Masterarbeit am Institut für Informatik der Freien Universität Berlin

Arbeitsgruppe Software Engineering

Gemeinsame Wissensproduktion in der Paarprogrammierung

Yannick Gerhard Pascal Schütt
Matrikelnummer: 4642778

Betreuer: Franz Zieris, M.Sc
Gutachter: Prof. Dr. Lutz Prechelt
Zweitgutachterin: Prof. Dr. Elfriede Fehr

Berlin, den 3. November 2014

Zusammenfassung

Als Paarprogrammierung bezeichnet man die Praktik, bei der zwei Personen zusammen und mit vereinten Kräften an einem Softwareartefakt arbeiten - zumeist Quellcode. Die Tatsache, dass Paarprogrammierung im Gegensatz zur Soloprogrammierung die doppelten Kosten zu verursachen scheint hat zu zahlreichen quantitativen Studien geführt, die bemerkenswert inhomogene Ergebnisse geliefert haben. Auf Grund dieser Tatsache erscheint es sinnvoll den Paarprogrammierungsprozess durch qualitative Forschung erst einmal in der Tiefe zu verstehen, bevor weitere quantitative Studien durchgeführt werden. Eine der Thesen warum Paarprogrammierung insgesamt sogar kostengünstiger sein könnte als Soloprogrammierung ist der Wissenstransfer der bedingt durch die Kommunikation während der Paarprogrammierung unvermeidlich stattfindet. Eben der Wissenstransfer und dessen unterliegende Prozesse stehen im Fokus der vorliegenden Arbeit. Genauer beschrieben ist die Thematik mit der sich die vorliegenden Arbeit befasst die gemeinsame Wissensproduktion in der Paarprogrammierung, also ein Aspekt des Wissenstransfers.
Die Ergebnisse dieser Arbeit sind mittels des qualitativen Forschungsstils Grounded Theory Methodology erlangt worden. Die Ergebnisse dieser Arbeit sind vier Konzepte, die die gemeinsame Wissensproduktion in der Paarprogrammierung konzeptualisieren. Darüber hinaus gibt diese Arbeit eine Methodik an, an Hand derer sich ein systematisches Problemlösen von einem vermeintlich weniger geordneten Vorgehen in Phasen der gemeinsamen Wissensproduktion in der Paarprogrammierung unterscheiden lässt.

Inhaltsverzeichnis

1 Einleitung **1**

2 Stand der Forschung **3**
2.1 Paarprogrammierung & Pair Programming Skill 3
2.2 Anekdotische Berichte 4
2.3 Quantitative Studien 4
2.4 Qualitative Studien 5
2.5 Forschungsfrage 6

3 Methodik **8**
3.1 Grundgedanken & Passung an den Forschungsgegenstand 8
3.2 Grounded Theory Method (Strauss & Corbin (1990)) 9
3.2.1 Offenes Kodieren 9
3.2.2 Axiales Kodieren 11
3.2.2.1 Paradigmatisches Handlungsmodell 12
3.2.2.2 Techniken zur Erhöhung der theoretischen Sensibilität 13
3.2.3 Selektives Kodieren 15
3.2.4 Sonstige Praktiken der GTM 16
3.2.5 Mögliche Schwierigkeiten 17
3.2.6 4 Erweiterungen der GTM durch Prechelt et al. 17

4 Daten und Datenerhebung **20**

5 Qualitative Forschung der AGSE **23**
5.1 Basisschicht 23
5.2 Wissenstransfer 30

6 Ergebnisse **33**
6.1 Problembild-Äußerungen 39
6.2 Lösungsbild-Äußerungen 43
6.3 Relevanz-Äußerungen 45
6.4 TOPIC-Veränderungen 47
6.5 Funktionen der Konzepte 50
6.5.1 Funktionen von Problembild-Äußerungen 50
6.5.2 Funktionen von Lösungsbild-Äußerungen 51
6.5.3 Funktionen von Relevanz-Äußerungen 51
6.5.4 Funktionen von TOPIC-Veränderungen 53
6.6 Zusammenspiel der Konzepte 53
6.7 Sonstige Konzepte 65

7 Validität und Güte **67**

8 Zusammenfassung & Ausblick **69**

Anhang **76**

A Transkriptionssystem **76**

B Eidesstattliche Erklärung **77**

Literaturverzeichnis **78**

Abbildungsverzeichnis

1 Grounded Theory Method – Offenes Kodieren – Eigenschaften und Dimensionen der Kategorie „Beobachten“ 11
2 Grounded Theory Method – Axiales Kodieren – Paradigmatisches Handlungsmodell 13
3 Erweiterungen der Grounded Theory Method – Namensschema für die Benennung von Konzepten 18
4 Grounded Theory Method – Erweiterungen – Metamodell der Analysemethoden und -objekte 19
5 Screenshot der Paarprogrammierungssitzung BA1 – (14:14) . . . 21
6 Softwarewerkzeug ATLAS.ti Screenshot der Sitzung BA1 – (6:20-10:46) mit Konzeptannotationen 22
7 Basisschicht der qualitativen Paarprogrammierungsforschung – Schichtenmodell 24
8 Basisschicht der qualitativen Paarprogrammierungsforschung – Vereinfachte Darstellung 26
9 Basisschicht der qualitativen Paarprogrammierungsforschung – Objektbestandteile 27
10 Basisschicht der qualitativen Paarprogrammierungsforschung – Verbenbestandteile 28
11 Basisschicht der qualitativen Paarprogrammierungsforschung – Gesamtdarstellung 29
12 Zeitliche Verteilung von Aktivitäten einer Paarprogrammierungssitzung 30
13 Wissenstransferkonzepte – Zusammenfassung 32
14 Beispiel eines Problemlösungsbaums 54
15 Thematischer Gesprächsverlauf der Sitzung BA1 – (1:33 - 12.30) 56
16 Kategorisierter Gesprächsverlauf der Sitzung BA1 – (1:33 - 12.30) 58
17 Thematischer Gesprächsverlauf der Sitzung CA2.2 – (32:57 - 36:36) 63
18 Kategorisierter Gesprächsverlauf der Sitzung CA2.2 – (32:57 - 36:36) 64
19 Konzepte gemeinsamer Wissensproduktion in der Paarprogrammierung – Zusammenfassung 70
20 Negatives Beispiel eines kategorisierten Gesprächsverlaufs 73
21 Positives Beispiel eines kategorisierten Gesprächsverlaufs 74
22 Pragmatisches Halbinterpretatives Transkriptionssystem für Paarprogrammierungssitzungen 76

1 Einleitung

Die Art und Weise der Softwareentwicklung hat sich in den letzten Jahrzehnten erheblich verändert. Im Allgemeinen wird Abstand genommen von plangetriebenen Prozessmodellen wie klassischer Weise dem Wasserfallmodell oder RUP (Rational Unified Process [27]), hin zu agilen Entwicklungsmethoden wie Extreme Programming [11], Scrum [37], oder der Crystal-Familie [16], bei denen Adaptivität über Prädiktion und Iteration über irreversibles Vorgehen gestellt wird. Kent Beck beschreibt den Extreme Programming (XP) Prozess mit seinem Buchtitel wie folgt: „Embrace Change“ [12].
Beim Softwareetwicklungsprozess Extreme Programming ist Paarprogrammierung (PP) eine vorgeschriebene Praktik. Die Frage die seither natürlich auf der Hand liegt ist ob die doppelten Kosten, verursacht durch die beiden Entwickler, durch die Qualität der Software oder die Entwicklungsgeschwindigkeit wieder aufgewogen werden können, oder gar überholen können. Einer der größten postulierten Vorteile der Paarprogrammierung ist dabei neben den höheren Codequalität, die zeitlich kürzere Entwicklungsdauer und der Wissenstransfer zwischen den Partnern. Der Fragestellung über die Effizienz der Paarprogrammierung hat in der Vergangenheit zu zahlreichen Studien geführt, die bemerkenswert inhomogene Ergebnisse offenbart haben. In der Folge ist der Forschungsansatz der Arbeitsgruppe Software Engineering (AGSE) des Instituts für Informatik an der Freien Universität Berlin den Prozess der Paarprogrammierung durch qualitative Forschung zunächst in der Tiefe verstehen zu können, bevor man weitere Messungen vornimmt. Als Arbeitsgruppe der praktischen Informatik gilt ihr Interesse dabei nicht allein der Entdeckung der unterliegenden Prozesse während der Paarprogrammierung, vielmehr geht es darum Softwareentwicklern den *Pair Programming Skill* verständlich und erlernbar zu machen, ein Artefakt dessen Existenz Kent Beck dem PP-Prozess bereits 1999 unterstellte

> „Pair programming [...] [is] a subtle skill.“’ [11]

Dies soll in einer Art geschehen wie auch Programmdesign durch Entwurfsmuster [23] erlernbar gemacht wird. Dabei ist das langfristige Ziel der Forschung der AGSE den Pair Programming Skill durch Verhaltensmuster erlernbar zu machen.

Auf dem Weg zu diesem gewünschten Regelwerk hat bereits Stephan Salinger seine Promotion „Ein Rahmenwerk für die qualitative Analyse der Paarprogrammierung“ [34] geschrieben. Auf dieser Arbeit wird die meinige inhaltlich aufbauen. Derzeit gibt es ein weiteres Promotionsvorhaben in der Arbeitsgruppe, dass sich mit dem Wissenstransfer in der Paarprogrammierung befassen wird. Meine Arbeit ist also im Kontext der Forschung der Arbeitsgruppe Software Engineering, so wie den beiden vorgenannten Arbeiten zu sehen.

Um die gemeinsame Wissensproduktion in der Paarprogrammierung also tiefergehend verstehen zu können, wird zuerst der Stand der Forschung und die daraus resultierende Forschungsfrage aufgezeigt werden (Kapitel 2). Im Anschluss daran wird die verwendete Forschungsmethode und die Passung dieser Methodik an den Forschungsgegenstand dargelegt (Kapitel 3), bevor anschließend das zugrundeliegende Datenmaterial so wie die verwendeten Softwaretools beschrieben werden (Kapitel 4). Kapitel 5 wird die relevanten Teile der bisherigen qualitativen Forschung der AGSE darlegen, insbesondere das Rahmenwerk für die qualitative Analyse der Paarprogrammierung und bisherige Erkenntnisse über den Wissenstransfer in der Paarprogrammierung. Kapitel 6 offenbart die Ergebnisse der hier vorliegenden Arbeit. Dies geschieht an Hand eines einführenden Beispiels aus dem die vier Konzepte exemplarisch hergeleitet werden können. Im Anschluss daran wird auf das Zusammenspiel dieser Konzepte und der Frage, ob es eine Systematik gibt mit der Paarprogrammierer gemeinsamen Wissen produzieren fokussiert. In Kapitel 7 werden die Ergebnisse hinsichtlich ihrer Validität kritisch betrachtet, bevor sie in Kapitel 8 eingeordnet und dessen Tragweite aufgezeigt wird. Ferner wird in diesem Kapitel ein Ausblick auf offen gebliebene Aspekte und weitere mögliche Forschungsarbeiten gegeben.

2 Stand der Forschung

In diesem Kapitel wird der Stand der Forschung der Paarprogrammierung zusammengefasst und dargestellt. Dazu wird zuerst erklärt was Paarprogrammierung ist und weitere Grundbegriffe der Paarprogrammierungsforschung dargelgt. Im Anschluss daran werden anekdotische Belege für die positiven Effekte der PP dargelegt. Darauf folgend werden die Ergebnisse strukturierter Beobachtungen dargelegt, die des besseren Verständnisses halber in quantitative und qualitative Ansätze unterteilt werden. Zum Abschluss dieses Kapitels werden Schlussfolgerungen aus diesem aktuellen Stand der Forschung gezogen, die zugleich die Forschungsfrage der vorliegenden Arbeit darlegt.

