sind in der Einleitung zu Kapitel C zu finden). Die ECG-Daten werden durch Phasenbeschreibungen ergänzt (Kommentarspur – für Details siehe S. 475 ff.). Bezüglich der Spur `msdt.fileactive.xsd` gilt: Mit Rottönen sind Java-Dateien markiert, die im Rahmen der Sitzung editiert wurden, mit Grüntönen solche, die zwar geöffnet, aber nicht modifiziert wurden. Dateinamen bzw. Pfade sind gemäß eines Geheimhaltungsvertrages mit Firma 2 partiell unkenntlich gemacht worden (z.B. durch Verwendung des Platzhalters „`****`“).

Anhang

D Im Rahmen des BISJ2EE-Experiments ausgegebene Unterlagen

Auf den folgenden Seiten sind die im Rahmen des Experiments zur Lehrveranstaltung „Bau betrieblicher Informationssysteme mit Java2 Enterprise Edition (J2EE)“ ausgegebenen Unterlagen zu finden (für Details zum Experiment siehe Abschnitt 4.2.1). Im Einzelnen handelt es sich um folgende Dokumente:

- S. 482 - S. 488: Das auf das Experiment vorbereitende Aufgabenblatt. Es wurde eine Woche vor dem Experiment ausgegeben. In den Aufgaben geht es vor allem darum, dass die Studierenden ihren privaten Rechner so konfigurieren, dass er im Experiment verwendet werden kann.
- S. 489 - S. 495: Kurz nachdem das auf das Experiment vorbereitende Aufgabenblatt ausgegeben worden war, stellte sich heraus, dass es auf einigen Rechnern Probleme im Zusammenspiel zwischen den vorhandenen Webcams[1] und der für die Aufzeichnung vorgesehenen Software mediaCam[2] kam. Außerdem konnten Probleme mit der bereitgestellten Version des ElectroCodeoGrams[3] beobachtet werden. So wurde ein ergänzendes Aufgabenblatt ausgegeben. Hier wird erläutert, dass nicht mediaCam sondern Camtasia Studio[4] zu installieren ist. Außerdem wird darauf hingewiesen, dass eine neue Version des ElectroCodeoGrams einzuspielen ist.
- S. 496 - S. 501: Das eigentliche Aufgabenblatt, welches zu Beginn des Experiments ausgegeben wurde.
- S. 502 - S. 506: Der vor dem Experiment ausgegebene Fragebogen.
- S. 507 - S. 510: Der nach dem Experiment ausgegebene Fragebogen.

[1] Siehe Abschnitt 4.1.

[2] Die Software mediaCam ist eine „Screen Recording Software“ der Firma netu2 zur simultanen Aufzeichnung eines Bildschirms, eines Webcambildes sowie von Audiodaten (http://www.netu2.com/ (Abruf: 16.03.2012)).

[3] Siehe Abschnitt 4.1.

[4] Siehe Abschnitt 4.1.

Vorlesung "Bau betrieblicher Informationssysteme mit J2EE"
Freie Universität Berlin, Institut für Informatik, Arbeitsgruppe Software Engineering
Prof. Dr. Lutz Prechelt, Stephan Salinger
Übungsblatt 14A WS 2005/2006 2006-01-30

Tragen Sie hier Ihre **Teilnehmernummer** ein und merken Sie sich diese: ☐
Tragen Sie hier Ihren **Kameratyp** ein: ☐

Aufgabe 14A-1: (Infrastuktur für Experiment auf eigenem Rechner einrichten)

Vorbemerkung: In der KW6 (06.02 – 10.02) werden alle Teilnehmer der Veranstaltung an einem Experiment teilnehmen. Am Ende dieses Aufgabenzettels finden Sie den Termin Ihrer Teilnahme. **Bitte seien Sie auf jeden Fall pünktlich im angegebenen Raum und bringen Sie Ihren Laptop mit** (sofern Sie einen solchen besitzen - falls nicht, beachten Sie Aufgabe 14A-2). Die notwendigen Konfigurationen auf Ihrem Laptop werden im Folgenden beschrieben. **Bitte führen Sie diese sehr sorgfältig durch**.

Im Experiment wird es darum gehen, dass die Teilnehmer eine Programmieraufgabe bearbeiten, die sich mit einem Thema aus der Veranstaltung beschäftigt. Die Bearbeitung wird zum Teil in Paaren stattfinden. Sie erfahren am Versuchstag, ob sie alleine oder in einem Paar arbeiten.

Während der Bearbeitung werden Sie aufgezeichnet (Audio, Video, Desktop, Programmierereignisse). **Beachten Sie**: Die Teilnahme an den Aufzeichnungen ist freiwillig (allerdings müssen Sie, falls Sie einen Schein erwerben wollen zumindest an der Bearbeitung der Aufgabe teilnehmen). Für unsere Forschung wäre es aber sehr hilfreich, wenn Sie sich zur Aufzeichnung bereit erklären.

Im Weiteren wird nun beschrieben, wie Ihr Rechner konfiguriert sein soll. Die Installationen orientieren sich an Windows XP. Bei Windows 2000 kann es Abweichungen im Detail geben (beim Mac können Sie weder mediaCam noch ECG installieren).

Gehen Sie Schritt für Schritt vor. **Haken Sie die Punkte ab**, **wenn sie erledigt (und verifiziert) sind. Viele der Punkte sollten sowieso schon auf Ihrem Rechner erfüllt sein. Sie sollten in diesem Fall *keine* Neuinstallationen vornehmen**. Ziehen Sie im Zweifelsfall die referenzierten Übungsblätter zu Rate. Die zu installierende Software für die Beobachtung ihrer Tätigkeiten (siehe weiter unten) finden Sie auf der Seite http://projects.mi.fu-berlin.de/w/bin/view/SE/VorlesungBISJ2EE2005 in der Datei experiment.zip (unter dem Punkt Übung 14A[1]) oder der übergebenen CD.

Falls Sie mit einem Punkt Schwierigkeiten haben, setzen Sie Sich unbedingt bis zum 03.02.2006 15:00 mit salinger[*AT***]inf.fu-berlin.de in Verbindung oder kommen Sie in Raum 007 vorbei.**

Sie können die Installationen auch mit dem Übungsleiter zusammen durchführen. Kommen Sie hierzu am Mittwoch, den 01.02 zwischen 16 und 19Uhr oder am Freitag, den 03.02 zw. 16 und 19Uhr in Raum 007.

☐	1. Sie sollen ein **JDK** (z.B. 1.4.2) installiert haben (so wie auf Übungsblatt 2 in Aufgabe 1 beschrieben, bzw. so wie Sie es für die zurückliegenden JBoss-Aufgaben verwendet haben)
☐	2. Falls Sie unter Punkt 1 kein **JDK 1.5** installiert haben, so tun sie dies nun bitte *zusätzlich*[2](Sie finden das JDK unter http://projects.mi.fu-berlin.de/w/bin/view/SE/VorlesungBISJ2EE2005Softwarepakete). Sie müssen (und sollten) dieses JDK nicht als Standard-JDK auf Ihrem Rechner einrichten. Das bedeutet, dass die Umgebungsvariable *JAVA_HOME* (so wie auf Übungsblatt 2 in Aufgabe 1 beschrieben) weiterhin auf ein anderes JDK zeigen darf. Wir werden das JDK 1.5 nur zum Start von Eclipse benötigen. Details hierzu finden Sie unter Punkt 3.

[1] Mit dem bekannten Login und Passwort.

☐	3. Sie sollen **Eclipse 3.1** (oder eine andere Version ab Eclipse 3.0 aufwärts) installiert haben. **Wichtig ist allerdings, dass Sie im Experiment Ihr Eclipse mit der Java-Runtime von Java 1.5 starten.** Sollte also JDK 1.5 nicht Ihr Standard-JDK sein, gehen sie bitte wie folgt vor. a. **Stellen Sie sicher, dass Sie unter dem Benutzeraccount angemeldet sind unter dem Sie später entwickeln wollen**. b. Beenden Sie ggf. Eclipse c. Kopieren Sie einen Shortcut zur Datei %Eclipse-Home%/eclipse.exe auf Ihren Desktop. d. Klicken Sie mit der rechten Maustaste auf den eben erstellten Shortcut und wählen Sie „Properties". e. Unter dem Reiter „Shortcut" im Feld Target können Sie den Aufruf der Datei `eclipse.exe` mit Parametern versehen und somit auch die zu verwendende Runtime übergeben. In Abhängigkeit davon, wohin Sie Ihr Eclipse und Ihr JDK 1.5.0 installiert haben sieht der Eintrag unter Target ungefähr wie folgt aus: `D:\j2ee\eclipse3.1\eclipse\eclipse.exe -vm "C:\Program Files\Java\jdk1.5.0_04\jre\bin\javaw" -vmargs -Xmx256M` f. Drücken Sie OK. g. Starten Sie nun Eclipse über diesen Shortcut. Alles sollte wie gewohnt funktionieren. **h. Verwenden Sie im Rahmen des Experiments bitte unbedingt nur diesen Shortcut zum Starten von Eclipse.**
☐	4. Stellen Sie sicher dass Sie in Eclipse mit **Ant** arbeiten können. Installieren Sie falls nötig Ant 1.6 (so wie auf Übungsblatt 2 in Aufgabe 1 beschrieben) und stellen sie es unter Eclipse zur Verfügung (*window->Preferences…->Ant->Runtime-Classpath->Add External JARs…*). Mit der Installation erhalten Sie auch die Ant-Dokumentation.
☐	5. Sie sollen eine lauffähige Installation von **JBoss** (JBoss 3.2.2 RC2) besitzen (so wie auf Übungsblatt 2 in Aufgabe 1 beschrieben, bzw. so wie bzgl. Übungsblatt 11 in Aufgabe 11 verwendet). Testen Sie Ihre Installation, indem sie nach dem Start von JBoss http://localhost:8080/web-console aufrufen.
☐	6. Sie sollten eine Installation von **XDoclet** 1.2 besitzen damit Sie **Zugriff auf die XDoclet-Dokumentation** haben (so wie auf Übungsblatt 2 in Aufgabe 1 beschrieben). Rufen Sie in Ihrem Browser $XDocletHome\docs\index.html auf und setzen Sie sich diese Seite als Bookmark.
☐	7. Innerhalb von Eclipse sollen Sie ein Projekt besitzen in dem sowohl der **XPetstore-Dokumentenbaum wie auch ihre lauffähige Lösung von Aufgabe 11** (Übungsblatt 11) enthalten ist. Ferner soll die (höchstwahrscheinlich von Ihnen im Rahmen von Aufgabe 11 angepasste) „XPetstore-ejb"-Build-Datei (`build.xml`) ausführbar sein. Überprüfen Sie dies, in dem Sie folgende Schritte ausführen: a. Starten Sie JBoss.

[2] Es reicht hier auch die Java 1.5 Runtime (siehe hierzu Punkt 3)

	b. Rufen Sie in ihrem Browser http://localhost:8080/xpetstore-ejb/index.jsp. Wenn der *Pet Store* (und keine Fehlermeldung) erscheint, sollten sie den *Pet Store* erst einmal „Undeployen". Rufen Sie hierzu innerhalb von Eclipse das „*undeploy*"-Target der „XPetstore-ejb"-Build-Datei auf. Überprüfen sie durch einen erneuten Aufruf von http://localhost:8080/xpetstore-ejb/index.jsp, ob das „Undeploy" funktioniert hat. c. Rufen Sie innerhalb von Eclipse das „*all*"-Target der Build-Datei auf (es sollten *clean*, *xdoclet*, *compile*, *jar*, *war* und *ear* erfolgreich durchlaufen werden). d. Rufen Sie das „deploy"-Target der Build-Datei auf. Beobachten Sie dabei die Ausgaben im Konsolenfenster des JBoss. Rufen Sie nun http://localhost:8080/xpetstore-ejb/index.jsp. Es sollte die *Pet Store* Startseite erscheinen. Klicken Sie auf den Link „fish". Es sollte eine Seite mit einer Fehlermeldung (*http Status 500*) erscheinen. Dies ist kein Wunder, wir haben ja noch nicht die Datenbank befüllt (dies muss für XPetstore nach jedem „Deploy" oder nach jedem Neustart des JBoss erfolgen). e. Befüllen Sie die Datenbank, indem Sie in Eclipse das „db"-Target der Builddatei aufrufen. Danach sollte der Link „fish" funktionieren. f. Überprüfen Sie nun noch, ob sich Ihre Implementierung von Aufgabe 11 über das von Ihnen eingerichtete Target „*startEmailSpy*" in Eclipse aufrufen lässt (siehe Übungsblatt 11). Als Ergebnis sollte eine Email versendet werden (in der JBoss-Konsole sollte so etwas wie „`…MailerMDB.send(…`" ausgegeben werden). Wenn Sie keine Mail bekommen, überprüfen Sie Ihre Einträge in `$JBossHome\server\default\deploy\mail-service.xml` (siehe auch Übungsblatt 2 in Aufgabe 1 im Abschnitt XPetstore)
☐	8. Sie sollen die **JDK-API-Dokumentation** lokal auf ihrem Rechner installiert haben (siehe Übungsblatt 2 Aufgabe 1). Sie finden die Dokumentation auf http://projects.mi.fu-berlin.de/w/bin/view/SE/VorlesungBISJ2EE2005Dokumentationspakete
☐	9. Sie sollen die **J2EE 1.4 API-Dokumentation** lokal auf ihrem Rechner installiert haben. Sie finden die Dokumentation auf http://projects.mi.fu-berlin.de/w/bin/view/SE/VorlesungBISJ2EE2005Dokumentationspakete
☐	10. Kopieren Sie den **Vorlesungsfoliensatz zu JMS** (`45_JMS.pdf`) auf Ihren Rechner. Sie finden ihn unter http://projects.mi.fu-berlin.de/w/bin/view/SE/VorlesungBISJ2EE2005
☐	11. Kopieren Sie **Übungsblatt 2** (`22U_j2ee_infrastruktur.pdf`) auf Ihren Rechner. Sie finden es auf der Vorlesungswebseite.
☐	12. Holen Sie sich die Datei **`experiment.zip`** von http://projects.mi.fu-berlin.de/w/bin/view/SE/VorlesungBISJ2EE2005 (unter dem bekannten Login/Passwort). Sie enthält die Software, die zur Beobachtung Ihrer Tätigkeiten eingesetzt wird. **Entpacken Sie die Datei in ein temporäres Verzeichnis.** Alle folgenden Angaben zu der zu installierenden Software beziehen sich hierauf.
☐	13. **Installation des Treibers für die WebCam**: Im Verzeichnis „**`QuickCam_Pro_4000`**" finden Sie die Datei `qc848deu.exe`, im Verzeichnis „**`QuickCam_Pro_5000`**" die Datei `qc950deu.exe`. Dies ist die Treibersoftware für die WebCam, die wir an Ihren Rechner per USB anschließen werden. Führen Sie die Installation (**als Administrator**) desjenigen Kameratyps aus, die Ihnen zugewiesen wurde. (wählen Sie dabei die vollständige Installation!).

	Nach der Installation müssen Sie Ihren Rechner neu starten. Nach dem Experiment können Sie diese Software leicht wieder deinstallieren.
☐	14. **Installation von mediaCam** (Vorbemerkung: Nach dem Start der installierten Software und Eingabe von der „Serial Number" und dem „Unlock Key" werden Sie später aufgefordert werden sich online zu registrieren. **Tun Sie dies nicht, sondern wählen Sie „Don't Register"**). Für die Installation gehen Sie wie folgt vor: a. Im Verzeichnis „`mediaCam`" finden Sie die Dateien *mediaCamAVInstaller.exe* und *mediaCamKey.txt*. Bei *mediaCamAVInstaller.exe* handelt es sich um den Installer für die Software, die die Audio-,Video- und Desktopaufzeichnungen durchführen wird (von der Firma netu2). Diese Software ist kostenpflichtig und *muss* von Ihnen nach dem Experiment von Ihrem Rechner gelöscht werden. b. Rufen Sie den Installer auf. **Hierfür benötigen Sie Administrationsrechte auf Ihrem Rechner**. c. Ändern Sie auf keinen Fall die erscheinenden Defaultwerte des Installers (außer bei der Zustimmung zu den Lizenzbestimmungen) d. **Legen Sie sich nach der Installation ein Shortcut auf mediaCamAV.exe (z.B. zu finden unter `C:\Program Files\netu2`) auf Ihren Desktop**. e. Starten Sie das Programm über den Shortcut. f. Geben Sie in einem ersten Schritt „Serial Number" und „Unlock Key" ein (finden Sie in *mediaCamKey.txt*). In einem zweiten Schritt werden Sie aufgefordert sich online zu registrieren. **Diese Online-Registrierung sollten Sie nicht durchführen – wählen Sie „Don't Register"!**
☐	15. **Konfiguration mediaCam**: a. **Stellen Sie sicher, dass Sie unter dem Benutzeraccount angemeldet sind unter dem Sie später entwickeln wollen**. b. Ein Teil der nötigen Konfiguration (z.B. Auswahl des Audio-Devices bzw. Video-Device) kann erst vorgenommen werden wenn die WebCam angeschlossen ist. Dies tun wir gemeinsam vor dem Experiment. Aus diesem Grund ist es wichtig, dass Sie zum angegebenen Termin pünktlich erscheinen. c. Zur Durchführung aller anderen notwendigen Einstellungen finden Sie im Verzeichnis „`mediaCam_Setup`" eine Batchdatei mit dem Namen `mcamSetup.bat`. Führen Sie diese Datei in einer Shell aus (beenden Sie vorher mediaCam): i. Evtl. werden Sie etwas gefragt wie „Are you sure you want to add the information in mcam.reg to the registry?". Klicken Sie auf "Yes". ii. Ferner wird ein Fenster mit dem Titel „Account Setup" geöffnet (evtl. nur minimiert in der Taskbar!) welches Sie auffordert: „*Enter your Name/Group Number*". Geben Sie hier die Ihnen zugeteilte Teilnehmernummer, einen Unterstrich und Ihren Familiennamen ein (z.B. *15B_Salinger*). Drücken Sie OK. iii. Zuletzt sollte nun mediaCam automatisch gestartet werden. Deselektieren Sie ggf. beim erscheinenden „Tip of the Day" die Box „Show Tipps on startup" und drücken Sie OK.

	iv. mediacam ist jetzt so eingestellt, dass es sich minimiert, wenn die Aufnahme gestartet wird. Die Aufnahme kann dann über ein Icon (rotes Quatrat) in der Taskleiste gestoppt werden. **Starten Sie aber jetzt keine Aufnahme** (sehen sie hierzu Punkt d.) sondern beenden Sie mediaCam. d. **Ändern Sie nach der Ausführung der Batchdatei nichts mehr an den Einstellungen von mediaCam**! Beachten Sie: Sie können im Augenblick keine Aufzeichnungen starten, da die eben eingespielte Konfiguration verlangt, dass Audio- und Videodevices ausgewählt sind. Dies geht aber erst nachdem die WebCams angeschlossen sind. **Starten Sie also jetzt keine Aufnahme mehr**.
☐	16. **Installation des Werkzeuges zur Aufzeichnung der Eclipse-Events**: In diesem Schritt wird das so genannte Electrocodeogram (ECG) als Plugin von Eclipse installiert. Das ECG wird Ihre Aktivitäten in Eclipse (z.B. Öffnen einer bestimmten Datei) protokollieren. Die Dateien für die Installation finden Sie im Verzeichnis „`ecg_Setup-Inline-Server`". Gehen Sie wie folgt vor: **a. Stellen Sie sicher, dass Sie unter dem Benutzeraccount angemeldet sind unter dem Sie später entwickeln wollen.** b. Beenden Sie Eclipse. c. Kopieren Sie die Datei `org.electrocodeogram.sensor.eclipse.ecg.sensorshell_1.0.0.zip` in das Pluginverzeichnis Ihrer Eclipse-Installation (`%Eclipse_Home%/plugins`)und entpacken Sie sie dort. d. Führen Sie dann in einer Shell die Batchdatei `ecgSetup.bat` aus (hierdurch werden die Ordner `ecg_log` und `.hackystat` in Ihr Benutzerprofil kopiert). e. Zur Überprüfung, ob die Installation des Plugins funktioniert hat, starten Sie nun Eclipse (falls Sie nicht standardmäßig unter JDK 1.5 arbeiten, müssen Sie Eclipse über den unter Punkt 3 eingerichteten Shortcut starten). f. Sehen Sie *danach* in das Verzeichnis `ecg_log` in Ihrem Benutzerprofil (`…\Documents and Settings\<username>`). Wenn Sie in diesem Verzeichnis eine oder mehrere Dateien finden ist alles gut gegangen. g. Beenden Sie Eclipse
☐	17. **Einstellen der Windows Taskleiste**: Damit bei den Desktopaufzeichung immer die aktuelle Zeit sichtbar ist, stellen Sie die Windows-Taskleiste bitte so ein, dass sie *immer* im Vordergrund dargestellt wird und die Uhr eingeblendet ist: a. **Stellen Sie sicher, dass Sie unter dem Benutzeraccount angemeldet sind unter dem Sie später entwickeln wollen**. b. Klicken Sie mit der rechten Maustaste auf die Windows-Taskleiste und wählen Sie im Popup „Properties". c. Unter dem Reiter "Taskbar" selektieren Sie „Lock the Taskbar", „Keep the Taskbar on top of other windows" und "Show Clock". d. Deselektieren Sie "Auto-hide the taskbar" und „Hide inaktive icons". e. Drücken Sie OK.

☐	**18. Stellen Sie die Systemuhr auf die korrekte aktuelle Uhrzeit.** Als Administrator klicken Sie dazu auf die Uhranzeige in der Taskleiste.
☐	**19. Festplattenplatz bereitstellen**. Damit die Audio-, Video- und Desktopaufzeichnungen auf ihrem Rechner funktionieren, **müssen Sie auf einer Ihrer Partitionen ca. 1,5 GB frei haben**. Am günstigsten wäre dies auf der Systempartition (diejenige, auf der auch die Benutzerprofile liegen; im allgemeinen `C:`). Sorgen Sie für den freien Platz! Sollte es nicht möglich sein, so viel Platz auf der Systempartition zur Verfügung zu stellen, so tun Sie dies bitte auf einer anderen Partition. In diesem Fall müssen wir dann aber noch mediaCam beibringen nicht auf die Systempartition zu schreiben (siehe Punkt 20). **Stellen Sie aber in jedem Fall sicher, dass es auf der Systempartition mindestens 100MB freien Platz gibt**.
☐	**20. Nochmalige Umkonfiguration mediaCam:** **Achtung:** Dieser Punkt muss nur von denjenigen ausgeführt werden, die die unter Punkt 19 geforderten 1,5 GB nicht auf der Systempartition sondern nur auf einer anderen Partition zur Verfügung stellen konnten. Um mediaCam beizubringen, in eine von Ihnen bestimmte Partition zu schreiben, gehen Sie wie folgt vor: a) **Stellen Sie sicher, dass Sie unter dem Benutzeraccount angemeldet sind unter dem Sie später entwickeln wollen**. b) Legen Sie in der Wurzel der Partition mit den freien 1,5GB (hier im weiteren [X:] genannt) ein Verzeichnis mit Namen `mediaCam_captures` an. c) Starten Sie mediaCam (über Ihren Shortcut auf ihrem Desktop) d) Klicken sie im linken Menü auf „File Management" e) Klicken rechts auf den Reiter „My Places" f) Klicken Sie auf das Symbol „Create Shortcut" (linkestes Symbol: ) g) Tragen Sie im neu geöffneten Fenster unter „Name" den String „`mediaCam_captures`" ein und wählen Sie unter Target über den angebotenen Dateibrowser das eben angelegte Verzeichnis „`mediaCam_captures`" auf Ihrer Partition [X:] aus (z.B. „`X:\mediaCam_captures`"). Tragen Sie unter „File Prefix" einen String der Form *<ihre Teilnehmernummer>_<ihr_Familienname>* (z.B. `15A_Salinger`) ein. Drücken Sie OK. h) Selektieren Sie nun in der Liste unterhalb des Reiters „My Places" den neuen Eintrag „mediaCam_captures" und Klicken dann auf das Symbol für die Wahl des Defaultordners (rechtestes Symbol:) i) Der Listeneintrag „mediaCam-captures" sollte nun mit einem kleinen schwarzen Haken versehen sein. j) Beenden Sie mediaCam.
☐	21. **Überprüfen Sie nun noch einmal, ob Sie alle Punkte erledigt haben**. Läuft Ihr Eclipse unter JDK1.5? Läuft Ihr JBoss? Läuft ihr XPetstore? Funktioniert der Deploy-Vorgang via Ant von XPetstore aus Eclipse heraus? Haben Sie genügend Festplattenplatz zur Verfügung gestellt? Etc.
☐	22. **Kommen Sie zw. dem 01.02 und dem 03.02.2005 mit Ihrem Laptop**

	im Büro 007 vorbei. Am günstigsten am 01.02 zw. 16 und 19Uhr oder am 03.02 zw. 16 und 19Uhr. Wir überprüfen dann zusammen kurz (Dauer: 10 Minuten), ob Ihre Installation OK ist und testen wir auch gleich den Anschluss der Kamera.

Aufgabe 14A-2: (Bereitstellung von Daten von Daten außerhalb Ihres Entwicklungsumfeldes)

Vorbemerkung: Diese Aufgabe gilt für alle Teilnehmer (mit und ohne Laptop). Evtl. haben Sie die gestellten Anforderungen schon im Rahmen der Übung 13.2 und der zugehörigen Mail auf der Mailingliste erfüllt.

Ziel dieser Aufgabe ist es, dem Veranstalter einen Zip-Export von Ihrem Eclipse-Projekt mit XPetStore und Ihrem `XpetstoreEmailSpy` (Übung 11) sowie die Konfigurationsdateien Ihres JBoss zur Verfügung zu stellen. Gehen Sie hierbei wie folgt vor:

1) Erstellen und testen Sie einen Zip-Export wie in Aufgabe 13.2 auf Übungsblatt 13 oder auf der Mail in der Mailingliste beschrieben. Die Exportdatei soll folgenden Namen haben: `<Familienname>_UE11eclipse.zip`
2) Packen Sie das Verzeichnis %JBoss_Home%\server\default\deploy (bzw. das Verzeichnis, welches die von Ihnen benutze JBoss-Konfiguration enthält) in ein Zip mit dem Namen `<Familienname>_UE14Ajboss.zip`. Damit das Verzeichnis nicht unnötig groß wird, führen Sie bitte vorher (bei laufendem JBoss) ein „Undeploy" (siehe entsprechendes Ant-Target) ihres XpetStore durch. Überprüfen Sie danach ruhig einmal, ob das „Undeploy" funktioniert hat. D.h., rufen Sie den XPetStore auf (http://localhost:8080/xpetstore-ejb/index.jsp) und erhalten Sie eine Fehlermeldung (HTTP Status 500).
3) Bringen sie beide Zip-Dateien (wenn Sie es noch nicht gemacht haben) bis zum 01.02.2006 auf einem USB-Stick oder einer CD in Raum 007 vorbei.

Aufgabe 14A-3: (Erklärungen ausfüllen)

Füllen Sie die beiden ausgeteilten Erklärungen (Einverständniserklärung, Erklärung zur Softwareverwendung) aus, unterschreiben Sie diese und bringen Sie diese in der KW5 in Raum 007 vorbei oder bringen Sie diese zum Experiment mit.

Aufgabe 14A-4: (Fragebogen ausfüllen)
Füllen Sie den ausgeteilten Fragebogen zu Ihrer Programmiererfahrung aus und bringen Sie diesen in der KW5 in Raum 007 vorbei oder bringen Sie diesen zum Experiment mit.

Teilnehmer und Termine:

- **Kommen Sie unbedingt pünktlich zu Ihrem Termin.**
  - **Montag, den 06.02.2006: 09:30 bis 14Uhr: Treffpunkt R005**
  - **Dienstag, den 07.02.2006: 11:30 bis 16Uhr: Treffpunkt R007**
  - **Donnerstag, den 09.02.2006: 09:30 bis 14Uhr: Treffpunkt R007**
  - **Freitag, den 10.02.2006: 15:00 bis 19:30Uhr: Treffpunkt R007**
- **Vergessen Sie Ihr konfiguriertes Laptop nicht.**
- **Bringen Sie den ausgefüllten Fragebogen mit.**
- **Bringen Sie sich ggf. eine Maus mit.**

Vorlesung "Bau betrieblicher Informationssysteme mit J2EE"
Freie Universität Berlin, Institut für Informatik, Arbeitsgruppe Software Engineering
Prof. Dr. Lutz Prechelt, Stephan Salinger
Übungsblatt 14A **Ergänzung** WS 2005/2006 2006-02-01

Aufgabe 14A-Ergänzung: (Ergänzungen und Änderungen zu den Angaben in Übungsblatt 14A)

Vorbemerkung: Leider führt das in Übungsblatt 14A angesteuerte Zusammenspiel der Software *mediaCam* und der *Logitech Quickcam Pro 5000* zu unerwarteten Problemen. Um diese Probleme zu lösen, müssen die Anweisungen von Übungsblatt 14A ergänzt bzw. geändert werden. Diese Ergänzungen und Änderungen sind hier im Weiteren beschrieben.

Gehen Sie Schritt für Schritt vor. **Haken Sie die Punkte ab**, **wenn sie erledigt (und verifiziert) sind.**

Die neu hinzugekommene zu installierende Software *Camtasia Studio 3* für die Beobachtung ihrer Tätigkeiten (siehe weiter unten) finden Sie auf der Seite http://projects.mi.fu-berlin.de/w/bin/view/SE/VorlesungBISJ2EE2005 in der Datei camtasiaf.exe (unter dem Punkt Übung 14A[1]). *Camtasia* ersetzt die eigentlich für diesen Zweck vorgesehen Software *mediaCam*. Sollten Sie *mediaCam* schon installiert und konfiguriert haben, so belassen Sie diese einfach auf Ihrem Rechner (bis nach dem Experiment – dann löschen!). Wir werden sie aber nicht verwenden.

Falls Sie mit einem Punkt Schwierigkeiten haben, setzen Sie Sich unbedingt bis zum 03.02.2006 15:00 mit salinger[*AT***]inf.fu-berlin.de in Verbindung oder kommen Sie in Raum 007 vorbei.**

Sie können die Installationen auch mit dem Übungsleiter zusammen durchführen. Kommen Sie hierzu am Donnerstag, den 01.02 zwischen 18 und 19Uhr oder am Freitag, den 03.02 zw. 16 und 19Uhr in Raum 007.