2.1 Paarprogrammierung & Pair Programming Skill

Als Paarprogrammierung (PP) bezeichnet man den Prozess, bei dem zwei Personen zusammen und mit vereinten Kräften an allen Aspekten eines Artefakts arbeiten. Dieses Artefakt ist dabei nicht beschränkt auf Programmcode, es kann sich ebenfalls um einen architektonischen Entwurf oder einen Algorithmus handeln. Williams et al. [42] beschreiben PP wie folgt:

> "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."

Kent Beck beschreibt PP als einen Dialog zwischen den beiden Partnern die simultan programmieren, analysieren und testen.

> "Pair programming is a dialog between two people simultaneously programming (and analyzing and designing and testing) and trying to program better." [12]

Zudem hat er zu ergänzen, dass PP eine Praktik ist, die mehr verlangt als die bloßen Programmierkenntnisse der beiden Partner. Er sagt PP ist eine feinfühlige Fähigkeit -

> "Pair Programming [...] [is] a subtle skill."[11]

Er postuliert also eine speziell erlernbare Fähigkeit für Paarprogrammierung, den so genannten „Pair Programming Skill".

Durch die Konstellation in der PP stattfindet, für gewöhnlich gibt es lediglich eine Tastatur und eine Maus, entstanden Begriffe die die aktuelle Rollenverteilung der beiden Partner charakterisieren. Diese sind *Driver* und *Observer*. Der

Driver ist derjenige der an der Tastatur und Maus sitzt und das Artefakt bearbeitet während der Observer derjenige ist, der die Ausführung des Drivers stetig kontrolliert. Williams et al. [42] beschreiben dies wie folgt:

> „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] 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."

2.2 Anekdotische Berichte

Wie eingangs erwähnt existieren anekdotische Berichte über die Praktik der Paarprogrammierung, welche zumeist sehr positiv ausfallen. Diese Berichte betreffen verschiedene Aspekte der PP, die auch strukturierte Beobachtungen untersucht haben.

Jeffries et al. berichten, dass Paarprogrammierer mehr Code produzieren als Soloentwickler. Davon abgesehen produzieren sie Code von besserer Qualität. Darüber hinaus stellen sie fest, dass Paarprogrammierer länger arbeiten können ohne zu ermüden [26].

Williams et al. stellen sogar fest, dass Paarprogrammierer mehr als doppelt so schnell arbeiten wie Soloprogrammierer. Darüber hinaus konsultieren sie mehr als doppelt so viele Lösungsmöglichkeiten wie Soloentwickler. Außerdem stellen sie fest, dass es erstaunlich sei wie viele offensichtliche Defekte eine zweite Person entdecke, die von Soloentwicklern unentdeckt geblieben wären [43].

McBreen berichtet ebenfalls von besserer Codequalität. Es gäbe nur ein geringes Risiko hoher Defektraten. Dies begründet er damit, dass PP wie ein permanents Code-Review[1] wirke [29].

2.3 Quantitative Studien

Der Ansatz der quantitativen Forschung ist die Paarprogrammierung als Black Box zu betrachten und an den Eingängen, den unabhängigen Variablen, zu manipulieren um die Ergebnisse aus den Ausgängen, den abhängigen Variablen, zu messen und miteinander in Relation zu setzen.

Bezüglich der Entwicklungsgeschwindigkeit fand Nosek [31], dass die Soloprogrammierer im Durchschnitt 41% länger als die Paarprogrammierer benötigten.

[1] Code Review ist eine statische Testmethode. Dabei wird ein gewisses Artefakt der Softwareentwicklung von einer anderen Person als dem Entwickler des Artefakts nach Fehlern und Verbesserungsmöglichkeiten durchgesehen. Da es sich um eine statische Testmethode handelt, wird das Artefakt nicht ausgeführt, sondern lediglich z.B. der Quellcode angesehen.

Dabei handelte es sich jedoch um eine statisch nicht signifikante Abweichung. Arisholm et al. [9] kamen hingegen zu dem Ergebnis, dass die Paare „nur“ 8% schneller waren. Zwei der Studien zeigten sogar einen negativen Effekt der PP auf die Entwicklungsgeschwindigkeit, also eine Verlangsamung [10],[13].

Ebenfalls wurden Studien durchgeführt, die Aussagen über die Defektraten von Code von Einzelprogrammierern gegenüber Paarprogrammierern machen. Auch hier kommt es zu außerordentlich inhomogenen Ergebnissen: Williams et al. [42] bekamen mehr bestandene Testfälle als Ergebnis, wohingegen Vanhanen [41] zu dreimal so vielen Defekten wie die Solo-Vergleichsgruppe kommt.

Des Weiteren hat es Studien zur Codequalität als Ganzes gegeben. Williams et al. [42] kamen zu dem Ergebnis, dass die Anzahl als korrekt durchlaufender Testfälle bei Paarprogrammierern signifikant höher sind als bei Soloentwicklern. Arisholm et al. [9] kamen hingegen nur zu einer Verbesserung von 7% gegenüber den Soloentwicklern. Domino et al. [18] kommen zu dem Ergebnis, dass PP sogar einen negativen Effekt auf die Codequalität hat. Sie sind jedoch die einzigen, die zu einem solchen Ergebnis kommen.

Ein weiterer Aspekt der bisherigen quantitativen Forschung der Paarprogrammierung ist der Spaß an der Arbeit bzw. das Vertrauen in die erstellte Lösung. Hierbei wurden in der Regel Aussagen von Paarprogrammierern mit solchen von Soloprogrammierern verglichen. Hierbei handelt es sich um die wenigen Ergebnisse, die ein einheitliches Bild liefern: Paarprogrammierer haben im Vergleich zu Soloprogrammierern mehr Spaß an ihrer Arbeit und ebenfalls ein größeres Vertrauen in die von ihnen erstellten Lösung [10],[18],[30],[31], [36].

Das Fazit aus Sicht der AGSE ist, dass die Heterogenität der vorgenannten (und ebenfalls einer Vielzahl hier nicht aufgeführter) Studien dadurch zu erklären ist, dass der Pair Programming Skill in allen Studien konsequent ignoriert wurde. In gewisser Weise wurden also salopp formuliert Birnen mit Äpfeln verglichen, da gewissermaßen Anfänger der Fähigkeit der Paarprogrammierung nicht von Experten dieser Fähigkeit unterschieden worden sind.

2.4 Qualitative Studien

Entgegen des Ansatzes der quantitativen Forschung besteht die Methodik der qualitativen Forschung darin die Black Box der quantitativen Forschung zu öffnen um zu verstehen, was in ihr passiert und auf diese Weise Erkenntnisse über die unterliegenden Prozesse der Paarprogrammierung zu erlangen.

An dieser Stelle werden insbesondere Studien genannt die nicht von der AGSE sind, da solche im weiteren Verlauf (Kapitel 5) der vorliegenden Arbeit genauer dargelegt werden, da sie das Fundament für die vorliegende Arbeit bilden und besser nachzuvollziehen sind wenn sowohl die Forschungsmethode als auch das Datenmaterial näher erläutert wurden.

Bezüglich der Fertigkeiten und Erfahrung und dessen Einfluss auf eine PP-Sitzung berichteten Cao und Xu [14] über ihre explorative Studie zu Aktivitätsmustern der PP (activity patterns of pair programming):

> „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 [...]
>
> 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."

Bezüglich der weitläufig vorhandenen (und weiter oben ebenfalls dargestellten) Unterteilung der beiden Programmierer einer PP-Sitzung in Driver und Observer berichteten Chong und Hurlbutt [15] folgendes:

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

An dieser Stelle sei ferner auf eine vergleichbare Studie mit dem Titel „Liberating Pair Programming Research from the Oppressive Driver/Observer Regime" der AGSE verwiesen [35].

2.5 Forschungsfrage

Der Stand der Forschung lässt sich folgendermaßen zusammenfassen: Anekdotische Berichte zeichneten ein äußerst positives Bild der Effekte der Paarprogrammierung. Quantitative Studien lieferten bemerkenswert inhomogene Ergebnisse. Dies ist mit einer Gemeinsamkeit die alle Studien verbindet zu erklären, nämlich der Ignoranz des Pair Programming Skills.

Erklärungsmodelle für das Zustandekommen dieser Inhomogenität liegen kaum vor. Ganzheitliche Erklärungsversuche für die unterliegenden Prozesse der Paarprogrammierung liegen bisher gar nicht vor.

Schlussendlich kann also festgestellt werden, dass ein sehr großer Bereich der Theorie zur Praktik der PP zu füllen ist. Eben einen dieser Bereiche versucht die AGSE durch qualitative Paarprogrammierungsforschung zu erhellen. Eben in dieses Vorhaben reiht sich auch die vorliegende Arbeit ein, da sie versucht einen zentralen Aspekt der PP zu erhellen, den Wissenstransfer. Da es sich jedoch

bei dieser Arbeit lediglich um eine Masterarbeit handelt kann ein solch großer Themenkomplex unmöglich vollständig abgedeckt werden, weswegen sich diese Arbeit auch mit einem Aspekt des Wissenstransfers beschäftigt, nämlich der gemeinsamen Wissensproduktion. Die zu Grunde liegende Forschungsfrage der vorliegenden Arbeit lässt sich also folgendermaßen zusammenfassen:

Wie produzieren Paarprogrammierer gemeinsam Wissen?

3 Methodik

Dieses Kapitel stellt die verwendete Methodik für die Ergebnisse der vorliegenden Arbeit dar. Dazu wird zu Beginn ein kurzer Überblick über die Grundgedanken der verwendeten Forschungsmethode dargelegt um im Anschluss daran die Passung der Methodik an den Forschungsgegenstand beurteilen zu können. Anschließend wird die verwendete Methodik ausführlich dargestellt, so wie deren mögliche Risiken beleuchtet.

3.1 Grundgedanken & Passung an den Forschungsgegenstand

Die verwendete Methodik ist die Grounded Theory Method (GTM) nach Anselm Strauss und Juliet Corbin aus dem Jahr 1990 [38]. Bei der GTM handelt es sich um eine Alternative zum sequentiellen Forschungsprozess der zumeist aus Hypothesengenerierung, Methodenprüfung, Datenerhebung, Datenauswertung und Verifikation bzw. Falsifikation der Hypothese(n) besteht. Das Verfahren der GTM ist nicht dafür konzipiert um eine vorhandene Theorie zu verifizieren, vielmehr geht es bei Anwendung der GTM darum zu identifizieren, welche Konzepte und Hypothesen für den Bereich, den man untersuchen möchte, überhaupt relevant sind.

> „One does not begin with a theory, then prove it. Rather, one begins with an area of study and what is relevant to that area is allowed to emerge.“ ([38], S.23)

Der Grundsatz der GTM ist, dass alle gemachten Aussagen ausschließlich in den untersuchten Daten verankert sind, woher auch die Namensgebung *grounded* kommt. 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“ [24].

Es handelt sich um einen Forschungsstil in dem die theoretischen Vorannahmen und die gewonnen Daten in einem permanenten Dialog miteinander sind [8].

Durch die soeben beschriebenen Charakteristika und dadurch, dass die GTM keinerlei Annahmen über den Forschungsgegenstand macht, einen behavioristischen Ansatz verfolgt und zur Erzeugung von neuem theoretischen Wissen ausgelegt ist, ist sie um den Forschungsgegenstand zu untersuchen sehr gut geeignet und wird daher im Folgenden ausführlich dargelegt.

3.2 Grounded Theory Method (Strauss & Corbin (1990))

Die GTM stammt ursprünglich aus der Disziplin der Sozialforschung und wurde anfangs insbesondere im Krankenhausmilieu verwendet um beispielsweise Erkenntnisse über Handlungsstrategien bei Risikoschwangerschaften zu erlangen. GTM gibt es in zwei verschiedenen Strömungen. Das Vorhandensein zweier verschiedener Strömungen der gleichen Theorie rührt daher, dass es zwischen den beiden Erfindern dieser Methodik Barney Glaser und Anselm Strauss nach ihrer ersten gemeinsamen Publikation mit dem Titel „The Discovery of Grounded Theory: Strategies for Qualitative Research“ (1967) zu einem ideologischen Bruch kam, woraufhin die beiden von dort an nur noch getrennt publizierten. Zur genaueren Ausführung auf diesen ideologischen Streit sei hier auf [40] verwiesen.