Kommen Sie *auf jeden Fall* bis zum 03.02.2006 im Raum R007 vorbei. Wir führen dann die fehlenden Konfigurationsschritte (Punkt 6 und Punkt 7) mit angeschlossener Kamera durch und verifizieren kurz den Rest Ihrer Installation. Nutzten Sie hierzu möglichst die oben angegebenen Termine am Donnerstag oder am Freitag. Sie können aber auch zu einem anderen Zeitpunkt erscheinen.

☐	1. Sie müssen die Punkte 14 (Installation von *mediaCam*), 15 (Konfiguration von *mediaCam*) und 20 (Nochmalige Umkonfiguration von *mediaCam*) von Übungsblatt 14A **nicht durchführen**. Haben Sie diese Schritte aber schon gemacht, so ist es auch nicht schlimm. Wir werden mediaCam einfach nicht benutzen.
☐	2. **Installation des Treibers für die WebCam**: Installieren Sie jetzt den Treiber für die *Logitech Quickcam Pro 5000.* Dies ist der bisher ausgelassene Schritt 13 aus Übungsblatt 14A. **Achtung**: Alle Teilnehmer installieren sich den Treiber für die *Logitech Quickcam Pro 5000* (außer Teilnehmer 2B, dieser nimmt *Logitech Quickcam Pro 4000).* Der Treiber liegt im Verzeichnis `QuickCam_Pro_5000` der Zip-Datei `experiment.zip`.
☐	**3. Ausschalten des Schnellassistent der WebCam:** Diesen Schritt dürfen Sie nicht vergessen, da sonst der Kameraschnellassistent später nach dem Start einer Aufnahme immer im Vordergrund erscheint:

[1] Mit dem bekannten Login und Passwort.

	a. Klicken Sie mit der rechten Maustaste auf das Kamerasymbol in der Taskbar und wählen Sie „Schnellassistent"->"Schnellassistent aktivieren" (eine etwas seltsame Bezeichnung für das Ausschalten ;-)). b. Nach dem Ausführen sollte der Haken neben dem „Schnellassistent"->"Schnellassistent aktivieren" verschwunden sein. Der Schnellassistent ist also deaktiviert.
☐	4. **Installation von Camtasia Studio 3**: In diesem Schritt wird die neue Aufzeichnungssoftware (Audio, Video, Desktop) installiert. a. Laden Sie sich hierzu von der Webseite der Veranstaltung (http://projects.mi.fu-berlin.de/w/bin/view/SE/VorlesungBISJ2EE2005) die Datei `camtasiaf.exe` herunter. Es handelt sich um den Installer für die Software. b. Führen Sie den Installer dann als Administrator aus. Lassen Sie dabei alle Defaulteinstellungen bis auf „Support für Liveausgabe" unverändert. „Support für Liveausgabe" wählen Sie aus. c. Falls nach der Installation Programme (Camtasia) gestartet werden, so beenden Sie diese.
☐	**5. Anlegen ein Shortcuts auf Camtasia Recorder:** Legen Sie einen Shortcut auf die Software „Camtasia Recorder" auf Ihren Desktop (z.B. über Start->All Programms->Camtasia Studio 3-> Applications-> Camtasia Recorder). Verwenden Sie im Weiteren zum Starten von Camtasia nur diesen Shortcut.
☐	6. **Verzeichnis für die Aufzeichnungsdaten von Camtasia auf der *freien* Partition erzeugen**: Erzeugen Sie in Ihrem Dateisystem auf derjenigen Partition auf der Sie die geforderten 1,5 GB Platz haben (siehe Übungsblatt 14A Nummer 19) ein Verzeichnis mit dem Namen „`camtasia_captures`" (also z.B. „`x:\camtasia-captures`").
☐	7. **Einstellen der Bildschirmauflösung**: Die in den weiteren Schritten zu installierende Software „Camtasia" ist bei Ihren Aufnahmen sehr ressourcenintensiv. Damit Ihr Rechner nicht in die Knie geht sollten Sie vor Aufzeichnungen mit Camtasia die **Bildschirmauflösung auf 1024x768 einstellen**. Größere Auflösungen sollten nur dann gewählt werden, wenn Ihnen mindestens 3GB Festplattenspeicher zur Verfügung stehen und Ihr Rechner mit mindestens 2GHz getaktet ist.
☐	8. **Konfiguration von Camtasia Recorder**: **Anmerkung**: Diese Schritte können Sie nur mit angeschlossener Kamera durchführen. **Kommen Sie also hierzu im Laufe der Woche auf jeden Fall – am besten zu den oben angegebene Terminen – in R007 vorbei.** **Die Camtasia Software besteht aus unterschiedlichen Programmen. Wir werden nur den Camtasia Recorder benötigen**. Starten Sie den *Camtasia Recorder* über den unter Punkt 4 angelegten Shortcut. Gehen Sie dann wie folgt vor: a. Auf dem Welcome!-Screen wählen Sie "I would like to evaluate Camtasia Studio" und drücken "Finish" **b. Wählen Sie aus dem Menu „Capture" den Punkt „Wizard..." und führen Sie der Reihe nach aus:**

i. Bei "What would you like to record" wählen Sie "Entire Screen" und dann "Next"

ii. Beim wiederholten "What would you like to record?":

1. "Record Audio" wählen
2. "Record Camera" wählen
3. "Camera Preview" **nicht** wählen
4. "Next"

iii. Bei "Choose Audio Settings":

1. Bei "Audio Device" wählen Sie "Logitech Microphone (Pro 5000)"
2. Wählen Sie „Microphone Audio" (aktivieren)
3. Stellen Sie ein brauchbares „Input Level" ein. Diese wird wohl zw. ½ und 2/3 liegen.
4. "Next"

iv. Bei „Choose Camera Settings":

1. Bei „Available video device" wählen Sie „Logitech QuickCam Pro 5000"
2. Drücken Sie dann den Button „Video Format…". Im neuen Fenster wählen Sie:
   a. „Frame Rate": 5.000
   b. „Output Size": 176x144
   c. Den Rest der Einstellungen gleich lassen!
   d. OK
3. "Next"

v. Bei „Begin Recording":

1. Bei den „Performance Options" wählen Sie die Selectbox „Disable display acceleration during capture" (aktivieren)
2. „Finish"

c. Wählen Sie aus dem Menu „Tools" den Punkt „Options..." und führen Sie der Reihe nach aus:

i. Wählen Sie den Reiter „General"

1. Selektieren (auswählen) Sie „Enable Tips"
2. Deselektieren (ausschalten) Sie:
   a. "Always on top"
   b. "Display recording preview after recording is stopped"
   c. "Display options dialog after recording is saved"
3. Unter "File Options":
   a. Selektieren Sie „Save as CAMREC"

b. Drücken Sie den Button „File Name Options“ und editieren Sie im erscheinenden Dialog:
   i. Wählen Sie „Automatic file Name“
   ii. Tragen Sie im Feld Prefix einen String der Form *<Ihre_Teilnehmernummer>_<Ihr_Familienname>* ein (also z.B. *15A_Salinger*)
   iii. Wählen Sie unter „Output folder“ das Verzeichnis „`camtasia_captures`“ (dies ist das, welches Sie unter Punkt 6 angelegt haben (also z.B. „`x:\camtasia-captures`“).
   iv. OK

☆ 4. Unter “Temporary file folder”:
   a. Wählen Sie unter „Output folder“ das Verzeichnis „`camtasia_captures`“ (dies ist das, welches Sie unter Punkt 6 angelegt haben (also z.B. „`x:\camtasia-captures`“).

ii. **Wählen Sie den Reiter „Streams“**
   1. Unter „Screen video option“
      a. Deselektieren (ausschalten) Sie „Auto config“
      b. Unter „Capture frame rate“ tragen sie „4“ ein
      c. Deselektieren (ausschalten) Sie „Time-laps capture“
   2. Unter „Audio options“
      a. Selektieren Sie „Interleave audio every“
      b. Wählen Sie „1“ „Second“
      c. Unter „Audio Capture Device“ wählen Sie „Logitech Microphone(Pro 5000)" (sollte schon vorgewählt sein)

iii. Wählen Sie den Reiter „Camera“
   1. Unter „Available video device“ wählen Sie "Logitech Quickcam Pro 5000" (sollte schon vorgewählt sein)

iv. Wählen Sie „OK“

d. **Wählen Sie aus dem Menu „Effects“ den Punkt „Options...“ und führen Sie der Reihe nach aus**:
   i. In „System stamp“:
      1. Selektieren Sie:
         a. „Time/Date“
         b. „Elapsed time“
         c. „Show Stamp for 30000 seconds“
      2. Deselektieren Sie
         a. „Computer name“
         b. „User name“

	3. Drücken Sie den Button „Options…" und wählen Sie im neuen Fenster: a. In Style: i. Selektieren Sie 1. „Normal" 2. „Transparent background" 3. „Word wrap" b. In Position: i. Klicken Sie auf das Quadrat in der obersten Reihe in der Mitte ☆ c. Wählen Sie als „Text Color…" ein leuchtendes rot. d. „OK" 4. Wählen Sie „OK" ☆ e. **Wählen Sie aus dem Menu „Effects" den Punkt „Annotations" und wählen Sie dort den Punkt „Add System Stamp"** (danach sollte „Add System Stamp" mit einem Haken versehen sein!) f. Beenden Sie „Camtasia Recorder"
☐	9. **Testen von Camtasia Recorder**: **Anmerkung**: Diese Schritte können Sie nur mit angeschlossener Kamera durchführen und wenn Sie den vorhergegangen Punkt vollständig durchgeführt haben. **Kommen Sie also hierzu im Laufe der Woche auf jeden Fall – am besten zu den oben angegebene Terminen – in R007 vorbei.** a. Überprüfen Sie Ihre Bildschirmauflösung (siehe oben)! b. Starten Sie Camtasia Recorder über den Shortcut auf Ihrem Desktop c. Drücken Sie Record (der „rote Ball"). Die Aufnahme startet dann, wenn Sie nicht mit zwei Monitoren arbeiten. Bei zwei Moni d. Die Aufnahme startet dann, wenn Sie nicht mit zwei Monitoren arbeiten. Bei zwei Monitoren startet die Aufzeichnung, sobald Sie mit der Maus noch einmal irgendwohin auf dem Desktop klicken. i. Evtl. gibt es noch eine kurze Verzögerung, weil die Kamera initialisiert werden muss. Dies wird Ihnen durch ein Popup angezeigt. e. Wenn die Aufnahme läuft, wechselt der „Camtasia"-Ball in der Menuleiste zyklisch seine Farbe von rot nach grün. Über diesen Ball können Sie die Aufnahme durch einen Rechtsklick und die Auswahl von Stop abbrechen. Tun Sie dies nach ca. einer Minute. f. Nach dem Stoppen sollte sich der (während der Aufnahme automatisch minimierte) Camtasia-Recorder wieder zeigen. **Achtung**: Camtasia ist zu diesem Zeitpunkt noch nicht fertig. Jetzt wird erst im Hintergrund das eigentliche Video erzeugt. Warten Sie so lange, bis der Bildschirm einmal kurz schwarz aufflackert und das Hauptfenster vom Camtasia Recorder wieder das Logo „TechSmith CAMTASIA Recorder" anzeigt. Erst dann ist die Aufzeichnung fertig

	g. Beenden Sie nun Camtasia-Recorder h. Sehen sie nun in dem von Ihnen angelegten Verzeichnis „`camtasia_captures`" nach, ob sich dort eine Datei mit dem Namen „`<Ihre_Teilnehmernummer>_<Ihr_Familienname>001.camrec`" befindet. Sie können das Video mit einem Doppelklick starten (es wird dann die Applikation „Camtasia Studio" gestartet).
☐	☆ **10. Umkonfiguration ECG:** In diesem Punkt wird davon ausgegangen, dass Sie ECG wie im Übungsblatt 14A unter Punkt 16 installiert haben. Leider gibt es Probleme mit dieser Version vom ECG. Deshalb gehen Sie bitte wie folgt vor: a. Besorgen Sie sich von der Vorlesungswebseite unter Punkt Übung 14A die Datei `org.electrocodeogram.sensor.eclipse_0.9.0.zip`. b. Beenden Sie Eclipse. Seien Sie unter dem Account angemeldet, unter dem Sie entwickeln. c. Löschen Sie im `plugins`-Verzeichnis Ihrer Eclipse-Installation den Ordner `org.electrocodeogram.sensor.eclipse.ecg.sensorshell_1.0.0` (das ist der, den Sie auf Übungsblatt 14A unter Punkt 16 angelegt haben) d. Entpacken Sie nun `org.electrocodeogram.sensor.eclipse_0.9.0.zip` in das `plugins`-Verzeichnis Ihrer Eclipse-Installation. Danach haben Sie im `plugins`-Verzeichnis ein Verzeichnis `org.electrocodeogram.sensor.eclipse_0.9.0`. e. Wechseln Sie in das Verzeichnis `org.electrocodeogram.sensor.eclipse_0.9.0`. im `plugins`-Verzeichnis Ihrer Eclipse-Installation und gehen Sie dort in das Verzeichnis `ecg`. f. Dort finden Sie eine Datei `ecg.bat`, die wir noch editieren müssen. Öffnen Sie also `ecg.bat` in einem Editor. g. Uns interessiert nur der Teil ganz am Anfang der ersten (und einzigen) Zeile: `"c:\Program Files\Java\jre1.5.0_06\bin\java.exe"`. Es handelt sich um den Java-Aufruf von Java 1.5. Ändern Sie den Pfad nun so, dass er auf das `java.exe` Ihrer Java-1.5-Installation zeigt (diese sollten Sie ja nach Übungsblatt 14A Punkt 2 besitzen). h. Speichern Sie die Datei ab und beenden Sie Ihren Editor. i. Nun können Sie Eclipse (über Ihren Shortcut auf dem Desktop (siehe Übungsblatt 14A Punkt 3)) starten. j. Wenn alles gut gegangen ist, erscheint nun neben dem Eclipsefenster ein kleines Fenster mit dem Titel „ElectroCodeoGram". **Schließen Sie dieses Fenster während des Experimentes nicht!**
☐	**11. Kommen Sie bis zum 03.02.2006 in Raum 007 vorbei um Ihre Konfiguration zu vervollständigen.**

Teilnehmer und Termine:

- **Kommen Sie <u>unbedingt pünktlich</u> zu Ihrem Termin.**
  - **Montag, den 06.02.2005: 09:30 bis 14Uhr: Treffpunkt R005**
  - **Dienstag, den 07.02.2005: 11:30 bis 16Uhr: Treffpunkt R007**
  - **Donnerstag, den 09.02.2005: 09:30 bis 14Uhr: Treffpunkt R007**
  - **Freitag, den 10.02.2005: 15:00 bis 19:30Uhr: Treffpunkt R007**
- **Vergessen Sie Ihr konfiguriertes Laptop nicht.**
- **Bringen Sie den ausgefüllten Fragebogen mit.**
- **Bringen Sie sich ggf. eine Maus mit.**

Vorlesung "Bau betrieblicher Informationssysteme mit J2EE"
Experiment zur Paarprogrammierung
Freie Universität Berlin, Institut für Informatik, Arbeitsgruppe Software Engineering
Prof. Dr. Lutz Prechelt, Stephan Salinger
Übungsblatt 14B WS 2005/2006 Februar 2006

Diese Anleitung wird nach dem Experiment wieder eingesammelt, da Sie in ihr wichtige Zeiten notieren werden. Reißen Sie die Anleitung also bitte nicht auseinander! Sie dürfen sich wichtige Stellen gern anstreichen.
Blättern Sie bitte immer erst zur nächsten Seite, wenn der Übungsleiter Sie dazu auffordert!
Bevor Sie auf dieser Seite weiter lesen, tragen Sie zuerst folgende Daten ein:

Ihre Teilnehmernummer: _____

Ihr Familienname: _____

Ihr Vorname: _____
Sie arbeiten in diesem Experiment im Paar? Ja ☐ Nein ☐
Ggf. die Teilnehmernummer ihres Partners im Experiment: _____

Herzlich Willkommen zu unserem Experiment im Rahmen der Veranstaltung „Bau betrieblicher Informationssysteme mit J2EE."
Sie werden in diesem Experiment eine Programmieraufgabe lösen, die sich mit JMS (Java Messaging Service) beschäftigt. Beachten Sie, dass es sich um ein wissenschaftliches Experiment handelt. Es geht nicht darum Ihre persönliche Leistung zu messen oder in eine Bewertung einzubringen.

Ein Teil von Ihnen wird diese Aufgabe alleine bearbeiten, ein anderer Teil als Paar (allerdings steht auch jedem Paar nur ein Rechner zur Verfügung). Die Soloprogrammierer können Ihrem gewohnten Arbeitsstil nachgehen. Die Paarprogrammierer sollten versuchen den Vorteil zu nutzen, der Ihnen daraus erwächst, dass Sie zu zweit sind. Sie sollten sich (müssen es aber nicht) an der Tastatur und Maus abwechseln. Gehen Sie so vor, wie es Ihnen am günstigsten erscheint.

Verwenden Sie alle Dokumentation die Sie zur Verfügung haben (auch Internet), aber schauen und hören Sie *nicht* zu, was Ihre Nachbarn bzw. Nachbargruppen machen.

Fast alle Teilnehmer werden auf ihrem eigenen Rechner (bzw. dem ihres Partners) arbeiten. Für Teilnehmer ohne Windows-Laptop stehen speziell vorbereitete Rechner zur Verfügung.

Für alle Teilnehmer gilt: Sie werden aufgezeichnet (Bild, Ton, Bildschirm). Sie können diese Aufzeichnung jederzeit ohne Angabe von Gründen abbrechen. Es wäre aber schön, wenn Sie bis zum Ende dabei bleiben, denn daraus entsteht der Hauptnutzen für das Experiment.

Für die Lösung der Aufgabe stehen Ihnen max. vier Stunden zur Verfügung. Versuchen Sie in dieser Zeit zu einem lauffähigen Programm zu kommen, das die gegebenen Anforderungen erfüllt (aber mehr auch nicht).

Der Übungsleiter darf Ihnen während des Experimentes nicht bei der Lösung oder bei Problembeseitigungen helfen. Versuchen Sie alleine (bzw. im Paar) klar zu kommen.

Beenden Sie während des Experimentes auf keinen Fall die Aufzeichnungssoftware Camtasia.
Bitte erst nach Aufforderung umblättern!

Der weitere Ablauf:

1. Einführung in den Ablauf
2. Rechner vorbereiten
3. Aufnahme mit Camtasia testen
4. Alle notwendigen Applikationen auf ihrem Rechner starten.
5. Mit der Aufgabe starten

1. Einführung in den Ablauf

Das eigentliche Experiment startet später mit dem Notieren der aktuellen Zeit und dem Aufrufen der Aufzeichnungssoftware Camtasia. Dies passiert, nachdem wir uns davon überzeugt haben, dass unsere Aufzeichnung komplett funktioniert und alle notwendigen Applikationen (z.B. JBoss und Eclipse) gestartet sind.

Wenn das Experiment dann läuft, sind Sie auf sich gestellt. Lösen Sie die Ihnen gestellte Aufgabe. Testen Sie Ihre Lösung. Wenn Sie meinen, dass Ihr Programm funktioniert, notieren Sie die aktuelle Zeit und heben den Arm, um dem Übungsleiter Bescheid zu geben. Dieser wird schnellstmöglich kommen um Ihrer Lösung zu überprüfen. Gegebenenfalls bekommen Sie Gelegenheit zu Nachbesserungen. Genaue Anweisungen hierzu finden Sie später in der Aufgabenstellung.

Am Ende des Experimentes werden wir die aufgezeichneten Daten von Ihrem Rechner auf eine portable Festplatte überspielen. **Gehen Sie nicht, bevor Ihre Daten gesichert sind!**

Wichtig: Bitte sprechen Sie in den nächsten drei Wochen mit niemanden über den Inhalt (z.B. die Aufgabe) des Experimentes. Dies garantiert, dass nachfolgenden Teilnehmer nicht beeinflusst werden. Sollte z.B. die Aufgabe zu anderen Teilnehmern durchsickern, wird das Experiment wertlos.

Achtung:

- Stoppen Sie während des Experimentes *nie* die Aufzeichnung durch Camtasia ohne Rücksprache mit dem Übungsleiter!
- Vermeiden Sie während des Experimentes möglichst auch einen Neustart von Eclipse!
- Beenden Sie nie von Hand das kleine Fenster „ElectroCodeoGram" von ECG, welches bei Start von Eclipse zusätzlich geöffnet wird.
- Erst wenn Sie am Ende des Experimentes vom Übungsleiter *explizit* aufgefordert werden, die *Camtasia*-Aufnahme zu beenden, dann machen Sie dies über einen Rechtsklick auf den kleinen rotgrünen Ball in der Taskbar und die Auswahl von Stop. **Beachten Sie**: *Camtasia* braucht dann noch eine gewisse Zeit, um die Daten zu transformieren und auf Ihre Platte zu schreiben. *Camtasia* ist erst fertig, wenn im wiedererschienenen *Camtasia-Record*-Fenster das *Camtasia*-Logo angezeigt wird. Machen Sie in dieser Zeit möglichst nichts an Ihrem Rechner.

Bitte erst nach Aufforderung umblättern!

2. Rechner vorbereiten

Diese Vorbereitung gilt im Prinzip nur für Teilnehmer, die auf ihrem eigenen Rechner bzw. auf dem Rechner ihres Partners arbeiten.
Personen, die hingegen auf gestellten Rechnern arbeiten haben von diesem Schritt nur Punkt 7 durchzuführen.

1. Schließen Sie die Kamera an Ihren Rechner an.
2. Deaktivieren Sie für das Experiment ggf. eine bei Ihnen *zusätzlich installierte* Firewall so, dass Sie auch nach einem Neustart des Rechners deaktiviert ist.
3. Rebooten Sie Ihren Rechner.
4. Beenden und deaktivieren Sie ggf. Dienste wie Tomcat oder andere Applikationsserver.
5. Stoppen Sie alle jetzt nicht benötigten Programme wie z.B. Messenger, Email-Klienten oder ähnliches. Auf Ihrem Rechner sollten möglichst wenige Prozesse laufen, da wir den Rechner während des Experimentes stark mit der Aufzeichnung belasten.
6. Überprüfen Sie Ihren Zugang zum Internet. Sollte er nicht funktionieren, so ist dies nicht schlimm. Sie sind nicht darauf angewiesen.
7. Tragen Sie Ihre Emaildaten in die JBoss-Konfiguration ein, wenn Sie dies noch nicht gemacht haben (`$JBOSS_HOME\server\default\deploy\mail-service.xml`).
8. Noch einmal: Beenden Sie alle laufenden Programme!

3. Aufnahme mit Camtasia testen

In diesem Schritt wollen wir überprüfen, ob bei Ihrer Camtasia-Installation Ton und Bild richtig eingestellt sind.

1. Achten Sie darauf, dass Ihre Camera angeschlossen ist.
2. Starten Sie die Software *Camtasia Recorder* über den Shortcut auf Ihrem Desktop (bitte nicht mit der Software „Camtasia Studio" verwechseln)
3. Wählen Sie aus dem Menu von *Camtasia Recorder* „Capture->Wizard…".
4. Drücken Sie so lange „Next>" (ohne etwas zu verstellen) bis Sie beim Punkt „Choose Audio Setting" angekommen sind.
5. Reden Sie ein wenig in der Lautstärke, in der Sie auch beim Experiment mit Ihrem Partner sprechen wollen und regeln Sie dabei „Input Level" so, dass sich die Aussteuerung ungefähr im gelben Bereich bewegt. Seien Sie auch kurz still, und überprüfen Sie, ob nicht zuviel Ton von Ihren Nachbarn überspringt.
6. Verstellen Sie sonst nichts.
7. Drücken Sie „Next>".
8. Stellen Sie die Kamera so ein, dass Sie (ggf. beide) gut im Bild zu sehen sind. Bedenken Sie dabei, dass Sie sich später auch ein wenig hin und her bewegen werden.
9. Verstellen Sie sonst nichts.
10. Drücken Sie „Next>".
11. Drücken Sie „Finish".
12. Sie könnten jetzt die Aufnahme starten, indem sie auf den roten Ball im Camtasia-Recorder-Fenster klicken. Das *Camtasia-Recorder*-Fenster würde verschwinden und in der Taskbar würde ein kleiner Ball zyklisch von rot nach grün wechseln. **Wir verzichten aber an dieser Stelle auf einen Start der Camtasia-Aufnahme.**
13. Beenden Sie „Camtasia Recorder"

Bitte erst nach Aufforderung umblättern!

2. Alle notwendigen Applikationen auf ihrem Rechner starten.

In diesem Schritt werden wir schon einmal alle Programme auf Ihrem Rechner starten, die wir benötigen.
Gehen Sie in der angegebenen Reihenfolge vor:

1. Starten Sie JBoss.
2. Starten Sie Eclipse (wenn bei Ihnen ECG problemlos läuft und auch sonst keine Probleme existieren ggf. unter Java 1.5) über Ihren Link auf dem Desktop.
3. Warten Sie bis JBoss und Eclipse fertig gestartet sind.
4. Wenn Sie Eclipse mit ECG verwenden, sollte nun auch das kleine ElectroCodeoGram-Fenster erschienen sein. Minimieren Sie es, **schließen Sie es aber nicht!**
5. Starten Sie „Camtasia Recorder" über den Link auf Ihrem Desktop – **Starten Sie aber noch nicht die Aufnahme**.
6. Warten Sie!

Bitte erst nach Aufforderung umblättern!

Mit der nächsten Seite startet das Experiment.
Sie werden dann als erstes aufgefordert werden die aktuelle Uhrzeit aus der Taskleiste zu notieren und dann die Aufnahme in *Camtasia* zu starten!

Bevor Sie weiter lesen: Tragen Sie hier die aktuelle Uhrzeit aus Ihrer Taskbar ein (hh:mm): [:]

Starten Sie nun die Aufnahme mit Camtasia (Klick auf den roten Ball)
(der kleine Ball in der Taskleiste sollte nun zyklisch die Farbe von rot nach grün wechseln – beenden Sie Camtasia nun nicht mehr!)

Aufgabe: (Umstellung von Queue auf Topic)

Lesen Sie zuerst den gesamten Text (zwei Seiten) der Aufgabe gründlich!

Sie sollen die Aufgabe vor allem gründlich und richtig erledigen, nicht vor allem schnell. Sie haben genug Zeit!

In dieser Aufgabe sollen Sie Ihre außerhalb von *Xpetstore* laufende Klasse `XpetstoreEmailSpy` aus Übung 11 umbauen. Bis jetzt hat dieses Programm den Email-Versende-Mechanismus (`MailerMDB`) von *Xpetstore* verwendet um selber Emails zu versenden.

Die neu zu erstellende Version soll sich nun stattdessen als Mithörer in den Verkehr einklinken, der intern in *Xpetstore* an diesen Email-Versende-Mechanismus gerichtet ist.

Ihre Anwendung soll dabei ein Protokoll der über *Xpetstore* verschickten Emails auf die Konsole schreiben (`System.out`). Dabei soll jeweils der Empfänger und das *Subject* der Mail ausgegeben werden (z.B. „`Email To salinger@inf.fu-berlin.de: "[Petstore] Order Confirmation"`).

Beachten Sie dabei folgende Punkte:

- Dieser Umbau von `XpetstoreEmailSpy` kann nur funktionieren, wenn aus der *Queue*, über die derzeit der Versand von Emails in *Xpetstore* angestoßen wird, ein ***Topic*** wird, so dass mehrere Empfänger möglich sind. Sie müssen also die hierbei betroffenen Klassen von *Xpetstore* von *Queue* auf *Topic* umstellen bzw. so ergänzen, dass diese auch mit *Topics* umgehen können. Dies betrifft Änderungen im Kode sowie in den XDoclet-Anweisungen.
- **Sie müssen dafür sorgen, dass der JBoss ein *Topic* zur Verfügung stellt**, welches Sie im *Xpetstore* und im `XpetstoreEmailSpy` verwenden können. Bisher sind in JBoss nur *Queues* eingerichtet. Sie können aber die gleiche Syntax verwenden, um ein Topic zu erzeugen; die dabei einzutragende Klasse ist `org.jboss.mq.server.jmx.Topic`.
- Modifizieren Sie ansonsten alle Stellen, an denen auf *queue/mail* Bezug genommen wird; *queue/order* bleibt natürlich weiterhin eine *Queue*
- **Achtung**: Die beiden Interfaces `QueueConnectionFactory und TopicConnectionFactory` werden bei JBoss von der selben Klasse implementiert, deshalb ist auch der zu verwendende JNDI-Name gleich.
- **Testen** Sie die modifizierte Anwendung aus.
    - Starten Sie also JBoss (und somit den von Ihnen modifizierten *Xpetstore*)
    - Starten Sie ihre Applikation `XpetstoreEmailSpy`.
    - Führen Sie eine Bestellung in *Xpetstore* durch.
    - Ihr *XpetstoreEmailSpy* sollte nun eine Ausgabe erzeugen, die die Email des Bestellers und ein entsprechendes *Subject* enthält.

- **Hinweis 1**: Ihr Programm muss so gestaltet werden, dass es lange genug läuft, um auf evtl. eintreffende Nachrichten warten zu können. Hierzu können Sie die statische Methode **`sleep`**`(long millis)` aus `java.lang.Thread` verwenden.
- **Hinweis 2**: Es ist anscheinend nötig, die `MailerMDB` ihr Abonnement als „***NonDurable***" eintragen zu lassen, sonst gibt es ständige Fehlermeldungen von Jboss über „Null or empty subscription".

Ihre Lösung wird vom Übungsleiter nach der Fertigstellung kurz kontrolliert. Rufen Sie diesen hierzu durch Handzeichen zu sich. Aber nur wenn Sie sich sicher sind, dass Ihr Programm fertig ist. **Stoppen Sie aber nicht die Aufnahme oder Eclipse!**

In der Zeit, in der Sie auf den Übungsleiter warten, arbeiten Sie bitte nicht weiter an Ihrem Rechner bzw. Programm. Entspannen Sie sich!

Dieser Schritt kann mehrmals wiederholt werden (falls sich doch noch Fehler eingeschlichen haben). **Notieren Sie hierbei jeweils direkt vor dem Rufen des Übungsleiters die aus der Taskbar abgelesene Uhrzeit in der nachstehenden Tabelle.** Ebenso soll die Uhrzeit notiert werden, an der Sie ggf. die Arbeit wieder aufnehmen.