Die beiden Strömungen unterscheiden sich unter anderem in der Art wie sie mit Vorkenntnissen der Forschern umgehen, insbesondere mit Literatur. Für genauere Ausführungen zu den Unterschieden sei hier ebenfalls auf [40] verwiesen. Für den Rahmen dieser Arbeit ist es jedoch hinreichend festzustellen, dass die Methodik von Strauss & Corbin die von Strauss & Glaser expliziert und somit handfester und insbesondere für Novizen anwendbarer anmuten lässt [24], weswegen auch in dieser Arbeit nur die Variante von Anselm Strauss und Juliet Corbin von 1990 dargelegt wird. Nach dem oben genannten ideologischen Bruch zwischen Glaser und Strauss dauerte es weitere 23 Jahre bevor Anselm Strauss zusammen mit Juliet Corbin eine Darstellung der Ideen der Grounded Theory Methodology publizierte, die auch als Einführung für Anfänger konzipiert war: „Basics of Qualitative Research: Grounded Theory Procedures and Techniques“ [38].

Im folgenden wird der Prozess der Datenanalyse (der sogenannte Kodierungsprozess) dargelegt, der sich in drei *nicht sequentiell* zu durchlaufende Phasen – oder auch Arbeitsmodi genannt – unterteilt.

3.2.1 Offenes Kodieren

Beim offenen Kodieren wird das der Studie zu Grunde liegende Material zunächst auf besonders interessant erscheinende Stellen hin untersucht. Solche Stellen nennen Strauss & Corbin *Phänomen*:

> „Science could not exist without concepts. Why are they so essential? Because by the very act of naming phenomena, we fix continuing attention on them. Once our attention is fixed, we can begin to examine and ask questions about those phenomena.“ ([38] S. 62)

Im Anschluss daran werden gleichartige Phänomene benannt. Einen solchen Namen nennen die Autoren *Konzept*.

> „Concepts [...] [are] Conceptual labels placed on discrete happenings, events, and other instances of phenomena.“ ([38] S. 61)

Offenes Kodieren ist insbesondere dazu da um durch aufmerksames Inspezieren der Daten Phänomene zu identifizieren und zu benennen. Im weiteren Verlauf werden diese dann auf Gemeinsamkeiten und Unterschiede untersucht:

> „Open Coding is the part of analysis that pertains specifically to the naming and categorizing of phenomena through close examination of the data [...] During Open Coding the data are broken down into discrete parts, closely examined, compared for similarities and differences [...].“ ([38] S. 62)

Konzepte sind die Grundbausteine der GTM. Es ist oft der Fall, dass sich Konzepte kategorisieren lassen. Dabei meint kategorisieren das Zusammenfassen von ähnlichen Konzepten:

> „Category [...] [is] A classification of concepts. This classification is discovered when concepts are compared one against another and appear to pertain to a similar phenomenon.“ [...]
>
> The process of grouping concepts that seem to pertain to the same phenomena is called categorizing.“ ([38] S. 61-65)

Im weiteren Verlauf werden die Konzepte bzw. Kategorien von Konzepten daraufhin untersucht was sie unterscheidet. Dabei meint eine Unterscheidung hier lediglich geringe Unterschiede, also Unterschiede die nicht so groß sind als dass dieses Phänomen einem anderen Konzept hätte zugewiesen werden müssen, als es wurde. Im Zuge dieser Unterscheidung stößt ein Forscher dann auf Eigenschaften, an Hand derer er ein Konzept bzw. eine Kategorie von Konzepten unterteilen kann. Da sich solche Eigenschaften im Allgemeinen entlang eines Kontinuums, einer Dimension, einordnen lassen, wird dieser Schritt *Dimensionalisierung* genannt:

> „[...] properties are attributes or characteristics of a phenomenon (category). [...] Each time an instance of a category occurs in the data, it is possible to locate it somewhere along the dimensional continua“ ([38] S. 70)

Ein Konzept bzw. eine Kategorie von Konzepten lässt sich also beispielsweise wie in Abbildung 1 darstellen.

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

Abbildung 1: Eigenschaften und Dimensionen der Kategorie „Beobachten“. Während der Phase des Offenen Kodierens werden die Konzepte innerhalb ihrer Kategorien auf Gemeinsamkeiten und Unterschiede untersucht. Da sich diese Eigenschaften im Allgemeinen entlang eines Kontinuums – einer Dimension – einordnen lassen, wird dieser Schritt Dimensionalisierung genannt. [34]

3.2.2 Axiales Kodieren

Beim axialen Kodieren wird nun ein Schritt der über das Kategorisieren des offenen Kodierens hinaus geht gemacht, nämlich das Zusammensetzen der erkannten Konzepte auf neue Art. Bei diesem Reorganisieren der bereits entdeckten Konzepte geht es insbesondere darum die Beziehungen in denen die Konzepte zueinander stehen zu entdecken. Mit diesen Beziehungen ist jedoch noch nicht das Erkennen von Beziehungen zwischen Hauptkategorien gemeint, sondern Beziehungen zwischen Kategorien und deren Subkategorien, um die weitere Beschreibung von dem was eventuell einmal eine Hauptkategorie werden wird voranzutreiben:

> „Open Coding [...] fractures the data and allows one to identify some categories, their properties, and dimensional locations. Axial coding puts those data back together in new ways by making connections between a category and its subcategories [...] we are still concerned with the development of a category, but development beyond properties and dimensions“ ([38] S.97)

Im Arbeitsmodus des Axialen Kodierens geht es im Wesentlichen darum herauszufinden, was die Ursache für das Auftreten eines Konzepts ist, in welchem Kontext die Konzepte eingebettet sind, was die Handlungsstrategien bei Auftreten eines gewissen Konzepts sind und was die Konsequenzen des Auftretens eines Konzepts sein können:

> „However, in axial coding the use of these procedures is more focused, and geared toward discovering and relating categories in terms

> of the paradigm model. That is, we develop each category (phenomenon) in terms of the causal conditions that give rise to it, the specific dimensional location of this phenomenon in terms of its properties, the context, the action/interactional strategies used to handle, manage, respond to this phenomenon in light of that context, and the consequences of any action/interaction that is taken.“ ([38] S. 114-115)

Um sich über die soeben genannten Beziehungen zwischen den Konzepten besser bewusst werden zu können, beschreiben die Autoren ein Handlungsmodell, das paradigmatische Handlungsmodell.

3.2.2.1 Paradigmatisches Handlungsmodell

Um sich Lücken in den bisher erkannten Konzepten und den unterschiedlichen Beziehungen dieser Konzepte untereinander leichter bewusst zu werden, schlagen Strauss & Corbin das paradigmatische Handlungsmodell vor. Da die in dem Modell anzuordnenden Konzepte in einer graphischen Darstellung im Wesentlichen auf einer Achse liegen ist auch der Name der gesamten Phase des Axialen Kodierens abzuleiten.

Das paradigmatische Handlungsmodell enthält ein *Phänomen*. Ein Phänomen ist dabei das zentrale Ereignis oder eine Idee um das bzw. um die sich die anderen Konzepte anordnen:

> „[The phenomenon] is the central idea, event, happening, about which a set of actions/interactions is directed at managing or handling, or to which the set is related.“ ([38] S. 100)

Zudem kennt das Modell *ursächliche Bedingungen*. Diese sind solche die dazu führen, dass das Konzept auftritt.

Ferner kennt das paradigmatische Handlungsmodell einen *Kontext*. Der Kontext ist dabei ein spezifisches Konstrukt aus Eigenschaften, dass das Phänomen betreffen:

> „A context represents the specific set of properties that pertain to a phenomenon; “ ([38] S. 101)

Außer den vorherig genannten verfügt das Modell noch über *intervenierende Bedingungen*. Intervenierende Bedingungen sind dabei generelle Konditionen die auf die Handlungsstrategien einwirken:

> „Intervening conditions are the broad and general conditions bearing upon action/interactional strategies. These conditions include: time, space, culture, economic status, technological status, career, history, and individual biography“ ([38] S. 103)

Zudem kennt das Modell noch *Handlungs- und interaktionale Strategien*. Diese sind zielgerichtete und absichtliche Handlungen die die Reaktion auf ein Phänomen sind.

Zuletzt verfügt das paradigmatische Handlungsmodell noch über *Konsequenzen*. Zu diesen heben Strauss & Corbin hervor, dass die Konsequenzen eines Phänomens die ursächlichen Bedingungen für ein anderes Phänomen darstellen können:

> „The consequences of one set of actions may become part of the conditions [...] affecting the next set of action/interactions occuring in a sequence [...]“ ([38] S. 106)

Das paradigmatische Handlungsmodell lässt sich wie in Abbildung 2 graphisch darstellen:

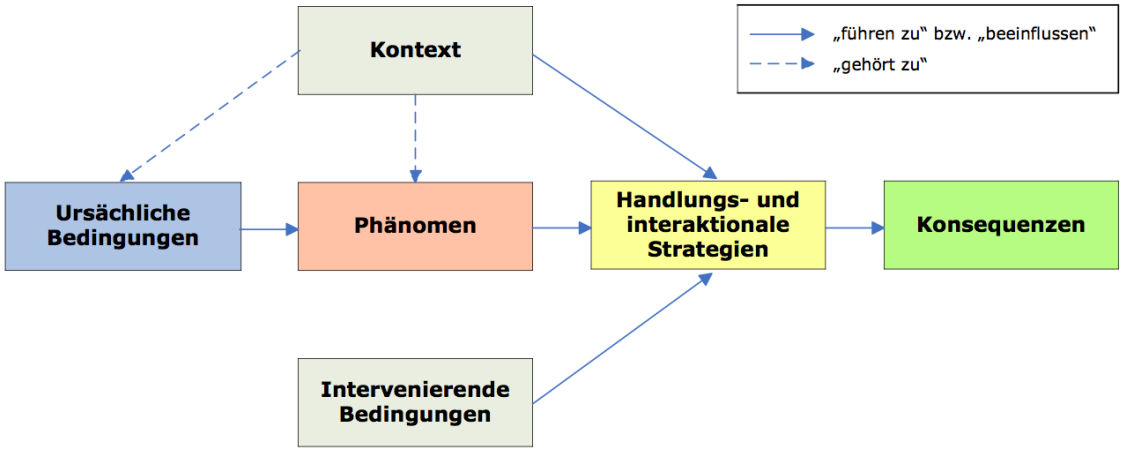

Abbildung 2: Paradigmatisches Handlungsmodell. Während der Phase des Axialen Kodierens wird versucht Lücken in den bisher erkannten Konzepten zu entdecken und sich den unterschiedlichen Beziehungen dieser Konzepte untereinander bewusst zu werden. Um diesen Vorgang zu unterstützen geben Strauss & Corbin das paradigmatische Handlungsmodell an. [34]

3.2.2.2 Techniken zur Erhöhung der theoretischen Sensibilität

Die theoretische Sensibilität ist eine der Voraussetzungen für erfolgreiche qualitative Forschungsarbeit. Ohne sie würden relevante Unterschiede unentdeckt bleiben. Die von Strauss & Corbin genannten Techniken zur Erhöhung der theoretischen Sensibilität werden an dieser Stelle genannt, da sie einen weiteren

Einblick in die Arbeitsweise eines GT-Forschers geben.
Theoretische Sensibilität ist eine persönliche Qualität eines Forschers.