#	Zeitpunkt, an dem der Übungsleiter gerufen wurde (hh:mm)	Zeitpunkt, an dem die Arbeit wieder aufgenommen wurde (mm:hh)
1	:	:
2	:	:
3	:	:
4	:	:
5	:	:

Datum: _____ Name: _____Teilnehmernr.:_____

Fragebogen zum Experiment in BISJ2EE 2005/2006

Fragen zur Person

Der Fragebogen wird nur anonymisiert ausgewertet, die Teilnehmernummer dient nur dazu, die Antworten mit den passenden Aufzeichnungen aus dem Experiment verbinden zu können. Die Namensangabe dient nur zur Absicherung gegen eventuelle Fehler oder Lücken und wird entfernt, sobald die Zuordnung der Daten erreicht ist. Lediglich die Experimentatoren der AG Software Engineering bekommen Ihren Namen zu sehen.
Ihre Antworten haben keinerlei Einfluss auf ihre Bewertung!

Was studieren Sie (z.B. Bachelor Informatik) und in welchem Fach- und Gesamtsemester sind Sie?

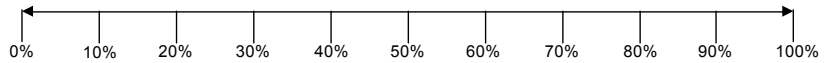
_____ P1
Abschluss Fach Fachsemester Hochschulsemester

Wie viele Jahre Programmiererfahrung in Java haben Sie etwa? _____ P2

In wie vielen Programmiersprachen haben Sie anwendbare Kenntnisse? _____ P3

Als wie geübt im Umgang mit der Entwicklungsumgebung Eclipse würden Sie sich selbst bezeichnen? (Schulnoten 1-6) _____ P5

Ordnen Sie sich selber auf der nachstehenden Skala aller Menschen, die programmieren können in eine passende Leistungsstufe ein: "Ich kann besser Software entwickeln als x% aller Programmierer der Welt." Machen Sie ein Kreuz:



P6

Welche Quellen und Dokumente ziehen Sie beim Programmieren in der Regel zu Rate, seien es Online-Recherchen, Bücher oder eigene Aufzeichnungen. Sie können hier nach Programmiersprachen untergliedern, falls nötig.

_____ P7

Bewerten Sie Ihre Kenntnisse in JMS (sowohl allgemein, als auch speziell bezüglich Queues bzw. Topics). (Schulnoten 1-6)

Allgemein: _____ P8.1

Queues: _____ P8.2

Topics: _____ P8.3

_____

Bewerten Sie Ihre Kenntnisse über xDoclet. (Schulnoten 1-6) _____ P9

Bewerten Sie Ihre Kenntnisse über xPetstore bzgl. der Bereiche, die im Rahmen der Veranstaltung betrachtet wurden.
(Schulnoten 1-6) _____ P10

Welche Teilaufgaben des Programmierens machen Ihnen am meisten Spaß (mehrere, auch eigene Antworten sind möglich)?

☐ Anforderungsanalyse	☐ Testen
☐ Spezifikation	☐ Eigene Ideen umsetzen
☐ Design	☐ Bibliotheken verwenden
☐ Coding	☐ Algorithmen entwickeln
☐ Debugging	☐ _____

_____P11

Fragen zur Zusammenarbeit

Die folgenden Fragen beziehen sich auf die Zusammenarbeit mit Ihrem festgelegten Übungsgruppenpartner:

Hatte Sie beide schon vorher gemeinsam programmiert? ☐ JA ☐ NEIN P12

Wenn ja: Wie regelmäßig ist Ihre Zusammenarbeit (in Stunden pro Woche über welchen Zeitraum, z.B. 2 Stunden pro Woche in den letzten 6 Monaten)?

_____ P13

Wenn nein: Arbeiten Sie auf einem anderen Gebiet gelegentlich oder häufig zusammen (z.B. gemeinsam Übungsaufgaben lösen)?

☐ JA ☐ NEIN P14

Haben Sie die Übungsaufgabe 11 (JMS) der BISJ2EE-Übung wirklich vollständig zusammen an einem Rechner bearbeitet? ☐ JA ☐ NEIN P15
(Niemand ist Ihnen böse, wenn Sie hier Nein antworten.)

Wie eingespielt ist die Zusammenarbeit mit Ihrem Partner. (Schulnoten 1-6) _____ P16

Kreuzen Sie an, welche Stärken Sie im Vergleich zu Ihrem Partner bei sich sehen

- ☐ Kreativität
- ☐ Genauigkeit
- ☐ Strukturiertes Vorgehen
- ☐ Schnelle Auffassungsgabe
- ☐ Kenntnisse über Algorithmen
- ☐ Erfahrung im Softwaredesign
- ☐ Erfahrung mit Softwarearchitekturen
- ☐ Gute Kenntnisse in Java
- ☐ Gute Kenntnisse in J2EE
- ☐ Erfahrung im Coding
- ☐ Kooperativität
- ☐ Geduld

_____ P17

Fragen zum Paarprogrammieren

Wissen Sie prinzipiell, was Paarprogrammieren (Pair Programming) ist? ☐ JA ☐ NEIN P18

Wenn ja: In wie vielen unterschiedlichen Paaren haben Sie schon regelmäßig zusammen programmiert?

_____ P19

Wenn ja: Welche Schritte der Softwareentwicklung haben Sie in der Regel in Ihren Paaren durchgeführt?

☐ Anforderungsanalyse ☐ Debugging

☐ Spezifikation ☐ Testen

☐ Design ☐ _____

☐ Coding ☐ _____

_____ P20

Gibt es Schritte bei der Softwareentwicklung, bei denen sich Ihre Paare in der Regel vorübergehend wieder getrennt haben? Welche sind dies?

☐ Anforderungsanalyse ☐ Debugging

☐ Spezifikation ☐ Testen

☐ Design ☐ _____

☐ Coding ☐ _____

_____ P21

Welche Vorteile haben Sie bei der Praxis der Paarprogrammierung erkannt? Wenn Sie in mehr als einem Paar gearbeitet haben: Sind diese Vorteile abhängig von Ihren Partnern?

_____ P22

Welche Nachteile haben Sie bei der Praxis der Paarprogrammierung erkannt? Wenn sie in mehr als einem Paar gearbeitet haben: Sind diese Nachteile abhängig von Ihren Partnern?

_____ P23

Um wie viel Prozent schneller oder langsamer können Sie typischerweise eine Aufgabe in einem Paar lösen, verglichen mit Alleinarbeit?

_____ ☐ schneller ☐ langsamer

Prozent P24

Datum: _____ Name: _____Teilnehmernr.:_____

Fragebogen nach dem Experiment in BISJ2EE 2005/2006

Der Fragebogen wird nur anonymisiert ausgewertet, die Teilnehmernummer dient nur dazu, die Antworten mit den passenden Aufzeichnungen aus dem Experiment verbinden zu können. Die Namensangabe dient nur zur Absicherung gegen eventuelle Fehler oder Lücken und wird entfernt, sobald die Zuordnung der Daten erreicht ist. Lediglich die Experimentatoren der AG Software Engineering bekommen Ihren Namen zu sehen.
Ihre Antworten haben keinerlei Einfluss auf ihre Bewertung!

Fragen zur Experimentaufgabe

Haben Sie die Aufgabe fertig gelöst? ☐ JA ☐ NEIN P01

Wie schwierig/leicht fanden Sie die zu lösende Aufgabe?
(sehr leicht, leicht, meinem Können/Wissen angemessen, ziemlich schwer, sehr schwer): _____ P02

Beschreiben Sie kurz die größten Schwierigkeiten mit der Aufgabe:

_____P03

Fragen zur Zusammenarbeit

Die folgenden Fragen beziehen sich auf die Zusammenarbeit mit Ihrem festgelegten Übungsgruppenpartner:

Haben Sie die Zusammenarbeit insgesamt als hilfreich empfunden?
☐ JA ☐ NEIN

P04

Welche speziellen Aspekte fanden Sie an der Zusammenarbeit hilfreich?

_____P05

Welche speziellen Aspekte haben Sie an der Zusammenarbeit gestört?

_____P06

Kreuzen Sie an, welche Stärken Sie nach der Bearbeitung der Aufgabe bei Ihrem Partner erkannt haben:

- ☐ Kreativität
- ☐ Genauigkeit
- ☐ Strukturiertes Vorgehen
- ☐ Schnelle Auffassungsgabe
- ☐ Kenntnisse über Algorithmen
- ☐ Erfahrung im Softwaredesign
- ☐ Erfahrung mit Softwarearchitekturen
- ☐ Gute Kenntnisse in Java
- ☐ Gute Kenntnisse in J2EE
- ☐ Erfahrung im Coding
- ☐ Kooperativität
- ☐ Geduld

P07

Als wie harmonisch haben Sie die Zusammenarbeit mit Ihrem Partner empfunden? (sehr harmonisch, harmonisch, neutral, weniger harmonisch, nicht harmonisch)

P08

Schlußfrage

Was möchten Sie dem Veranstalter sonst noch zum Experiment, zur Aufgabe, zum Setting oder ähnlichem sagen?:

Anhang

E Im Rahmen der Aufzeichnung von Sitzung PR1.1 ausgegebene Unterlagen

Auf den folgenden Seiten sind die im Rahmen der Aufzeichnungen der Sitzung PR1.1 ausgegebenen Fragebögen zu finden:

- S. 512: Der Fragebogen, der unmittelbar vor der Sitzung an jeden der beiden Entwickler ausgegeben wurde.
- S. 513 - S. 515: Der Fragebogen, der unmittelbar nach der Sitzung an jeden der beiden Entwickler ausgegeben wurde.

Die bei den anderen Beobachtungen von professionellen Entwicklern ausgegebene Fragebögen waren weitgehend identisch mit den hier dargestellten (siehe auch Anmerkungen auf S. 100).

Beschreibung der zu bearbeitenden Aufgabe(n)

Firma:_____ Datum:_____

Name Entwickler 1:_____ Name Entwickler 2:_____

Startzeit:_____ Endzeit:_____

1. Klassifizierung der zu bearbeitenden Aufgabenstellung(en):
☐ Neuentwicklung Funktionalität ☐ Weiterentwicklung Funktionalität
☐ „Debugging“ ☐ Testfallentwicklung
☐ Sonstiges: _____

2. Kurzbeschreibung der zu bearbeitenden Aufgabenstellung(en):

_____

_____

_____

_____

3. Die Aufgabenstellung hat folgende besondere Schwierigkeiten:

_____

_____

_____

4. Warum eignet sich die Aufgabe für die Paarprogrammierung?

_____

_____

_____

5. Seit wie viele Monate/Jahre praktizieren Sie regelmäßig (mehrere Male im Monat) Paarprogrammierung?

____ Jahre ____ Monate

6. Für wie eingespielt halten Sie sich mit Ihrem Partner (in Schulnoten)?

Note:___

Verlaufsbeschreibung Paarprogrammierungssitzung

Firma:_____ Datum:_____

Name Entwickler:_____ Name Partner: _____

Beachten Sie: Alle folgenden Fragen beziehen sich auf die gerade von Ihnen durchgeführte Paarprogrammierungssitzung und nicht auf Paarprogrammierung im Allgemeinen.

1. Bewerten Sie, in wiefern es hilfreich war die Aufgabe im Paar zu lösen (Schulnote)?

Note: ____ Grund:_____

2. Sind Sie mit der Bearbeitung der Aufgabe schneller vorangekommen als erwartet?

☐ Ja ☐ Nein ☐ wie erwartet

3. Bitte füllen Sie jetzt das Phasentemplate am Rechner aus.

4. Denken Sie kurz nach: Welches sind die drei bedeutendsten Phase, die Ihnen bez. der Sitzung einfallen? Bitte geben Sie eine kurze Begründung für Ihre Wahl an.

i. Phase: _____

Grund: _____

ii. Phase: _____

Grund: _____

iii. Phase: _____

Grund: _____

5. Bewerten Sie für die folgenden Aspekte, ob sie in dieser Sitzung eine Rolle spielten und ggf. wer von Ihnen beiden Gewinnbringendes zu diesem Aspekt beigetragen hat.

i. Vermittlung von Wissen

☐ spielte eine Rolle ☐ spielte keine Rolle

Ggf.: Gewinnbringende Beiträge

☐ überwiegend ich ☐ ausgeglichen ☐ überwiegend mein Partner

ii. Entwicklung einer (Lösungs-)Strategie

□ spielte eine Rolle □ spielte keine Rolle

Ggf.: Gewinnbringende Beiträge

□ überwiegend ich □ ausgeglichen □ überwiegend mein Partner

iii. Bugfixing

□ spielte eine Rolle □ spielte keine Rolle

Ggf.: Gewinnbringende Beiträge

□ überwiegend ich □ ausgeglichen □ überwiegend mein Partner

iv. Entwicklung von Softwaredesign/Architektur

□ spielte eine Rolle □ spielte keine Rolle

Ggf.: Gewinnbringende Beiträge

□ überwiegend ich □ ausgeglichen □ überwiegend mein Partner

v. Entwicklung von Algorithmen

□ spielte eine Rolle □ spielte keine Rolle

Ggf.: Gewinnbringende Beiträge

□ überwiegend ich □ ausgeglichen □ überwiegend mein Partner

vi. Kenntnisse von APIs

□ spielte eine Rolle □ spielte keine Rolle

Ggf.: Gewinnbringende Beiträge

□ überwiegend ich □ ausgeglichen □ überwiegend mein Partner

vii. „Den entscheidenden Einfall haben“

□ spielte eine Rolle □ spielte keine Rolle

Ggf.: Gewinnbringende Beiträge

□ überwiegend ich □ ausgeglichen □ überwiegend mein Partner

6. Gibt es noch weitere Aspekte, die beim Bearbeiten der Aufgabe(n) eine Rolle spielten? Geben Sie diese und ggf. wer von Ihnen beiden Gewinnbringendes zu diesem Aspekt beigetragen hat an.

i. Aspekt: _____

Ggf.: Gewinnbringende Beiträge

□ überwiegend ich □ ausgeglichen □ überwiegend mein Partner

ii. Aspekt: _____

Ggf.: Gewinnbringende Beiträge

□ überwiegend ich □ ausgeglichen □ überwiegend mein Partner

iii. Aspekt: _____

Ggf.: Gewinnbringende Beiträge

□ überwiegend ich □ ausgeglichen □ überwiegend mein Partner

7. Hätten Sie für bestimmte Teilschritte gerne die Paarkonstellation aufgehoben und alleine gearbeitet? Geben Sie ggf. die Teilschritte und einen kurzen Grund für Ihre Entscheidung an:

i. Schritt: _____

Grund: _____

ii. Schritt: _____

Grund: _____

iii. Schritt: _____

Grund: _____

Literaturverzeichnis

[1] Abran, A. (Hrsg.) ; Moore, J. W. (Hrsg.): *Guide to the Software Engineering Body of Knowledge*. IEEE Computer Society, 2004

[2] Acosta Chaparro, E. ; Yuksel, A. ; Romero, P. ; Bryant, S. : Factors Affecting the Perceived Effectiveness of Pair Programming in Higher Education. In: *Proceedings of the 17th Annual Workshop of the Psychology of Programming Interest Group*, 2005, S. 5–18

[3] Anderson, J. : *Language memory and thought*. Lawrence Erlbaum, 1976

[4] Arbaoui, S. ; Derniame, J.-C. ; Oquendo, F. ; Verjus, H. : A Comparative Review of Process-Centered Software Engineering Environments. In: *Annals of Software Engineering* 14 (2002), 311–340

[5] Arisholm, E. ; Gallis, H. ; Dybå, T. ; Sjøberg, D. I.: Evaluating Pair Programming with Respect to System Complexity and Programmer Expertise. In: *IEEE Transactions on Software Engineering* 33 (2007), Nr. 2, S. 65–86

[6] Atkinson, R. C. ; Shiffrin, R. M.: Human memory: A proposed system and its control processes. In: Spence, K. W. (Hrsg.) ; Spence, J. T. (Hrsg.): *The psychology of learning and motivation* Bd. 2. Academic Press, 1968, S. 89–195

[7] Austin, J. L.: *Zur Theorie der Sprechakte (How to do things with words)*. 2. Reclam, 1989

[8] Baheti, P. ; Gehringer, E. ; Stotts, D. : Exploring the Efficacy of Distributed Pair Programming. In: *Extreme Programming and Agile Methods – XP/Agile Universe 2002* Bd. 2418, Springer, 2002 (Lecture Notes in Computer Science), S. 387–410

[9] Ballstaedt, S.-P. : Kognition und Wahrnehmung in der Informations- und Wissensgesellschaft. In: Kübler, H.-D. (Hrsg.) ; Elling, E. (Hrsg.): *Wissensgesellschaft*. Bundeszentrale für politische Bildung, 2005

[10] Balzert, H. : *Lehrbuch der Software-Technik, Software-Management, Software-Qualitätssicherung, Unternehmensmodellierung*. Heidelberg : Spektrum Akademischer Verlag, 1998

[11] Balzert, H. : *Lehrbuch der Software-Technik, Software-Entwicklung*. 2. Heidelberg : Spektrum Akademischer Verlag, 2001

[12] Beck, K. : *Extreme Programming Explained: Embrace Change*. Addison-Wesley Professional, 1999

[13] Beck, K. : *Extreme Programming : Die revolutionäre Methode für Softwareentwicklung in kleinen Teams ; [Das Manifest]*. Addison-Wesley, 2003

[14] Beck, K. : *Extreme Programming Explained: Embrace Change, Second Edition*. Addison-Wesley Professional, 2004

[15] Bedny, G. ; Meister, D. : *The Russian Theory of Activity: Current Applications To Design and Learning*. Lawrence Erlbaum, 1997

[16] Bedny, G. Z. ; Harris, S. R.: The Systemic-Structural Theory of Activity: Applications to the Study of Human Work. In: *Mind, Culture, and Activity* 12 (2005), Nr. 2, S. 128–147

[17] Berg, C. ; Milmeister, M. : Im Dialog mit den Daten das eigene Erzählen der Geschichte finden. Über die Kodierverfahren der Grounded Theory Methodologie. In: *Historical Social Research, Supplement* (2007), Nr. 19, S. 182–210

[18] Bevan, J. ; Werner, L. ; McDowell, C. : Guidelines for the Use of Pair Programming in a Freshman Programming Class. In: *Proceedings of the 15th Conference on Software Engineering Education and Training*. IEEE Computer Society (CSEET '02), 100–107

[19] Blickle, G. : Interaktion und Kommunikation. In: Schuler, H. (Hrsg.): *Organisationspsychologie - Gruppe und Organisation - Enzyklopädie der Psychologie: Wirtschafts-, Organisations- und Arbeitspsychologie - Band 4*. Göttingen : Hogrefe, 2004, S. 55–128

[20] Bloom, B. S. (Hrsg.): *Taxonomy of Educational Objectives: The Classification of Educational Goals (Handbook 1: Cognitive Domain)*. Longman, 1956

[21] Blumer, H. : What is Wrong with Social Theory. In: *American Sociological Review* 18 (1954), S. 3–10

[22] Boehm, B. ; Turner, R. : Observations on Balancing Discipline and Agility. In: *Proceedings of the Conference on Agile Development*. IEEE Computer Society (ADC '03)

[23] Boehm, B. W.: *Software Engineering Economics*. Prentice Hall, 1981

[24] Boehm, B. W.: A spiral model of software development and enhancement. In: *SIGSOFT Softw. Eng. Notes* 11 (1986), Aug., Nr. 4, 14–24

[25] Boehm, B. W.: A spiral model of software development and enhancement. In: *IEEE Computer* 21 (1988), Nr. 5, S. 61–72

[26] Bogdan, R. C. ; Biklen, S. K.: *Qualitative Research for Education. An Introduction to Theories and Methods*. 2. Boston : Allyn & Bacon, 1992

[27] Böhm, R. : Prädikation - primär (und sekundär) notionalgrammatisch. In: *Bremer Linguistik Workshop*, 2001

[28] Bortz, J. ; Döring, N. : *Forschungsmethoden und Evaluation für Human- und Sozialwissenschaftler*. 4. Heidelberg : Springer Medizin Verlag, 2006

[29] Bratthall, L. ; Arisholm, E. ; Jørgensen, M. : Program Understanding Behavior during Estimation of Enhancement Effort on Small Java Programs. In: *Proceedings of the Third International Conference on Product Focused Software Process Improvement*. Springer-Verlag (PROFES '01), 356–370

[30] Breuer, F. : *Reflexive Grounded Theory: Eine Einführung für die Forschungspraxis*. Vs Verlag, 2010

[31] Brinker, K. ; Sager, S. F.: *Linguistische Gesprächsanalyse*. 5. Erich Schmidt Verlag, 2010

[32] Brooks, F. P.: *The Mythical Man-Month. Essays on Software Engineering*. Addison-Wesley, 1975

[33] Bryant, S. ; Romero, P. ; Boulay, B. du: The Collaborative Nature of Pair Programming. In: *Extreme Programming and Agile Processes in Software Engineering* Bd. 4044, Springer, 2006 (Lecture Notes in Computer Science), S. 53–64

[34] Bryant, S. ; Romero, P. ; Boulay, B. du: Pair Programming and the Mysterious Role of the Navigator. In: *International Journal of Human-Computer Studies* (2008)

[35] Bühler, K. : Tatsachen und Probleme zu einer Psychologie der Denkvorgänge. Über Gedanken. In: *Archiv für Psychologie* 9 (1907), S. 297–365

[36] Bühler-Niederberger, D. : Analytische Induktion als Verfahren der qualitativen Methodologie. In: *Zeitschrift für Soziologie* 14 (1985), Nr. 6, S. 475–485

[37] Bussmann, H. (Hrsg.): *Lexikon der Sprachwissenschaft*. 4. Kröner, 2008

[38] Butler, G. ; McManus, F. : *Psychologie: Eine Einführung*. Reclam, 2003

[39] Canfora, G. ; Cimitile, A. ; Garcia, F. ; Piattini, M. ; Visaggio, C. A.: Evaluating performances of pair designing in industry. In: *J. Syst. Softw.* 80 (2007), Nr. 8, S. 1317–1327

[40] Canfora, G. ; Cimitile, A. ; Visaggio, C. A.: Empirical Study on the Productivity of the Pair Programming. In: *Extreme Programming and Agile Processes in Software Engineering* Bd. 3556, Springer, 2005 (Lecture Notes in Computer Science), S. 92–99

[41] Cao, L. ; Xu, P. : Activity Patterns of Pair Programming. In: *Proc. of the 38th Annual Hawaii International Conf. on System Sciences (HICSS 2005)*. IEEE Computer Society, 88a

[42] Chalmers, A. F.: *Wege der Wissenschaft*. Springer, 1989. – Original erschienen 1976: What is this thing called science?

[43] Chao, J. ; Atli, G. : Critical Personality Traits in Successful Pair Programming. In: *AGILE 2006*. Los Alamitos, CA, USA : IEEE Computer Society, 2006, S. 89–93

[44] Chi, M. T. H.: Quantifying Qualitative Analyses of Verbal Data: A Practical Guide. In: *Journal of Learning Sciences* 6 (1997), Nr. 3, S. 271–315

[45] Choi, K. S. ; Deek, F. P. ; Im, I. : Exploring the underlying aspects of pair programming: The impact of personality. In: *Information and Software Technology* 50 (2008), October, 1114–1126

[46] Chong, J. ; Hurlbutt, T. : The Social Dynamics of Pair Programming. In: *Proceedings of the 29th International Conference on Software Engineering*. IEEE Computer Society (ICSE '07), 354–363

[47] Clausewitz, C. v.: *Vom Kriege*. Insel Verlag, 2005

[48] Colemana, G. ; O'Connor, R. : Using grounded theory to understand software process improvement: A study of Irish software product companies. In: *Information and Software Technology* 49 (2007), Nr. 6, S. 654–667

[49] Conradia, R. ; Babarb, M. A.: Controlled Experiments on Pair Programming: Making Sense of Heterogeneous Results. In: *Norsk informatikkonferanse (NIK-2010)*, 2010

[50] Creswell, J. W.: *Research design: Qualitative, quantitative, and mixed methods approaches*. 2. Thousand Oaks, Calif. : Sage Publ., 2003

[51] Dalton, M. : Preconceptions and Methods in Men Who Manage. In: Hammond, P. E. (Hrsg.): *Sociologists at Work: Essays on the Craft of Social Research*. New York : Basic Books, 1964

[52] Dick, A. J. ; Zarnett, B. : Paired Programming and Personality Traits. In: *Third International Conference on (eXtreme) Programming and Agile Processes in Software Engineering (XP2002)*, 2002

[53] Dölling, J. : *Sprechakte und Konversation*. Version: 2008. `http://www.uni-leipzig.de/~doelling/veranstaltungen/semprag12.pdf`, Abruf: 2009-02-27

[54] Domino, M. A. ; Collins, R. W. ; Hevner, A. R.: Controlled experimentation on adaptations of pair programming. In: *Information Technology and Management* 8 (2007), Nr. 4, S. 297–312

[55] Ehlich, K. ; Rehbein, J. : Halbinterpretative Arbeitstranskriptionen (HIAT). In: *Linguistische Berichte* 45 (1976), S. 21–41

[56] Ehlich, K. ; Rehbein, J. : Erweiterte halbinterpretative Arbeitstranskriptionen (HIAT2): Intonation. In: *Linguistische Berichte* 59 (1979), S. 51–75

[57] Ehlich, K. ; Rehbein, J. : Zur Notierung nonverbaler Kommunikation für diskursanalytische Zwecke (HIAT2). In: *Methoden der Analyse von Face-To-Face-Situationen*. Stuttgart : Metzler, 1979, S. 302–329

[58] Ericsson, K. A. ; Simon, H. A.: Verbal Reports as Data. In: *Psychological Review* 87 (1980), Nr. 3, S. 215–250

[59] Ericsson, K. A. ; Simon, H. A.: *Protocol Analysis: Verbal Reports as Data*. 1st paperback. Cambridge, Massachusetts : MIT Press, 1993

[60] Ernst, G. : *Einführung in die Erkenntnistheorie*. WBG, 2007

[61] Fagan, M. E.: Design and code inspections to reduce errors in program development. In: *IBM Systems Journal* 15 (1976), Nr. 3, S. 182–211

[62] Flick, U. : Methodenangemessene Gütekriterien in der qualitativ-interpretativen Forschung. In: Bergold, J. (Hrsg.) ; Flick, U. (Hrsg.): *Ein-Sichten: Zugänge zur Sicht des Subjekts mittels qualitativer Forschung*. dgvt-Verl., 1987, S. 247–262

[63] Foppa, K. : Über Regeln der sprachlichen Kommunikation und die Schwierigkeiten, nichtberichtbares Wissen zu erfassen. In: *Bulletin der Schweizer Psychologen* 4 (1990), S. 3–12

[64] Forsyth, D. R.: *Group Dynamics*. 5. Cengage Learning Emea, 2010

[65] Fowler, M. : *Refactoring: Improving the Design of Existing Code*. Addison-Wesley Longman, 1999

[66] Freudenberg, S. ; Romero, P. ; Boulay, B. du: "Talking the talk": Is intermediate-level conversation the key to the pair programming success story? In: *AGILE 2007*. IEEE Computer Society, 84–91

[67] Fuggetta, A. : Software process: a roadmap. In: *ICSE '00: Proceedings of the Conference on The Future of Software Engineering*. New York, NY, USA : ACM, 2000, S. 25–34

[68] Gallis, H. ; Arisholm, E. ; Dybå, T. : An Initial Framework for Research on Pair Programming. In: *Proc. of the 2003 Int'l. Symposium on Empirical Software Engineering*, IEEE CS press, 2003

[69] Gamm, G. : *Flucht aus der Kategorie: Die Positivierung des Unbestimmten als Ausgang der Moderne*. Suhrkamp, 1994

[70] Gamma, E. ; Helm, R. ; Johnson, R. J. ; Vlissides, J. : *Design Patterns. Elements of Reusable Object-Oriented Software*. Addison-Wesley Longman, 1995

[71] Gettier, E. L.: Is Justified True Belief Knowledge? In: *Analysis* 23 (1963), Nr. 6, S. 121–123

[72] Glaser, B. G.: The Constant Comparative Method of Qualitative Analysis. In: *Social Problems* 12 (1965), Nr. 4, S. 436–445

[73] Glaser, B. G.: *Theoretical Sensitivity: Advances in the Methodology of Grounded Theory*. Mill Valley, CA : Sociology Press, 1978

[74] Glaser, B. G.: *Emergence vs. Forcing: Basics of Grounded Theory Analysis*. Mill Valley, CA : Sociology Press, 1992

[75] Glaser, B. G. ; Strauss, A. L.: *Awareness of Dying*. Chicago : Aldine, 1965

[76] Glaser, B. G. ; Strauss, A. L.: *The Discovery of Grounded Theory: Strategies for Qualitative Research*. New York : Aldine de Gruyter, 1967

[77] Glaser, B. G. ; Strauss, A. L.: *Grounded Theory: Strategien qualitativer Forschung*. 2. Bern : Hans Huber, 2005. – Original erschienen 1967: The Discovery of Grounded Theory: Strategies for Qualitative Research.