> „Theoretical sensitivity refers to a personal quality of the researcher. It indicates an awareness of the subtleties of meaning of the data“ ([38] S. 41)

Doch woher kann ein Forscher diese erlangen? Als Quellen von theoretischer Sensibilität geben Strauss & Corbin beispielsweise Literatur an (ein erheblicher Unterschied gegenüber dem Ansatz von Strauss & Glaser 1967 - Kapitel 3.2). Eine weitere Quelle für theoretische Sensibilität kann professionelle Erfahrung im Feld der Studie sein:

> „Professional experierence is another source of sensitivity, if a researcher is fortunate enough to have had this experience. Throughout years of practice in a field, one acquires an understanding of how things work in that field, and why, and what will happen there under certain conditions.“ ([38] S. 42)

Als weitere Quelle für Sensibilität geben die Autoren persönliche Erfahrungen an:

> „As an example, the experience of having gone through a divorce can make one sensitive to what it means to experience loss.“ ([38] S. 43)

Die drei bis hierhin genannten Quellen sind allesamt im Zusammenhang mit den Erfahrungen des Forschers verbunden. Als wichtigste Quelle für theoretische Sensibilität geben die Autoren jedoch den analytischen Prozess der GT selbst an. Dazu geben sie dem Forscher mehrere Techniken zur Erhöhung der theoretischen Sensibilität an die Hand die im Folgenden genauer dargestellt werden.

Eine der Techniken zur Erhöhung der theoretischen Sensibilität die die Autoren vorschlagen ist das *Fragestellen*. Durch das Stellen von Fragen sollen die zu dem Zeitpunkt vorhandenen Daten weiter aufgebrochen werden können und somit Unterscheidungsmerkmale offenkundig werden. Als Fragewörter eignen sich dazu besonders Wer? Wann? Wo? Was? Wie? Wie viel? Warum?

Eine weitere Technik zur Erhöhung der theoretischen Sensibilität ist die *Analyse von einzelnen Wörtern*. Es wird dem Forscher vorgeschlagen sich bei der Analyse besonders eingehend mit einzelnen Teilen (von der Satzebene bis herunter zur Wortebene) seiner Beschreibung von Konzepten – seinen Memos (Kapitel 3.2.4) – auseinander zu setzen und auf diese Weise für das vorliegende Konzept zu sensibilisieren. Die Autorenschaft schlägt weiterhin vor, dass der Forscher über den gewählten Begriff so wie deren Assoziationen (auch außerhalb des vorhandenen Kontexts) nachdenkt:

> „Analysis of a word, phrase, or sentence is an especially valuable exercise because it can teach you how to raise questions about possible meanings, whether assumed or intended, by a speaker and those around him or her. [...] we will show you that one does not necessarily need a whole paragraph or a list of questions in order to „open up“ the data. This can be done with a sentence, a phrase, or sometimes even with a single word“ ([38] S. 81)

Eine weitere Kategorie von Techniken die der Erhöhung der theoretischen Sensibilität des GT-Forschers erhöhen sollen ist das Ziehen von Vergleichen. Im Rahmen dieser Arbeit soll lediglich eine der drei vorgeschlagenen Techniken vorgestellt werden, nämlich die sogenannte *Flip-Flop-Technik* (im Deutschen Kontrastierung genannt). Bei dieser werden die beiden Gegenextreme des gleichen Konzepts herangezogen (also die beiden Pole einer Dimension eines Konzepts) und der Frage nachgegangen was diese beiden Repräsentanten voneinander unterscheiden:

> „Nothing seems to strike you. When this happens, you might break through your mental paralysis by using the flip-flop technique. That is, by turning the concept of domination upside down, and imaging the very opposite [. . .].“ ([38] S. 84)

Als letzte Technik zur Erhöhung der theoretischen Sensibilität sei das sogenannte *Schwenken der roten Fahne* erwähnt. Bei dieser Technik handelt es sich im Wesentlichen darum, dass wenn ein GT-Forscher Wörter wie nie, immer oder unmöglich in seinen Notizen auffindet, er wahrscheinlich an dieser Stelle noch einmal nachhaken muss - ganz nach der Redensart sag niemals nie:

> „To emphasize our essential point: You should become very sensitive to certain words and phrases. Ones such as: „Never,“ „Always,“ „It couldn't possibly be that way,“ „“Everyone knows that's the way it is done,“ „There is no need for discussion.“ Every time you hear such words or phrases, you should wave a red flag — an imaginary flag, of course. These words and phrases should be taken as signals to take a closer look.“ ([38] S. 92)

3.2.3 Selektives Kodieren

Die Phasen des offenen Kodierens und des axialen Kodierens dienen vornehmlich der Konzeptentwicklung. Ziel des selektiven Kodierens ist es die Konzepte und deren Relationen zueinander zu einem Gesamtbild zusammenzufügen, zu einer Theorie im Sinne der GTM.

> „Selective Coding [is] The process of selecting the core category, systematically relating it to other categories, validating those relationships, and filling in categories that need further refinement and development.“ ([38] S. 166)

Eine der Hauptaktivitäten des selektiven Kodierens ist das *Erzählen einer Geschichte* und das *Darlegen des roten Fadens der Geschichte*. Dazu bedarf es erst einmal der Identifikation der Hauptkategorie(n). Im Anschluss daran werden die übrigen Konzepte um diese angeordnet:

> „[The] core category [is] The central phenomenon around which all the other categories are integrated.“ ([38] S. 116)

Trotz der abermals getrennten Benennung der verschiedenen Arbeitsmodi der GTM ist auch hier die Linie zwischen axialem Kodieren und selektivem Kodieren unscharf. Dazu sagen die Autoren:

> „Integration is not much different than axial coding. It is just done at a higher more abstract level of analysis.“ ([38] S. 117)

3.2.4 Sonstige Praktiken der GTM

In diesem Abschnitt sollen weitere mit der GTM fest verbundene Praktiken beschrieben werden, die den gesamten Forschungsprozess ebenfalls prägen. Um den Rahmen dieser Arbeit nicht zu sprengen, werden die folgenden Praktiken in sehr verkürzter Version dargestellt:

- **Ständiges Vergleichen** ist eine solche zentrale Praktik, die den gesamten Forschungsprozess durchzieht. Sie brachte der GTM sogar den Beinamen „Analysemethode der ständigen Vergleiche“ ein ([38], S.62),[39]. Mit ständigem Vergleichen ist gemeint, dass die durch den Kodierungsprozess erzeugten Konzepte bzw. Kategorien im ständigen Dialog mit den empirischen Daten sind. Auf diese Weise sollen die Unterschiede so wie die Ähnlichkeiten in den Daten entdeckt werden können.

- **Memos** sind eine weitere Praktik der GTM. Memos sind Aufzeichnungen die der Erzeugung einer Theorie dienen. Da der gesamte Forschungsprozess der Erzeugung einer Theorie dient, können Memos als ein extrem vielseitiges Mittel zur Aufzeichnung genutzt werden.

  > „Memos [...] [are] Written records of analysis related to the formulationof theory. [...] Memos represent the written forms of our abstract thinking about data“ ([38] S.197)

– **Theoretisches Sampling** ist eine Praktik zur Datenerhebung. Entgegen dem sequentiellen Vorgehen quantitativer Forschung ist bei der GTM auch das Sammeln von Daten iterativ möglich. Dies heißt, dass der Forscher die Freiheit hat beispielsweise zur weiteren Verfeinerung seiner bisherigen Theorie neues Datenmaterial in den Prozess miteinzubeziehen. Dabei kann er das neue Material ebenfalls auf Grundlager seiner bisherigen Theorie auswählen, also beispielsweise Datenmaterial einer spezifischen Gruppierung neu in den Prozess bringen.

3.2.5 Mögliche Schwierigkeiten

Bei der Forschung nach der GTM kann es zu Problemen kommen die durch den Forschungsstil bedingt sind. An dieser Stelle sollen jedoch nur zwei erwähnt werden, da diese auch meine Erfahrung während der Erstellung dieser Arbeit widerspiegeln.

Ein Forscher kann dazu verleitet sein, sein theoretisches Vorwissen in den Daten erkennen zu wollen und dadurch die theoretische Sensibilität (Kapitel 3.2.2.2 einbüßen, die nötig wäre um objektive und somit „gegroundete“ Daten zur Theoriebildung erhalten zu können. Diese mögliche Gefahr nennt man *Forcing*, da der Forscher versucht ein Resultat zu erzwingen und nicht aus den vorhandenen Daten zu entdecken.

Ein weiteres Problem der GTM kann sein, dass ein Forscher in der Fülle der Daten untergeht. Dies ist insbesondere zu Beginn einer Studie relevant, da gerade zu diesem Zeitpunkt noch nicht klar sein kann, welche Beobachtungen wichtig sein könnten und welche nicht. Ferner fehlt zu diesem Zeitpunkt ebenfalls oft der spezifische Blick auf die Daten, denn durch einen sich verändernden Blick auf die Daten verändern sich die Daten natürlich in ihren Phänomenen. Das überwältigt sein von den Daten bzw. der möglichen Blickwinkel auf die Daten wird auch *overwhelmed by the data* genannt [33].

3.2.6 4 Erweiterungen der GTM durch Prechelt et al.

Für das Forschungsvorhaben der AGSE wurde die GTM wie sie hier bisher dargestellt worden ist um vier weitere Praktiken erweitert, um den spezifischen Problemen bei Anwendung der GTM (Kapitel 3.2.5) zu begegnen [33].
Die erste Praktik ist das *Einschränken des Blickwinkels* auf die Daten. Um dieses tun zu können bedarf es natürlich einer Forschungsfrage die einerseits sehr offen formuliert ist, andererseits aber natürlich auch einschränken muss um sinnvoll zu sein. Ferner sollte sie Aufschluss darüber geben was man untersuchen möchte, so wie eine grobe Aussage darüber treffen was man darüber herausfinden möchte. Eine Einschränkung die hier von der AGSE gemacht wurde ist, dass ausschließ-

lich beobachtbare Phänomene (im behavioristischen Sinne) in die Konzeptualisierung aufgenommen werden sollten. Die Forschungsfrage die sich der AGSE beschlossen hat anzunehmen kann grob wie folgt wiedergegeben werden:

> „What mechanisms underlie knowledge transfer during pair programming and which of these work well or not so well?“ [45]

Die zweite Praktik ist das Definieren der *Syntax für Konzeptnamen*. Die GTM selber macht keinerlei Aussagen dazu wie Konzepte zu benennen sind. Durch diese Freiheit aber kann es passieren, dass ähnliche Konzepte vollkommen unterschiedliche Namen haben. Dies erschwert natürlich fortwährend den Forschungsprozess. Um dieser Schwierigkeit zu begegnen wurde das Namensschema aus Abbildung 3 eingeführt.

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

Abbildung 3: Namensschema für die Benennung von Konzepten der GTM. Die GTM selber macht keine Aussage dazu wie Konzepte zu benennen sind. Durch diese Freiheit kann es passieren, dass ähnliche Konzepte vollkommen unterschiedliche Namen tragen. Um dieser Schwierigkeit zu begegnen wurde das Namenschema eingeführt. P1 bzw. P2 stehen für die Entwickler. P denotiert das Paar als Ganzes. Mit der description wird eine Handlung beschrieben, die sich aus einem Verbbestandteil und einem Objektbestandteil zusammensetzt. Der Zusatz criterion kann dazu verwendet werden um die description noch genauer zu beschreiben. [44]

Die dritte Praktik hat vor allem den Hintergrund die Unterschiede in der GTM-Benennung und dem verwendeten Softwaretool zu überbrücken (mehr dazu in Kapitel 4). Weiterhin dient das *Metamodell der Analysemethoden und -objekte* dazu den Prozess der GTM an manchen Stellen klarer darzustellen, insbesondere in Bezug auf den Unterschied zwischen der „Phänomenwelt“ (reale Welt) und der konzeptuellen Welt. Abbildung 4 zeigt das im UML-Format (Unified Modeling Language [32]) dargestellte Metamodell der Analysemethoden.