[78] Glasersfeld, E. von: *Wege des Wissens: Konstruktivistische Erkundungen durch unser Denken*. Carl-Auer-Systeme, 1997

[79] Glasersfeld, E. von: *Radikaler Konstruktivismus: Ideen, Ergebnisse, Probleme*. 2. Suhrkamp, 1998

[80] Goffman, E. : *Das Individuum im öffentlichen Austausch: Mikrostudien zur öffentlichen Ordnung*. Suhrkamp, 1974

[81] Goldberg, L. R.: A broad-bandwidth, public-domain, personality inventory measuring the lower-level facets of several five-factor models. In: *Personality Psychology in Europe* 7 (1999), S. 7–28

[82] Goldberg, L. R. ; Johnson, J. A. ; Eber, H. W. ; Hogan, R. ; Ashton, M. C. ; Cloninger, C. R. ; Gough, H. G.: The international personality item pool and the future of public-domain personality measures. In: *Journal of Research in Personality* 40 (2006), Nr. 1, S. 84–96

[83] Good, J. ; Brna, P. : Program comprehension and authentic measurement: a scheme for analysing descriptions of programs. In: *International Journal of Human-Computer Studies* 61 (2004), Nr. 2, S. 169–185

[84] Gottschalk-Mazouz, N. : Was ist Wissen? Überlegungen zu einem Komplexbegriff an der Schnittstelle von Philosophie und Sozialwissenschaften. In: Ammon, S. (Hrsg.) ; Heineke, C. (Hrsg.) ; Selbmann, K. (Hrsg.): *Wissen in Bewegung*. Velbrück Wissenschaft, 2007, S. 21–40

[85] Grice, P. : Logic and Conversation. In: *Syntax and Semantics* 3 (1975), S. 41–58

[86] Hanks, B. ; McDowell, C. ; Draper, D. ; Krnjajic, M. : Program Quality with Pair Programming in CS1. In: *ITiCSE '04: Proceedings of the 9th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education*. New York, NY, USA : ACM Press, 2004, S. 176–180

[87] Hannay, J. E. ; Arisholm, E. ; Engvik, H. ; Sjøberg, D. I. K.: Effects of Personality on Pair Programming. In: *IEEE Transactions on Software Engineering* 36 (2010), Jan., Nr. 1, 61–80

[88] Hannay, J. E. ; Dybå, T. ; Arisholm, E. ; Sjøberg, D. I.: The effectiveness of pair programming: A meta-analysis. In: *Information and Software Technology* 51 (2009), Nr. 7, S. 1110–1122

[89] Hanson, N. R.: *Patterns of Discovery: An Inquiry into the Conceptual Foundations of Science*. Cambridge University Press, 1958

[90] Hartmann, P. : *Mathematik für Informatiker: Ein praxisbezogenes Lehrbuch*. 5. Vieweg+Teubner Verlag, 2012

[91] Hayes, J. : *The complete problem solver*. 2. Lawrence Earlbaum Associates, 1989

[92] Hazzan, O. ; Dubinsky, Y. : Bridging Cognitive and Social Chasms in Software Development Using Extreme Programming. In: Marchesi, M. (Hrsg.) ; Succi, G. (Hrsg.): *Extreme Programming and Agile Processes in Software Engineering* Bd. 2675, Springer Berlin Heidelberg (Lecture Notes in Computer Science), 47–53

[93] Heiberg, S. ; Puus, U. ; Salumaa, P. ; Seeba, A. : Pair-Programming Effect on Developers Productivity. In: *Extreme Programming and Agile Processes in Software Engineering* Bd. 2675, Springer, 2003 (Lecture Notes in Computer Science), S. 215–224

[94] Helbig, G. : Partikeln als illokutive Indikatoren im Dialog. In: *Deutsch als Fremdsprache* 4 (1977), S. 30–44

[95] Helbig, G. : *Deutsch als Fremdsprache. Ein internationales Handbuch*. Bd. 1. de Gruyter, 2001

[96] H.M.Sneed: *Software-Management*. Müller GmbH, 1987

[97] Höfer, A. : Video analysis of pair programming. In: *Proceedings of the 2008 international workshop on Scrutinizing agile practices or shoot-out at the agile corral*. ACM (APOS '08), 37–41

[98] Hughes, J. ; Parkes, S. : Trends in the use of verbal protocol analysis in software engineering research. In: *Behaviour and Information Technology* 22 (2003), Nr. 2, S. 127–140

[99] Janis, I. L.: *Groupthink: Psychological Studies of Policy Decisions and Fiascoes*. 2. Cengage Learning, 1982

[100] Jensen, R. W.: A Pair Programming Experience. In: *CrossTalk: A Journal of Defense Software Engineering* 16 (2003), Nr. 3

[101] Kampenes, V. B. ; Dybå, T. ; Hannay, J. E. ; Sjøberg, D. I.: A systematic review of effect size in software engineering experiments. In: *Information and Software Technology* 49 (2007), Nr. 11–12, S. 1073–1086

[102] Katira, N. ; Williams, L. ; Osborne, J. : Towards Increasing the Compatibility of Student Pair Programmers. In: *Proceedings of the 27th international conference on Software engineering*. New York, NY, USA : ACM, 2005 (ICSE '05), S. 625–626

[103] Katira, N. ; Williams, L. ; Wiebe, E. ; Miller, C. ; Balik, S. ; Gehringer, E. : On understanding compatibility of student pair programmers. In: *SIGCSE Bulletin* 36 (2004), Nr. 1, S. 7–11

[104] Kelle, U. : *Empirisch begründete Theoriebildung: Zur Logik und Methodologie interpretativer Sozialforschung*. 2. Weinheim : Deutscher Studienverlag, 1997

[105] Kelle, U. : "Emergence" vs. "Forcing" of Empirical Data? A Crucial Problem of "Grounded Theory" Reconsidered. In: *Forum Qualitative Sozialforschung / Forum: Qualitative Social Research* 6 (2005), Nr. 2

[106] Kelle, U. : Theoretisches Vorwissen und Kategorienbildung in der "Grounded Theory". In: *Qualitative Datenanalyse: computergestützt*. 2. VS Verlag für Sozialwissenschaften, 2007, S. 32–49

[107] Kelle, U. ; Kluge, S. : *Vom Einzelfall zum Typus*. Leske + Budrich, 1999

[108] Kirk, J. ; Miller, M. L.: *Reliability and Validity in Qualitative Research*. Sage Publications, 1986

[109] Kosz, A. : *Wissenschafts- und erkenntnistheoretische Grundlagen des Wissensmanagements: Michael Polanyis Konzept des „Tacit Knowing"*, Fachhochschule Eisenstadt, Diplomarbeit, 2007

[110] Krauss, R. M. ; Fussell, S. R.: Social Psychological Models of Interpersonal Communication. In: Higgins, E. T. (Hrsg.) ; Kruglanski, A. W. (Hrsg.): *Social Psychology: Handbook of Basic Principles*. New York : Guilford, 1996, S. 655–701

[111] Kuckartz, U. : *Einführung in die computergestützte Analyse qualitativer Daten*. VS Verlag für Sozialwissenschaften, 2005

[112] Kutschera, F. : *Grundfragen der Erkenntnistheorie*. De Gruyter, 1981

[113] Kvale, S. : Validity in the qualitative research interview. In: *Psykologisk Skriftserie Aarhus* 12 (1987), Nr. 1, S. 68–104

[114] Layman, L. : Changing Students' Perceptions: An Analysis of the Supplementary Benefits of Collaborative Software Development. In: *Proceedings of the 19th Conference on Software Engineering Education & Training*. IEEE Computer Society (CSEET '06), 159–166

[115] Lefrançois, G. R.: *Psychologie des Lernens*. 4. Springer, 2006

[116] Lehner, F. : *Wissensmanagement : Grundlagen, Methoden und technische Unterstützung*. 3. Carl Hanser, 2009

[117] Lenzen, W. : 80. Propositionale Einstellung. In: Dascal, M. (Hrsg.) ; Gerhardus, D. (Hrsg.) ; Lorenz, K. (Hrsg.) ; Meggle, G. (Hrsg.): *Sprachphilosophie* Bd. 7.2. Walter de Gruyter, 1995, S. 1175–1187

[118] Leontev, A. N.: *Activity, Consciousness, and Personality*. Prentice-Hall, 1978. – http://lchc.ucsd.edu/MCA/Paper/leontev/index.html

[119] Lewandowski, T. : *Linguistisches Wörterbuch*. Bd. 1. 5. Quelle & Meyer, 1990

[120] Lewandowski, T. : *Linguistisches Wörterbuch*. Bd. 2. 5. Quelle & Meyer, 1990

[121] Lim, K. H. ; Ward, L. M. ; Benbasat, I. : An Empirical Study of Computer System Learning: Comparison of Co-Discovery and Self-Discovery Methods. In: *Information Systems Research* 8 (1997), Nr. 3, S. 254–272

[122] Linke, A. ; Nussbaumer, M. ; Portmann, P. R.: *Studienbuch Linguistik*. 5. Tübingen : Niemeyer, 2004

[123] Ludwig-Mayerhofer, W. : *ILMES - Internet-Lexikon der Methoden der empirischen Sozialforschung*. Version: 2009. `http://www.lrz-muenchen.de/~wlm/ilmes.htm`, Abruf: 2009-08-11

[124] Madeyski, L. : The Impact of Pair Programming and Test-Driven Development on Package Dependencies in Object-Oriented Design – An Experiment. In: Münch, J. (Hrsg.) ; Vierimaa, M. (Hrsg.): *Product Focused Software Process Improvement* Bd. 4034, 2006 (Lecture Notes in Computer Science), S. 278–289

[125] Madeyski, L. : On the Effects of Pair Programming on Thoroughness and Fault-Finding Effectiveness of Unit Tests. In: Münch, J. (Hrsg.) ; Abrahamsson, P. (Hrsg.): *Product Focused Software Process Improvement* Bd. 4589, 2007 (Lecture Notes in Computer Science), S. 207–221

[126] Mahner, M. ; Bunge, M. : *Philosophische Grundlagen der Biologie*. Springer, 2000

[127] Martin, R. C. ; Computer, P. H. (Hrsg.): *Agile Software Development. Principles, Patterns, and Practices*. 2002

[128] Mayrhauser, A. v. ; Lang, S. : A Coding Scheme to Support Systematic Analysis of Software Comprehension. In: *IEEE Transactions on Software Engineering* 25 (1999), Nr. 4, S. 526–540

[129] Mayrhauser, A. v. ; Vans, A. : Identification of Dynamic Comprehension Processes During Large Scale Maintenance. In: *IEEE Transactions on Software Engineering* 22 (1996), S. 424–437

[130] Mayring, P. : Die qualitative Wende: Grundlagen, Techniken und Integrationsmöglichkeiten qualitative Forschung in die Psychologie. In: *Bericht über den 36. Kongress der DGfPs in Berlin*, 1989, S. 306–313

[131] Mayring, P. : *Qualitative Inhaltsanalyse: Grundlagen und Techniken*. Deutscher Studienverlag, 1993

[132] Mayring, P. : *Einführung in die qualitative Sozialforschung: Eine Anleitung zu qualitativem Denken*. 5. Beltz Verlag, 2002

[133] McDowell, C. ; Werner, L. ; Bullock, H. ; Fernald, J. : The Effects of Pair Programming on Performance in an Introductory Programming Course. In: *Proceedings of the 33rd SIGCSE technical symposium on Computer science education*, ACM Press, 2002, S. 38–42

[134] McDowell, C. ; Werner, L. ; Bullock, H. E. ; Fernald, J. : The Impact of Pair Programming on Student Performance, Perception, and Persistance. In: *ICSE '03: Proc. 25th Int'l Conf. on Software Engineering*, IEEE Computer Society, 2003, S. 602–607

[135] Mead, G. H.: Die objektive Realität der Perspektiven. In: Joas, H. (Hrsg.): *George Herbert Meat: Gesammelte Aufsätze* Bd. 2. Frankfurt a. M.. : Suhrkamp, 1987 <1927>, S. 211–224

[136] Merten, K. : *Inhaltsanalyse: Einführung in Theorie, Methode und Praxis.* Opladen : Westdeutscher Verlag, 1995

[137] Merton, R. K.: *Social Theory and Social Structure.* Glencoe, Ill. : Free Press of Glencoe, 1949

[138] Mey, G. ; Mruck, K. : Grounded Theory Methodologie — Bemerkungen zu einem prominenten Forschungsstil. In: *Historical Social Research, Supplement* (2007), Nr. 19, S. 11–39

[139] Mey, G. ; Mruck, K. : Methodologie und Methodik der Grounded Theory. In: Kempf, W. (Hrsg.) ; Kiefer, M. (Hrsg.): *Forschungsmethoden der Psychologie. Zwischen naturwissenschaftlichem Experiment und sozialwissenschaftlicher Hermeneutik* Bd. 3. Berlin : Regener, 2009, S. 100–152

[140] Muckel, P. : Die Entwicklung von Kategorien mit der Methode der Grounded Theory. In: *Historical Social Research, Supplement* (2007), Nr. 19, S. 211–231

[141] Mühlmeyer-Mentzel, A. ; Schümann, I. : Softwareintegrierte Lehre der Grounded-Theory-Methodologie. In: *Forum Qualitative Sozialforschung / Forum: Qualitative Social Research* 12 (2011), Nr. 3

[142] Müller, M. M.: Two controlled experiments concerning the comparison of pair programming to peer review. In: *Journal of Systems and Software* 78 (2005), Nr. 2, 166–179

[143] Müller, M. M.: A preliminary study on the impact of a pair design phase on pair programming and solo programming. In: *Inf. Softw. Technol.* 48 (2006), Mai, Nr. 5, 335–344

[144] Multhaup, U. : *Deklaratives und prozedurales Wissen, explizites und implizites Wissen.* Version: 2002. `http://www2.uni-wuppertal.de/FB4/anglistik/multhaup/brain_language_learning/`

html/brain_memory_stores/6_declarative_procedural_txt.html, Abruf: 2010-01-15

[145] Myers, C. ; Davids, K. : Knowing and Doing. Tacit Skill at Work. In: *Personnel Management* 24 (1992), Nr. 2, S. 45–47

[146] Myers, I. B.: *Gifts Differing : Understanding Personality Type*. Davies-Black Publishing, 1980

[147] Myers, I. B.: *Introduction to Type: A Guide to Understanding Your Results on the Myers-Briggs Type Indicator*. 6. Center for Applications of Psychological Type, 1998

[148] Nagappan, N. ; Williams, L. ; Ferzli, M. ; Wiebe, E. ; Yang, K. ; Miller, C. ; Balik, S. : Improving the CS1 Experience with Pair Programming. In: *Proceedings of the 34th SIGCSE technical symposium on Computer science education*. New York, NY, USA : ACM Press, 2003, S. 359–362

[149] Nagappan, N. ; Williams, L. A. ; Wiebe, E. ; Miller, C. ; Balik, S. ; Ferzli, M. ; Petlick, J. : Pair Learning: With an Eye Toward Future Success. In: *Extreme Programming and Agile Methods - XP/Agile Universe 2003* Bd. 2753, Springer, 2003 (Lecture Notes in Computer Science), S. 185–198

[150] Nawrocki, J. ; Wojciechowski, A. : Experimental Evaluation of Pair Programming. In: *Proc. European Software Control and Metrics Conference (ESCOM'01)*, 2001

[151] Nerdinger, F. W. ; Blickle, G. ; Schaper, N. : *Arbeits- und Organisationspsychologie*. Springer, 2008

[152] Neuweg, G. H.: *Könnerschaft und implizites Wissen*. Waxmann, 1999

[153] Nonaka, I. ; Takeuchi, H. : *Die Organisation des Wissens*. Campus, 1997. – Original erschienen 1995: The Knowledge-Creating Company.