Als letzte Praktik wurde das *Kodieren im Paar* eingeführt was logischerweise direkten Einfluss auf die Intersubjektivität der vergebenen Konzepte und somit der gesamten Studie hat. Ferner bezieht diese Praktik eben den Forschungsgegenstand in die Arbeit der Forscher mit ein was eventuell zu einer Erhöhung der theoretischen Sensibilität (Kapitel 3.2.2.2) führen könnte. An dieser Stelle sei

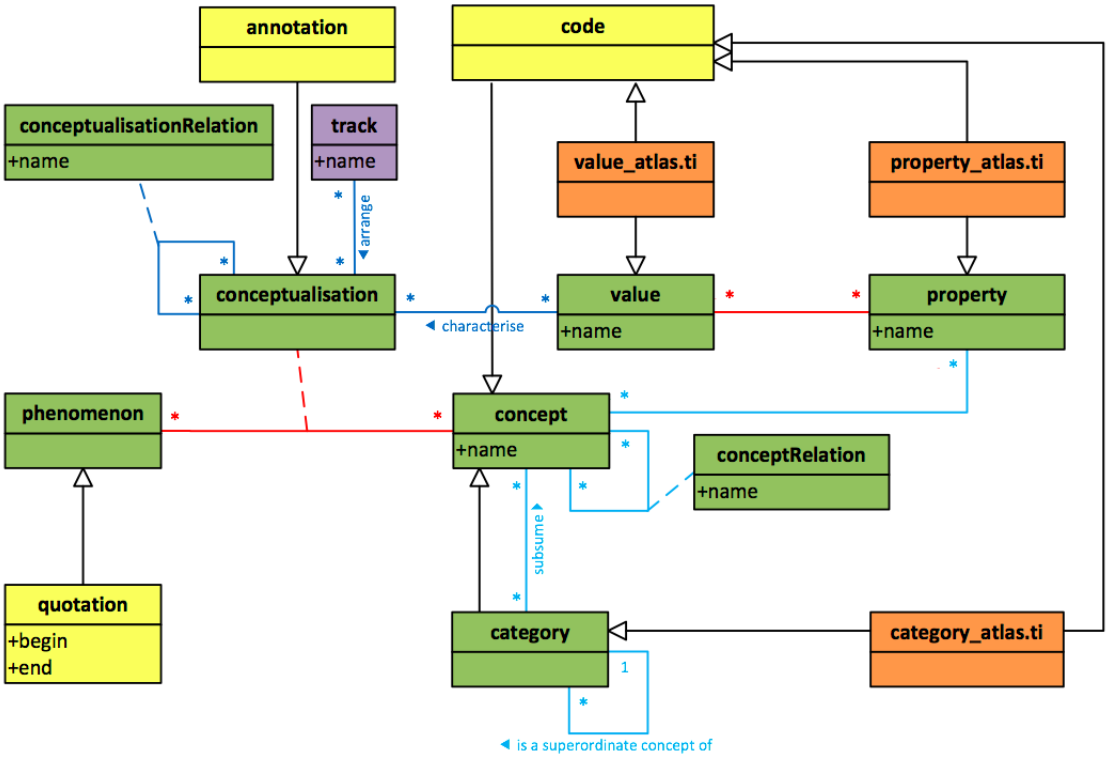

Abbildung 4: Metamodell der Analysemethoden und -objekte. Die grundlegenden Bestandteile der GTM sind mit Grün markiert. Mit Gelb sind die Realisierungen dieser Bestandteile in dem Softwarewerkzeug ATLAS.ti (Kapitel 4) dargestellt. Orange eingefärbt sind Klassen, die für die Analyse nach GTM mit ATLAS.ti benötigt werden, für die es aber keine ursprüngliche Entsprechung in der GTM gibt. Sie müssen also an Hand von anderen Klassen realisiert werden – beispielsweise 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 in ATLAS.ti. [34]

angemerkt, dass im Rahmen der vorliegenden Arbeit diese Praktik nicht zum Einsatz gekommen ist, da es sich ja um den Leistungsnachweis eigenständiger Arbeit handeln soll.

4 Daten und Datenerhebung

Da nun die Forschungsmethode erklärt worden ist wird in diesem Kapitel das Datenmaterial so wie die Datenerhebung dargelegt auf dessen Grundlage die Ergebnisse, die im weiteren Verlauf dieser Arbeit vorgestellt werden, gewonnen wurden.

Das Datenmaterial das meiner Arbeit zu Grunde liegt sind Videomitschnitte von Paarprogrammierungssitzungen. Es handelt sich um reale Mitschnitte von professionellen Softwareentwicklern aus mittelständischen deutschen Softwareunternehmen, die an realen Aufgaben arbeiten – also solchen die sie ohnehin hätten bearbeiten müssen [34]. Zu sehen ist auf diesen Videos der Bildschirm der beiden Partner der mittels Camtasia Studio [2] mitgeschnitten worden ist. Darüber hinaus sind ebenfalls die beiden Partner selbst zu sehen, da diese mit einer Webcam (Auflösung: 176x144 oder 320x240 Bildpunkte) aufgenommen worden sind. Dabei ist der gewählte Blickwinkel so, dass man (meistens) die Oberkörper der beiden Partner sehen kann, ebenso wie Tastatureingaben und Mausbewegungen. Außerdem wurden die Paarprogrammierer während der Sitzungen mit Steckmikrofonen ausgestattet, sodass also ebenfalls eine Audiospur vorhanden ist. Abbildung 5 zeigt einen typischen Frame aus dem Videomaterial.

Zur Datenerhebung wurde das Softwarewerkzeug ATLAS.ti [1] genutzt. Bei ATLAS.ti handelt es sich um eine Software die speziell für die qualitative Datenanalyse entwickelt worden ist. Sie bietet verschiedene Toolbars und Fenster die die Analyse qualitativer Daten sehr unterstützen und zugleich digitalisieren, was bei dem Umfang an Material von größter Wichtigkeit ist. Abbildung 6 zeigt einen typischen Screenshot von ATLAS.ti. Zu sehen ist im linken Teil des Bildschirms ein Fenster, in dem der aktuell bearbeitete Videomitschnitt abläuft. Parallel dazu läuft im mittleren Teil des Bildschirms der Zeitstrahl inklusive Audiospur mit. Im rechten Teil des Bildschirms kann man die vergebenen Annotationen des Forschers an dem Datenmaterial sehen. Es handelt sich hier um einen Screenshot meines Projekts der Sitzung BA1 - (6:20-10:46).

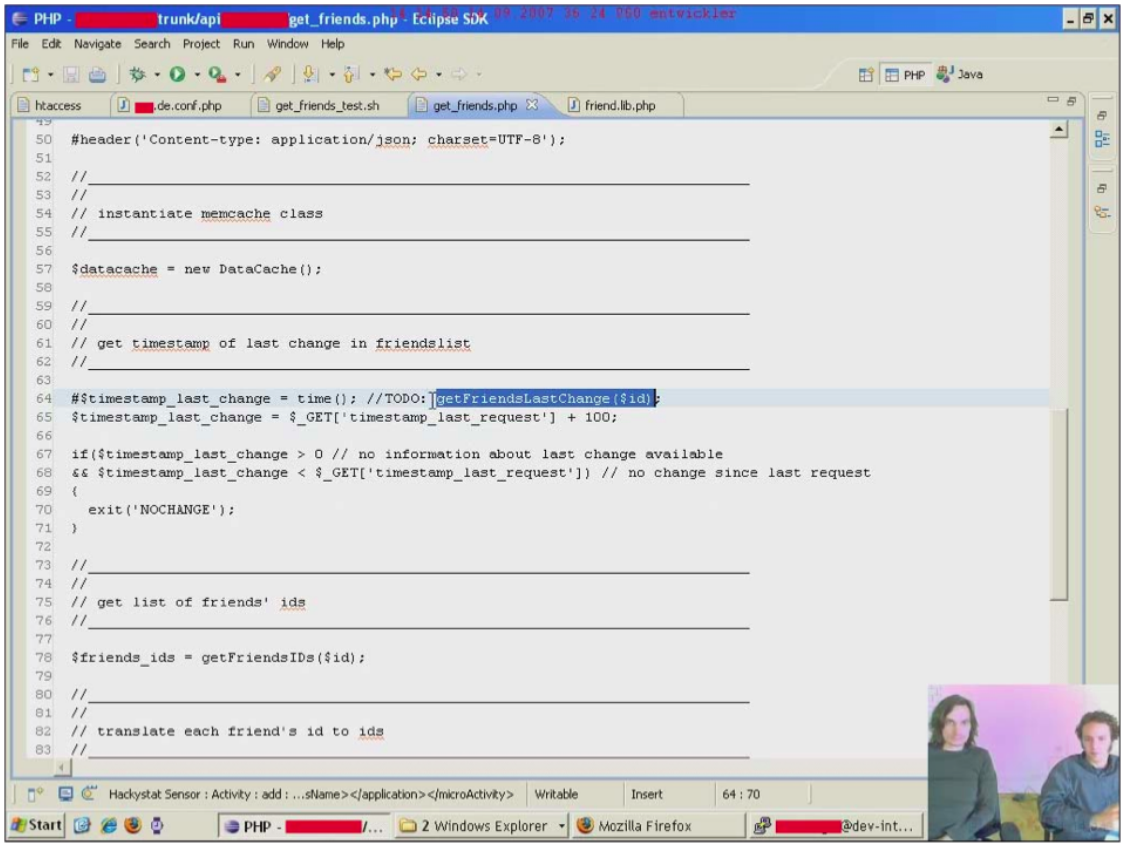

Abbildung 5: Screenshot der Paarprogrammierungssitzung BA1 – (14:14). Der linke der beiden Entwickler wird im Folgenden mit B1 denotiert. Entsprechend der rechte mit B2. [34]

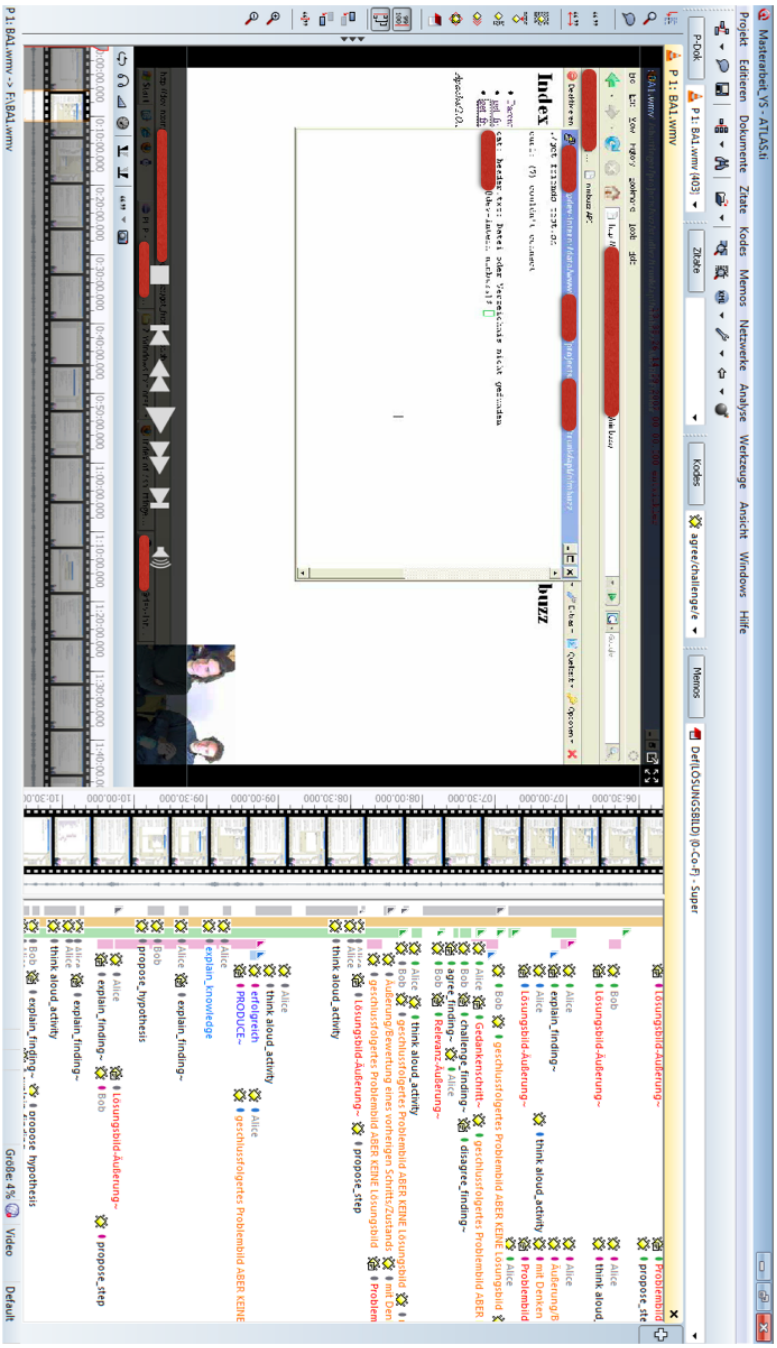

Abbildung 6: ATLAS.ti Screenshot mit Paarprogrammierungsmitschnitt als Video. Es handelt sich um einen Screenshot meines Projekts für die Sitzung BA1 – (6:20-10:46). Die Konzepte in nicht-grauer Schriftfarbe im rechten Teil des Bildschirms sind meine Annotationen.

5 Qualitative Forschung der AGSE

Wie in der Einleitung dieser Arbeit erwähnt betreibt die Arbeitsgruppe Software Engineering qualitative Forschung auf dem Gebiet der Paarprogrammierung. Die bisherigen Ergebnisse sind vielfältig, an dieser Stelle seien aber nur die für den weiteren Verlauf dieser Arbeit unmittelbar relevanten dargelegt. Diese sind zum einen die Basisschicht der Paarprogrammierung und zum anderen die bisherigen Erkenntnisse über den Wissenstransfer in der Paarprogrammierung.

5.1 Basisschicht

Anmerkung: Die nachfolgende Passage benutzt als einzige Quelle die Promotion von Stephan Salinger [34] – daher wird in diesem Abschnitt darauf verzichtet die Quelle von wörtlichen Zitaten erneut anzugeben.

Mit der Basisschicht (BS) sollte eine „Sprache über die PP“ geschaffen werden, die in möglichst vielen nachfolgenden qualitativen Untersuchungen der PP verwendet werden kann. Die Basisschicht ist also ein Rahmenwerk für die qualitative Paarprogrammierungsforschung, damit weiterführende Studien (so wie die hier vorliegende Arbeit) nicht bei null anfangen müssen, sondern im bildlichen Sinne auf dieses Fundament aufbauen können. Abbildung 7 zeigt das Schichtenmodell der Basisschicht. Die BS zielt darauf ab „die grundlegenden Vorgänge, aus denen der Paarprogrammierungsprozess zusammengesetzt ist“ zu formulieren. Wichtig ist dabei, dass grundsätzlich gilt dass die Äußerungen – die sogenannte Diskurswelt – höher zu bewerten ist als die wahren Tätigkeiten die dahinter stecken – der sogenannten Tätigkeitswelt. Ferner soll die primäre Intention des Sprechers kodiert werden und nicht etwa eine solche, die man als externer Beobachter auf die Sitzung vermuten könnte. Die BS ist durch die GTM in der in dieser Arbeit beschriebenen Form erlangt worden (Kapitel 3.2).
Die BS ist auf oberster Ebene in zwei Klassen unterteilt. Zum einen in Human-Human Interaction (HHI) Konzepte und zum anderen Human-Computer Interaction (HCI) Konzepte bzw. Human-Environment Interaction (HEI) Konzepte. Die Klasse der HHI-Konzepte umfasst Konzepte, mit denen Interaktionen zwischen den beiden Paarmitgliedern, also aufeinander bezogene Handlungen, beschrieben werden können. Hingegen subsummiert die Klasse der HCI/HEI Konzepte, deren Handlungen die Interaktionen einer Person mit ihrer sonstigen Umgebung adressieren, also beispielsweise Maus- und Tastatureingaben, so wie die Anfertigung handschriftliche Zeichnungen.
Die Klasse der HHI ist die deutlich umfangreichere der beiden und lässt sich untergliedern in:

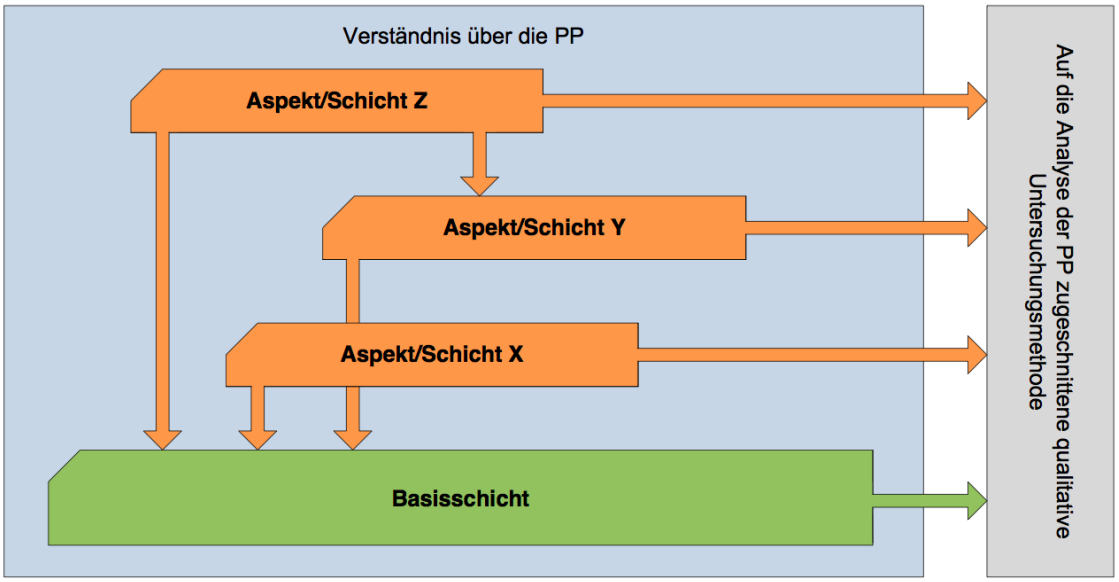

Abbildung 7: Schichtenmodell der Basisschicht. Die BS ist einerseits eine auf die Analyse von Paarprogrammierung zugeschnittene Untersuchungsmethode (in grau dargestellt), andererseits ein System von Konzepten an Hand derer sich die grundlegenden Aktivitäten der Entwickler während einer PP-Sitzung beschreiben lassen (in grün dargestellt). Wie zu erkennen ist, ist das Ziel der BS – wie der Name nahelegt – das Fundament für weitere spezialisierte Untersuchungen die das Ziel haben das Verständnis über die PP zu erhöhen darzustellen. Die vorliegende Arbeit stellt eine solche spezialisierte Untersuchung dar. [34]

- **Produktorientierte Konzepte**
  Produktorientierte Konzepte werden für Äußerungen verwendet, die die Gestalt des zu entwickelnden Programms betreffen. Dabei kann es beispielsweise um die Inhalte, die Struktur oder die Anordnung von Programmartefakten gehen.

  > „Wir können das auch da oben lassen, erst mal. (..)“

  ist ein Beispiel für eine Äußerung, die die Gestalt des zu entwickelnden Programms betrifft, da der Sprecher in dieser Situation vorschlägt eine zuvor besprochene Änderung erst einmal nicht nun anzugehen.

- **Prozessorientierte Konzepte**
  Prozessorientierte Konzepte referenzieren Äußerungen, die sich auf die Gestaltung des Arbeitsprozesses beziehen und beziehen sowohl Vorschläge wie auch Entgegnungen auf solche mit ein.

> „Die Frage ist, ob ich mich da jetzt rantrauen will, ob wir uns da jetzt rantrauen wollen?“

ist ein Beispiel für eine Äußerung die die Gestaltung des Arbeitsprozesses betrifft, da sie einen nächsten Schritt zur Diskussion stellt.

– **Universelle Konzepte**
Universelle Konzepte adressieren Äußerungen, mit denen Wissen angefordert, transferiert und beurteilt wird sowie Wissenslücken offenbart oder Vermutungen bzw. Hypothesen aufgestellt und bewertet werden. Sie werden universell genannt, da sie sowohl im Produkt- wie auch im Prozesskontext verwendet werden können.

> „Was passiert, wenn der nicht gesetzt ist?“

ist ein Beispiel für eine Äußerung die Wissen anfordert (da „der“ hier einen Übergabeparameter referenziert).

– **Fassadenkonzepte**
Fassadenkonzepte bilden eine Teilmenge der universellen Konzepte und liefern einen vereinfachten, auf einen bestimmten Aspekt fokussierten Blick auf in der Regel komplexere Phänomene. Die Benennung dieser Klasse ist durch das Entwurfsmuster „Facade“ motiviert [23].

> „Ich muss mich jedes mal wieder umgewöhnen, weil ich keine Apfeltaste mehr hab.“

ist ein Beispiel für eine Teilmenge der universellen Konzepte, nämlich eine Verbalisierung der Gedanken eines Entwicklers – eine sogenannte think aloud_activity.

– **sonstige Konzepte**
Sonstige Konzepte sind Kennzeichnungen für Äußerungen, die unverständlich sind oder nicht mit der Aufgabenbearbeitung zusammenhängen.

> „Ja das war mein Erlebnis mit Erdmännchen. Und den hätte ich mit nach Hause genommen, sofort“

ist ein Beispiel für eine Äußerung die offensichtlich nichts mit der Aufgabenbearbeitung zu tun hat.

Abbildung 8 fasst das bis hierhin gesagte auf graphische Art und Weise zusammen. Sie zeigt also die Basisschicht in ihrer bis hierhin ausformulierten Granularität.

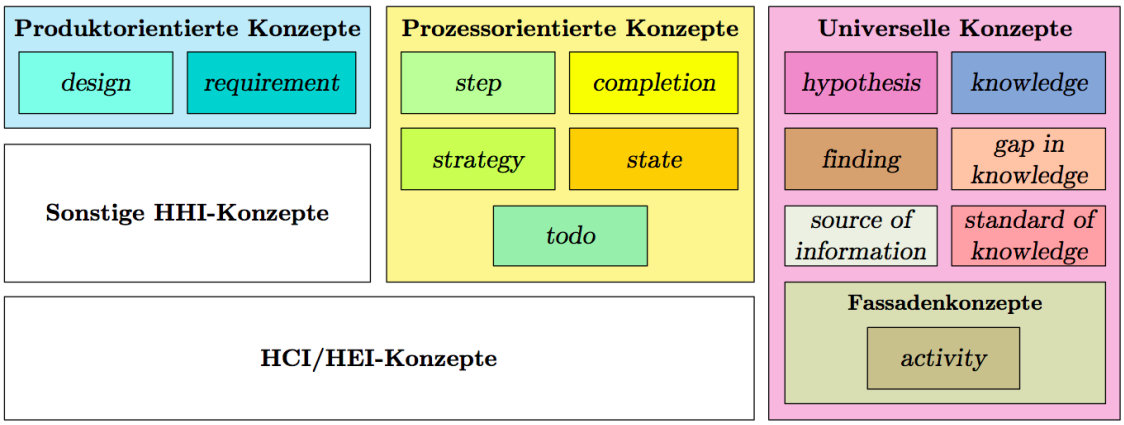

Abbildung 8: Vereinfachte Darstellung der Basisschicht [44] – Die Objekte auf die sich produktorientierte Konzepte beziehen sind design und requirement. Objekte der prozessorientierten Konzepte sind step, completion, strategy, state und todo. Die universellen Konzepte haben als Objekte hypothesis, knowledge, finding, gap in knowledge, source of information und standard of knowledge. Die Fassadenkonzepte verfügen als Objekt über activity.