[154] Nosek, J. T.: The case for collaborative programming. In: *Communications of the ACM* 41 (1998), Nr. 3, S. 105–108

[155] Oates, B. J.: *Researching Information Systems and Computing*. Sage Publ., 2006

[156] Okada, T. ; Simon, H. : Collaborative Discovery in a Scientific Domain. In: *Cognitive Science* 21 (1997), Nr. 2, S. 109–146

[157] Osterweil, L. J.: Unifying Microprocess and Macroprocess Research. In: *Unifying the Software Process Spectrum* Bd. 3840, Springer Berlin / Heidelberg, 2006 (Lecture Notes in Computer Science), S. 68–74

[158] Owen, S. ; Budgen, D. ; Brereton, P. : Protocol analysis: a neglected practice. In: *Commun. ACM* 49 (2006), Nr. 2, S. 117–122

[159] Park, W.-W. : A review of research on Groupthink. In: *Journal of Behavioral Decision Making* 3 (1990), Nr. 4, S. 229–245

[160] Pennington, N. : Comprehension strategies in programming. In: G. Olson, S. S. . E. S. (Hrsg.): *Empirical Studies of Programmers: Second Workshop*. Norwood, NJ, USA : Ablex Publishing Corp., 1987, S. 100–113

[161] Pfleeger, S. L. ; Atlee, J. M.: *Software Engineering*. Prentice Hall, 2006

[162] Phongpaibul, M. ; Boehm, B. : An empirical comparison between pair development and software inspection in Thailand. In: *Proceedings of the 2006 ACM/IEEE international symposium on Empirical software engineering*. ACM (ISESE '06), 85–94

[163] Phongpaibul, M. ; Boehm, B. : A Replicate Empirical Comparison between Pair Development and Software Development with Inspection. In: *ESEM '07: Proceedings of the First International Symposium on Empirical Software Engineering and Measurement*. Washington, DC, USA : IEEE Computer Society, 2007, S. 265–274

[164] Piaget, J. : *The Construction of Reality in the Child*. Basic Books, 1954

[165] Platon ; Martens, E. (Hrsg.): *Theätet*. Reclam, 1986

[166] Plonka, L. ; Segal, J. ; Sharp, H. ; Linden, J. van d.: Investigating Equity of Participation in Pair Programming. In: *Agile India*. Los Alamitos, CA, USA : IEEE Computer Society, 2012, S. 20–29

[167] Plonka, L. ; Segal, J. ; Sharp, H. ; van der Linden, J. : Collaboration in Pair Programming: Driving and Switching. In: *XP 2011: 12th International Conference on Agile Software Development*, 2011

[168] Polanyi, M. : *Personal Knowledge. Towards a Post-Critical Philosophy*. Routledge, 1958

[169] Polanyi, M. : *The tacit dimension*. Garden City, 1966

[170] Polya, G. : *Schule des Denkens: Vom Lösen mathematischer Probleme*. Francke, 1995. – Original erschienen 1949: How to solve it. A new aspect of mathematical method.

[171] Redder, A. : Aufbau und Gestaltung von Transkriptionssystemen. In: Brinker, K. (Hrsg.) ; Antos, G. (Hrsg.) ; Heinemann, W. (Hrsg.) ; Sager, S. F. (Hrsg.): *Text- und Gesprächslinguistik / Linguistics of Text and Conversation* Bd. 2. de Gruyter, 2001, S. 1038–1058

[172] Rehbein, J. ; Schmidt, T. ; Meyer, B. ; Watzke, F. ; Herkenrath, A. : Handbuch für das computergestützte Transkribieren nach HIAT. In: *Arbeiten zur Mehrsprachigkeit, Folge B* 56 (2004)

[173] Rehfus, W. D. (Hrsg.): *Handwörterbuch Philosophie*. Vandenhoeck & Ruprecht, 2003

[174] Reichertz, J. : Abduktives Schlußfolgern und Typen(re)konstruktion: Abgesang auf eine liebgewonnene Hoffnung. In: Jung, T. (Hrsg.) ; Müller-Doohm, S. (Hrsg.): *»Wirklichkeit« im Deutungsprozeß*. Suhrkamp, 1993, S. 258–282

[175] Robillard, P. N. ; d'Astous, P. ; Détienne, F. ; Visser, W. : Measuring cognitive activities in software engineering. In: *ICSE '98: Proc. 20th Int'l Conf. on Software Engineering*. Washington, DC, USA : IEEE Computer Society, 1998, S. 292–299

[176] Rostaher, M. ; Hericko, M. : Tracking Test First Pair Programming – An Experiments. In: Wells, D. (Hrsg.) ; Williams, L. (Hrsg.): *Extreme Programming and Agile Methods – XP/Agile Universe 2002* Bd. 2418, Springer Berlin Heidelberg (Lecture Notes in Computer Science), 174–184

[177] Rowley, J. : The wisdom hierarchy: representations of the DIKW hierarchy. In: *Journal of Information Science* 33 (2007), Nr. 2, S. 163–180

[178] Royce, W. W.: Managing the development of large software systems. In: *Proc. IEEE WESCON*, 1970

[179] Ryle, G. : *The Concept of Mind*. Hutchinson, 1949

[180] Salinger, S. ; Plonka, L. ; Prechelt, L. : A Coding Scheme Development Methodology Using Grounded Theory for Qualitative Analysis of Pair Programming. In: *Proceedings of the 19th Annual Workshop of the Psychology of Programming Interest Group*. Joensuu, Finland, 2007, S. 144–157. – www.ppig.org

[181] Salinger, S. ; Plonka, L. ; Prechelt, L. : A Coding Scheme Development Methodology Using Grounded Theory for Qualitative Analysis of Pair Programming. In: *Human Technology: An Interdisciplinary Journal on Humans in ICT Environments* 4 (2008), S. 9–25

[182] Salinger, S. ; Prechelt, L. : What happens during Pair Programming? In: *Proceedings of the 20th Annual Workshop of the Psychology of Programming Interest Group (PPIG '08)*. Lancaster, England, September 2008. – www.ppig.org

[183] Salleh, N. ; Mendes, E. ; Grundy, J. ; Burch, G. S. J.: An empirical study of the effects of personality in pair programming using the five-factor model. In: *ESEM '09: Proceedings of the 3rd International Symposium on Empirical Software Engineering and Measurement*. Washington, DC, USA : IEEE Computer Society, 2009, S. 214–225

[184] *Kapitel* 3: Kognition. In: Schaade, G. : *Demenz*. Springer, 2009, S. 19–25

[185] Schlesinger, F. ; Jekutsch, S. : ElectroCodeoGram: An Environment for Studying Programming. In: *Workshop on Ethnographies of Code*. Lancaster, UK, 2006

[186] Schultz, W. : Lernen, Gedächtnis und Gehirn. In: *Universitas Friburgensis* (1993), Mai. `http://www.unifr.ch/spc/UF/93mai/schultz.html`

[187] Schulz-Hardt, S. : Die große Illusion - Zur Synergie in Gruppen. In: *Forschung & Lehre* 12 (2012), Nr. 9, S. 744–745

[188] Schwaber, K. ; Beedle, M. : *Agile Software Development with Scrum*. Prentice Hall, 2002

[189] Searle, J. : *Sprechakte*. Suhrkamp, 1971

[190] Searle, J. : Indirect Speech Acts. In: Cole, P. (Hrsg.) ; Morgan, J. L. (Hrsg.): *Syntax and Semantics: Vol. 3: Speech Acts* Bd. 3. San Diego, CA : Academic Press, 1975, S. 59–82

[191] Sfetsos, P. ; Stamelos, I. ; Angelis, L. ; Deligiannis, I. : Investigating the Impact of Personality Types on Communication and Collaboration-Viability in Pair Programming – An Empirical Study. In: Abrahamsson, P. (Hrsg.) ; Marchesi, M. (Hrsg.) ; Succi, G. (Hrsg.): *Extreme Programming and Agile Processes in Software Engineering* Bd. 4044, Springer Berlin Heidelberg (Lecture Notes in Computer Science), 43–52

[192] Sfetsos, P. ; Stamelos, I. ; Angelis, L. ; Deligiannis, I. : An experimental investigation of personality types impact on pair effectiveness in pair programming. In: *Empirical Software Engineering* 14 (2009), Nr. 2, S. 187–226

[193] Sommerville, I. : *Software Engineering - 6. Auflage*. Pearson Studium, 2001

[194] Sommerville, I. : *Software Engineering; eighth edition*. Addison-Wesley, 2007

[195] Squire, L. R. ; Kandel, E. R.: *Gedächtnis: Die Natur des Erinnerns*. Heidelberg : Spektrum Akademischer Verlag, 1999

[196] Strauss, A. L.: *Qualitative Analysis for Social Scientists*. Cambridge : Cambridge University Press, 1987

[197] Strauss, A. L.: *Grundlagen qualitativer Sozialforschung: Datenanalyse und Theoriebildung in der empirischen und soziologischen Forschung*. Wilhelm Fink Verlag, 1994. – Original erschienen 1987: Qualitative Analysis for Social Scientists.

[198] Strauss, A. L. ; Corbin, J. : *Basics of Qualitative Research: Grounded Theory Procedures and Techniques*. London : Sage Publications, Inc., 1990

[199] Strauss, A. L. ; Corbin, J. : *Grounded Theory: Grundlagen Qualitativer Sozialforschung*. Weinheim : BELTZ, Psychologie Verlags Union, 1996. – Original erschienen 1990: Basics of Qualitative Research: Grounded Theory Procedures and Techniques.

[200] Strauss, A. L. ; Schatzman, L. ; Bucher, R. ; Ehrlich, D. ; Sabshin, M. : *Psychiatric Ideologies and Institutions*. London : Collier-Macmillan, 1964

[201] Strübing, J. : *Grounded Theory: Zur sozialtheoretischen und epistemologischen Fundierung des Verfahrens der empirisch begründeten Theoriebildung*. Vs Verlag, 2004

[202] Strübing, J. : Zwei Varianten von Grounded Theory? Zu den methodologischen und methodischen Differenzen zwischen Barney Glaser und Anselm Strauss. In: Mey, G. (Hrsg.) ; Mruck, K. (Hrsg.): *Grounded Theory Reader* Bd. 2. Vs Verlag, 2011

[203] Suddaby, R. : From the Editors: What Grounded Theory is Not. In: *Academy of Management Journal* 49 (2006), Nr. 4, S. 633–642

[204] Sutcliffe, A. G. ; Maiden, N. A. M.: Analysing the novice analyst: cognitive models in software engineering. In: *International Journal of Man-Machine Studies* 36 (1992), Nr. 5, S. 719–740

[205] Tessem, B. : Experiences in Learning XP Practices: A Qualitative Study. In: Marchesi, M. (Hrsg.) ; Succi, G. (Hrsg.): *Extreme Programming and Agile Processes in Software Engineering* Bd. 2675, Springer Berlin Heidelberg (Lecture Notes in Computer Science), 131–137

[206] Theis, A. M.: *Organisationskommunikation: Theoretische Grundlagen und empirische Forschungen*. Opladen, 1994

[207] Thomas, L. ; Ratcliffe, M. ; Robertson, A. : Code warriors and code-a-phobes: a study in attitude and pair programming. In: *Proceedings of the 34th SIGCSE technical symposium on Computer science education*. ACM (SIGCSE '03), 363–367

[208] Thun, F. Schulz v.: *Miteinander reden 1: Störungen und Klärungen*. rororo, 2010

[209] Truschkat, I. ; Kaiser, M. ; Reinartz, V. : Forschen nach Rezept? Anregungen zum praktischen Umgang mit der Grounded Theory in Qualifikationsarbeiten. In: *Forum Qualitative Sozialforschung / Forum: Qualitative Social Research* 6 (2005), Nr. 2

[210] Udy, S. H. J.: Cross-Cultural Analysis: A Case Study. In: Hammond, P. E. (Hrsg.): *Sociologists at Work: Essays on the Craft of Social Research*. Basic Books, 1964

[211] VanDeGrift, T. : Coupling pair programming and writing: learning about students' perceptions and processes. In: *Proceedings of the 35th SIGCSE technical symposium on Computer science education*. ACM (SIGCSE '04), 2–6

[212] Vanhanen, J. ; Lassenius, C. : Effects of pair programming at the development team level: an experiment. In: *Proceedings of the International Symposium on Empirical Software Engineering*. Los Alamitos, CA, USA : IEEE Computer Society, 2005, S. 336–345

[213] Watson, J. B.: Psychology as the Behaviorist Views it. In: *Psychological Review* 20 (1913), S. 158–177

[214] Watzlawick, P. ; Beavin, J. H. ; Jackson, D. D.: *Menschliche Kommunikation. Formen, Störungen, Paradoxien*. 3. Huber, 1972

[215] Wermke, M. (Hrsg.) ; Klosa, A. (Hrsg.) ; Kunkel-Razum, K. (Hrsg.) ; Scholze-Stubenrecht, W. (Hrsg.): *Duden Band 1*. 22. Dudenverlag, 2000

[216] Williams, L. : Integrating Pair Programming into a Software Development Process. In: *Proceedings of the 14th Conference on Software Engineering Education and Training (CSEET 2001)*. IEEE Computer Society

[217] Williams, L. ; Kessler, B. : The Effects of "Pair-Pressure" and "Pair-Learning" on Software Engineering Education. In: *CSEET '00: Proceedings of the 13th Conference on Software Engineering Education & Training*. Washington, DC, USA : IEEE Computer Society, 2000, S. 59

[218] Williams, L. ; Kessler, R. : *Pair Programming Illuminated*. Addison-Wesley Professional, 2002

[219] Williams, L. ; Kessler, R. ; Cunningham, W. ; Jeffries, R. : Strengthening the Case for Pair Programming. In: *IEEE Software* 17 (2000), Nr. 4, S. 19–25

[220] Williams, L. ; Layman, L. ; Osborne, J. ; Katira, N. : Examining the Compatibility of Student Pair Programmers. In: *AGILE 2006*. Los Alamitos, CA, USA : IEEE Computer Society, 2006, S. 411–420

[221] Williams, L. ; Wiebe, E. ; Yang, K. ; Ferzli, M. ; Miller, C. : In Support of Pair Programming in the Introductory Computer Science Course. In: *Computer Science Education* 12 (2002), Nr. 3, S. 197–212

[222] Windelband, W. : Geschichte und Naturwissenschaft. In: *Präludien: Aufsätze und Reden zur Philosophie und ihrer Geschichte* Bd. 2. Mohr, 1915

[223] Wittgenstein, L. ; Schulte, J. (Hrsg.): *Philosophische Untersuchungen*. Suhrkamp, 2001

[224] Wray, S. : How Pair Programming Really Works. In: *IEEE Software* 27 (2010), Nr. 1, S. 50–55

[225] Wunderlich, D. : *Grundlagen der Linguistik*. Rowohlt, 1974

[226] Wunderlich, D. : *Studien zur Sprechakttheorie*. Suhrkamp, 1976

[227] Xu, S. ; Rajlich, V. : Dialog-based protocol: an empirical research method for cognitive activities in software engineering. In: *International Symposium on Empirical Software Engineering (ISESE 2005)*. IEEE Computer Society, 383–392

[228] Xu, S. ; Rajlich, V. ; Marcus, A. : An empirical study of programmer learning during incremental software development. In: *Fourth IEEE Conf. on Cognitive Informatics (ICCI 2005)*. IEEE Computer Society, 340–349

[229] Xu, S. ; Rajlich, V. : Empirical Validation of Test-Driven Pair Programming in Game Development. In: *5th IEEE/ACIS International Conference on Computer and Information Science and 1st IEEE/ACIS International Workshop on Component-Based Software Engineering, Software Architecture and Reuse (ICIS-COMSAR'06)* (2006), S. 500–505

[230] Yngve, V. H.: On getting a word in edgewise. In: *Papers from the Sixth Regional Meeting of the Chicago Linguistic Society*. Chicago, 1970, S. 567–578

[231] Zieris, F. : *Qualitative Untersuchungen von Beeinflussungen der Aktivitäten des Partners bei der Paarprogrammierung*, Freie Universität Berlin, Masterarbeit, 2012