Zur Struktur der Konzepte der BS ist zu sagen, dass sie jeweils aus einer Verb-Objekt Verbindung besteht (Kapitel 3.2.6). Genauer gesagt wurden für die BS 13 Verben und 16 Objekte entdeckt. Dies würde wiederum vermuten lassen, dass es dementsprechend auch 13*16= 208 Konzepte gibt – dem ist aber nicht so da ja nach GT Methodik nur Phänomene konzeptualisiert werden, die tatsächlich in den Daten zu finden waren, nicht aber solche die sinnvoll erscheinen, dass sie zu finden sein sollten (Kapitel 3.2.6). Abbildung 9 zeigt die identifizierten 16 Objekte der Basisschicht. Abbildung 10 zeigt die identifizierten 13 Verben der Basisschicht.

Nun wo die Struktur der Konzepte so wie deren Bestandteile dargelegt wurden, soll die BS in ihrer Gesamtheit aufgezeigt werden, da diese im weiteren Verlauf der Arbeit mehrfach referenziert wird. Abbildung 11 zeigt die Basisschicht vollständig.

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.

Abbildung 9: Objektbestandteile der Basisschicht. Es handelt sich um die identifizierten Objekte der HHI-Konzepte. Diese bestehen ausschließlich aus tatsächlich verbalisierten Entitäten. [34]

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.

Abbildung 10: Verbenbestandteile der Basisschicht. Es handelt sich um die tatsächlich verbalisierten Objekte der HHI-Konzepte. [34]

Abbildung 11: Die Basisschicht – das Rahmenwerk qualitativer Paarprogrammierungsforschung. Zu sehen sind die 61 Konzepte der Basisschicht. Farblich zusammengefasst sind die erwähnten Klassen. Innerhalb dieser sind die Konzepte wiederum nach ihrem Objektbestandteil farblich zusammengefasst. [34]

5.2 Wissenstransfer

Wie soeben beschrieben bildet die Basisschicht das Fundament für spezifischere Studien, beispielsweise über den Wissenstransfer in der PP.

Qualitative Studien der AGSE zum Wissenstransfer kamen zu dem Ergebnis, dass es zwei dominante Arbeitsmodi während PP-Sitzungen zu geben scheint. Diese sind „Knowledge Transfer“ (Wissenstransfer) und „Decision Making“ (das Treffen von Entscheidungen). Wissenstransfer ist dabei durch drei spezifische Wissenskonzepte bzw. Wissensphänomene dingfest zu machen:
Zum einen dem „need for knowledge“ – also dem Bedarf nach Wissen. Da wie in Kapitel 3.2.6 gesagt jedoch nur beobachtbare Phänomene konzeptualisiert werden sollen, wurde diesem Phänomen jedoch kein eigenes Konzept zugeordnet. Allerdings lässt es sich durch das Ausführen von anderen Aktivitäten (z.B. durch das Artikulieren einer Frage an den Partner) dermaßen einkreisen, dass man intersubjektiv nachvollziehbar feststellen kann, dass es diesen Bedarf gibt.
Der zweite Aspekt von Wissenstransfer ist das TOPIC – also das Thema um das es in einer solchen Phase geht. Zuletzt ist das Konzept TARGET CONTENT – also der Zielinhalt – zu erwähnen. Dabei handelt es sich um das, was ausreicht um den Bedarf nach Wissen zu einem Thema zu befriedigen. Eine mögliche Form von Wissenstransfer kann also sein, dass ein Softwareentwickler seinem Partner etwas erklärt, dass derjenige vorher noch nicht wusste.
Darüber hinaus wurde beobachtet, dass Wissenstransfer 35% der Zeit einer PP-Sitzung ausmacht [45]. Abbildung 12 zeigt die zeitliche Verteilung einer Paarprogrammierungssitzung. In der Folge wurde dieser Arbeitsmodus genauer untersucht.

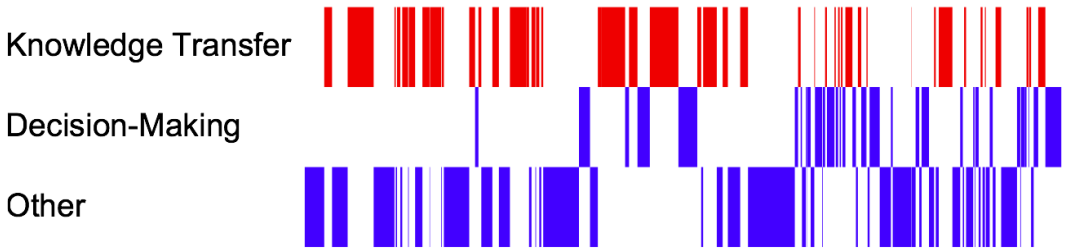

Abbildung 12: Zeitliche Verteilung von Paaraktivitäten einer Paarprogrammierungssitzung. Die Paaraktivitäten wurden in drei Kategorien unterteilt: Wissenstransfer, das Treffen von Entscheidungen und Anderen. Wissenstransfer machte 35% der Zeit der PP-Sitzung aus. [45]

Bei näherer Betrachtung wurde ein Konzept bezüglich der Rollenverteilung in

Phasen des Wissenstransfers beobachtet, PROPELLOR genannt – also der Antreiber. Ein PROPELLOR hat eine Idee zu dem aktuell besprochenen Thema und verfolgt die Klarstellung dieses Themas. Ferner zeichneten sich drei Modi für Phasen in denen Wissenstransfer stattfindet ab, diese wurden PUSH, PULL und PRODUCE genannt [45].
An dieser Stelle erscheint es sinnvoll noch ein anderes Konzept vorzustellen, nämlich das der EPISODE. Eine Episode ist ein zeitlich zusammenhängender Teil einer PP-Sitzung, die ein konstantes TOPIC verfolgen und einen konstanten Wissenstransfermodus haben. Eine PUSH-Episode ist dann vorhanden wenn der PROPELLOR selbst über den TARGET CONTENT verfügt und diesen verbalisiert um den Bedarf nach Wissen seines Partners zu befriedigen. Eine PULL-Episode liegt hingegen vor wenn der PROPELLOR die Person ist, die den Bedarf nach Wissen verspürt, äußert und im Folgenden den TARGET CONTENT von seinem Partner erhält.
Gelegentlich kann es vorkommen, dass keiner der beiden Partner über den TARGET CONTENT verfügt, der den aktuellen Bedarf an Wissen über ein spezifisches Thema befriedigt. Solche Episoden nennen Zieris und Prechelt PRODUCE-Episoden, da in diesen der TARGET CONTENT sozusagen „on the fly“ erzeugt wird [45]. Eben diese Episoden standen im Fokus der vorliegenden Arbeit. Jedoch ist anzumerken, dass in der vorliegenden Arbeit der Episoden Begriff nicht so eng angewandt wird, deswegen ist bisher von „Phasen“ gesprochen worden und nicht von Episoden. Für die Betrachtung der gemeinsamen Wissensproduktion schien mir der Begriff der Episode das Phänomen Wissensproduktion in viele zu kleine Episoden zu unterscheiden, die für meine Betrachtung irrelevant sind, da beispielsweise thematische Veränderungen zur Beendigung einer Episode führen, obwohl sie eventuell bloß als thematische Fokussierung oder Divergenz aufgefasst werden könnten.
Abbildung 13 zeigt eine graphische Zusammenfassung der Wissenstransferkonzepte.

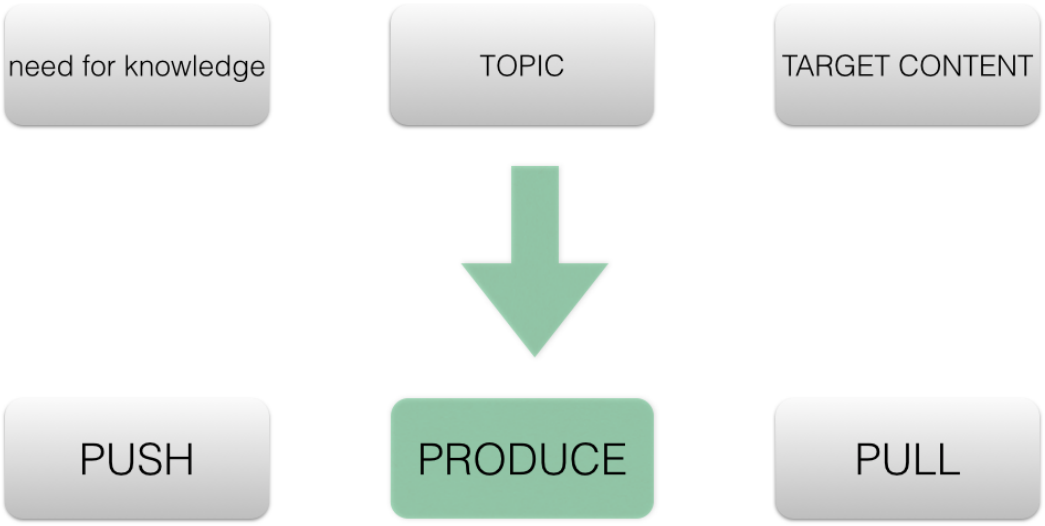

Abbildung 13: Zusammenfassung der Wissenstransferkonzepte. In der oberen Zeile sind die Konzepte in Großbuchstaben dargestellt, die Wissenstransfer charakterisieren. In der unteren Zeile sind die drei Modi in denen Wissenstransferphasen stattfinden können dargestellt: PUSH, PULL und PRODUCE. Da bei PRODUCE-Phasen das Wissen/der TARGET CONTENT „on the fly“ erzeugt wird, stellen sie die interessanten Passagen für die vorliegende Arbeit dar und sind deshalb grün eingefärbt.

6 Ergebnisse

Dieses Kapitel stellt die Ergebnisse meiner Studien über die gemeinsame Wissensproduktion in der Paarprogrammierung dar.

Wie im vorhergegangenen Kapitel erläutert sind die relevanten Phasen des Datenmaterials die PRODUCE-Phasen (Kapitel 5.2). In der Folge habe ich also diese zuerst in den PP-Sitzungsaufzeichnungen, meinem Datenmaterial (Kapitel 4), identifiziert und mit den Basiskonzepten (Kapitel 5.1)versehen, um mich in die Methodik einzuarbeiten und um einen ersten Überblick zu bekommen.

Eine GT-Studie lebt von Beispielen, da die Erkenntnisse ja eben aus den vorliegenden Daten gewonnen werden und auf diese Art und Weise die „Groundung“ der Daten gewährleistet werden kann (Kapitel 3.2).

Aus diesem Grund wird hier auch im folgenden ein reales Szenario aus einer Sitzung dargelegt, an welchem meine Erkenntnisse geeignet aufgezeigt werden können. Selbstverständlich stellt dieses Szenario lediglich einen kleinen Ausschnitt der insgesamt betrachteten Datenmenge dar, dennoch ist er aber geeignet um viele meiner Ergebnisse zu illustrieren und an den Daten zu belegen. Natürlich wird es im weiteren Verlauf dieser Arbeit auch notwendig auf andere beispielhafte Stelle der Daten zu verweisen. Als Ausgangspunkt ist das folgende Beispiel aber trotz seiner Länge sehr gut geeignet. Das folgenden reale Szenario ist gemäß des Kodierungsschemas der AGSE ausgeführt, welches sich im Anhang dieser Arbeit befindet (Kapitel Anhang).

Zur weiteren Illustration und einem besseren Verständnis sei an dieser Stelle eine kurze Situationsbeschreibung angegeben:
Die Sitzung findet in einem Unternehmen statt, das eine eigene Webapplikation im Bereich Social Media entwickelte und betrieb. Das Gesamtziel der Sitzung ist eine bereits bestehende Funktionalität zu erweitern. Das Hauptziel der Erweiterung besteht darin, die übertragene Datenmenge zwischen dem eigenen ereignisgekoppelten Back-End und einem weiteren Webserver einer Partnerfirma zu reduzieren. Die zu erweiternde Funktionalität ist in PHP [6] programmiert und verwendet JSON [5].

Es gibt eine Tastatur und eine Maus als Bedienelemente des Computers. Zu erwähnen ist außerdem, dass zumindest ein Entwickler der beiden üblicherweise an einem Mac arbeitet und sich daher mit der Tastatur des vorliegenden Computers, so wie mit manchen Bedienungsmöglichkeiten nicht (mehr) firm ist. Der Driver wechselt während aller hier dargestellten Abschnitte des Szenarios nicht.

Die PP-Sitzung ist wie auch an der Zeitstelle zu bemerken ist gerade gestartet. Zu Beginn führt das Paar einen Test aus, um die zu bearbeitende Funktionalität testen zu können. Das Ziel dabei ist es die Rückgabewerte des Skripts so zu verändern, dass sie im geöffneten Webbrowser eingesehen werden können.

Dies funktioniert aber nicht auf Anhieb wie erhofft.

BA1 1:33 – 12.30

B1	„cURL[2] kann sich immer noch nicht connecten“
B2	„(∼∼)“
B1	„Tja, wir können das ja hier einfach mal so ausprobieren („ ,) weil das doofe ist wir können es nicht einfach nur im Browser machen („) ich kopiere schnell hier heraus – wir können das nicht einfach nur im Browser machen, weil MimeTypeApplication JSON oder JavaScript ist. („) Wie kann man hier reinpasten?“
B2	„Mit rechts“
B1	„rechte Maustaste?“
B2	„mhm <*Zustimmung*>“
B1	„Habe ich gerade gemacht“
B2	„Ne, mit links kopierst du, mit rechts“
B1	„Oookay („ ,) Mit links kopiere ich? Mit links klicke ich doch einfach nur?“
B2	„Ja aber wenn du was markiert hast und dann links klickst dann wird es kopiert“
B1	„Achso. Also dann bitte („ „) couldn’t connect (,) Gibt es hier auch ctrl+a? („ „)“
B2	„Aber cURL ist schon noch drauf oder (∼∼)“
B1	„Ja („) dev-intern mal gucken was dann kommt (,) couldn’t connect“
B2	„Versuch’ mal ein wget [4] (∼∼)“
B1	„fehlgeschlagen“
B2	„Mach’ mal auf localhost statt dev-intern“
B1	„fehlgeschlagen. Hm, was ist denn da los?“
B2	„Firewall?“
B1	„Auflösen des Rechnernamens localhost“
B2	„Vielleicht nur die (∼) Ne, die hat er ja aufgelöst“

[2] cURL („Client for URLs“) ist ein Kommandozeilen-Werkzeug Datenübertragung welches u.a. die Protokolle FTP und HTTP unterstützt [3]

B1	„Genau (,) (∼) ist okay („ ,) Ich meine wir können es im Browser aufrufen, nur kommt halt dann nichts zurück. Wir könnten allerdings den MimeType herausnehmen jetzt zum Entwickeln („ ,) Ok, wir machen das nicht mit TextPad. Auf keinen Fall (,) Jetzt müssen wir ihm erstmal sagen womit er das öffnen soll.“
B2	„Kannst du das nicht herüberziehen? <*drag&drop*> (∼)“
B1	„Oh Gott. Da muss man jetzt erstmal sagen open with editor (∼) („ „) Gut und hier irgendwo geben wir doch einfach den Header aus und den geben wir jetzt erstmal nicht aus. Wir können mal die Schrift ein bisschen kleiner machen.“
B2	„Du hast keine Parameter mit angegeben in der get_-<**Entität**><*.php*>“
B1	„Stimmt („) Steht der Parameter irgendwo hier? <*kopiert ihn dann aus dem Testskript*>“
B2	„Achso, du hast nicht gespeichert (..) die get_<**Entität**>.php“
B1	„N bisschen ungewohnt alles. Oookay“
B2	„Okay, dann kommt er einfach nicht weiter aber wir können ja mal gucken wie weit der kommt?“
B1	„Aber er sollte eigentlich schon ein bisschen weiter kommen“
B2	„Dann checken wir mal wo er rausfliegt ok? („ „ ,) Was ist denn (∼)?“
B1	„Ich würde sagen wir machen erstmal die Schrift ein bisschen kleiner („ ,) Fonts & Colors gibt es das irgendwo?“
B2	„Appearance? ganz oben“
B1	„Ah, ja (,) Jetzt müssen wir noch herausfinden welcher <*Editor*> es ist. Ich tippe mal auf den normalen Java-Editor. („ „ ,) Mm <*Verneinung*>“
B2	„Ganz oben, „Basic“ vielleicht?“
B1	„Ah es gibt auch den Standard Text Editor. Text Font („ ,) Ich hab nen Courier („ „ ,) Eh („) Das Problem ist halt, dass es nicht viel kleiner geht („) Kann ich hier eine 8 eintragen? Hm, irgendwie nicht so schön <*das Schriftbild auf dem Display*> (....) Wollen wir es so lassen und uns dran gewöhnen, dass wir nur“

B2	„Schön wäre Syntax-Highlighting, damit man irgendwie klarkommt, aber muss jetzt auch nicht sein wenn das <*zu aufwendig ist*>“
B1	„Ah, ich glaube er erkennt das hier gar nicht als .php-Projekt an“
B2	„Aber da oben steht php“
B1	„Achso, wir sind hier, ja stimmt“
B2	„<Räuspern> Naja“
B1	„Du sagtest, die MemCache-Klasse brauchen wir hier gar nicht noch einmal instanziieren?“
B2	„Mhm <*Zustimmung*> (.) Ich würde gerne wissen wo er da aussteigt, also“
B1	„get_<**Entität**>_ids“
B2	„Vielleicht hat der ja auch nur am Anfang die AutoCreate nicht oder so, das wissen wir ja alles nicht, wie weit er überhaupt kommt“
B1	„Hmh <*Zustimmung*>“
B2	„Deswegen, versuch doch noch einmal ein echo <**Name des Systems**> web-path oder so“
B1	„Ah, vielleicht müssen wir den Chat enablen“
B2	„Hm <*Zustimmung*>, oder so“
B1	„Das wäre eine Möglichkeit. Dafür brauchen wir den AutoCreate gar nicht. Ich mache jetzt erstmal ein workingSet. Resource, die heißt „<**Kürzel des Systems**> trunk“ („ ,) Okay, jetzt können wir mal das andere zuklappen. „autoPrepend.de“ („ „) Ich glaube wir müssen ihm jetzt noch beibringen, dass er php Sachen in seinem Editor öffnet („) „File Association“ heißt das („ „ „ ,) Na wir haben scheinbar keinen php-Editor. #Text Editor, den Java Editor#“
B2	„Ich nehme an, wie du schon sagst, das wird der Java Editor sein. Dass er da dann auch gleich Syntax hat.“
B1	„(„ „) Oookaaay, das sieht schon schöner aus <*es hat funktioniert was sie einstellen wollten - php Files werden nun im integrierten Editor angezeigt*>“
B2	„Ok wir müssen jetzt in die Conf rein um den Schalter zu gucken. Features. Ist das nicht „conf“ und nicht „autoPrepend“?“

B1	„(∼) klar (∼∼) („ „) EnableNimbuzzChat - ah guck <*hat den Wert von false auf true gesetzt*>. Jetzt öffnen wir den einmal. <*get_<**Entität**>.sh wird ausgeführt*> („) Ok, der connected immer noch nicht.“
B2	„Das scheint doch irgendwie ein Zugriffsrechteproblem zu sein“
B1	„<*führt get_<**Entität**>.php im Browser aus*> Okay, da ist unsere Liste. Dann lassen wir den Header einfach so lange draußen.“
B2	„Mhm <*Zustimmung*>“
B1	„Ähm, was macht denn das Skript noch? Das Skript ist einfach (∼∼). Dann würde ich sagen wir machen da einmal zumindest so ein bisschen <*Ordnung*>. Wir haben da ja keine Umbrüche drin oder so irgendetwas <. . . . . > Äh, ok <..> Das ist ja echt der Horror- Font hier. <..> Oookay. YEAH <*es funktioniert*>“

im weiteren Verlauf der Sitzung tritt die folgende Stelle auf: BA1 36.26 - 37.24

B1	„Wir können uns auch etwas ganz anderes überlegen was der dann zurückgibt. <*bezieht sich auf den Rückgabewert*>“
B2	„Nö, finde ich schon gut so. (∼)“
B1	„Letzte Änderung 0. Das Problem ist halt wenn der den aktuellen Timestamp zurückgeben würde, hm, wir haben so wie so ein Problem, dass wir die Uhren synchron halten und gerade wenn man jetzt einen Timestamp macht, dann könnte das echt sein, dass es um ein paar Sekunden, Millisekunden, (. . . .) ne Sekunden, aber dass es eben um ein, zwei Sekunden hinterherhinkt und dass dann unnötig etwas verschickt wird.“
B2	„Ja gut, pf, das nehmen wir jetzt erst einmal in Kauf würde ich sagen.“
B1	„Ok, ich würde mal sagen“
B2	„Juuuhuuu“
B1	„Die ganze Testerei lagern wir jetzt gleich mal hierhinein aus <*<**Entität**>.lib.php*> get_<**Entität**>_-IDS, da passt es ja irgendwie ganz gut hin.“

wiederum etwas später kommt es zu dem folgenden Szenario, dabei handelt es sich beinahe um die letzten Äußerungen der Sitzung: BA1 1:44:27 – 1:46:00

B1	„Ich sehe immer noch ein bisschen ein Problem mit der Uhrzeit. Wenn man das jetzt irgendwie simulieren will, dann bräuchte man die gleiche Uhrzeit wie das <**Systemname**>-System, damit das vernünftig funktioniert.“
B2	„Ok, gehen wir mal davon aus wir haben eine Stunde Differenz.“
B1	„Na gut, dann wäre es nicht so schlimm. Dann ist natürlich auch die Frage in welche Richtung der Versatz ist. Der könnte auf der einen Seite dazu führen, dass man denkt, es hat sich nichts verändert obwohl sich etwas geändert hat und andersherum könnte das dazu führen, dass man denkt, äh, es hat sich etwas geändert obwohl sich nichts geändert hat.“
B2	„Wie könnte man so etwas feststellen?“
B1	„Wie bei denen die Uhr tickt?“
B2	„mhm <*Zustimmung*>“
B1	„Im Idealfall könnten sie einfach einen Timestamp von jetzt mitschicken, aber die können ja nicht jedes mal einen Timestamp mitschicken nur damit wir gucken, äh, geht eure Uhr jetzt anders. Die könnten uns jede Minute oder jede 10 Sekunden“
B2	„Die könnten ein keep-alive <*senden*>“
B1	„Das wäre eine Idee. (.) Also ich meine es gibt bestimmt auch irgendwelche Mittel und Wege die Uhr von zwei Servern synchron zu halten.“
B2	„Ich notiere das einmal als Problem und dann können wir ja mal versuchen das zu lösen“
B1	„Also eigentlich müssen sie doch nur an den gleichen TimeServer angeschlossen sein, dann ist doch alles gut, also mit dem verbunden sein, von dem die ihre Uhrzeit beziehen und daran werden die wohl denken“
B2	„Es ist ja auch unser Server, daher die gleiche Farm und daher die gleiche Uhrzeit würde ich mal annehmen, aber das müssen wir mal klären.“
B1	„Ich glaube auch, dass die den gleichen Zeitserver haben“

Gleich die allererste hier aufgezeigte Äußerung „cURL kann sich immer noch nicht connecten“ ist besonders beachtenswert, referenziert sie doch eine Schwierigkeit, die im gesamten weiteren Verlauf des ersten Teils der Sitzung versucht wird auszuräumen. Solche Äußerungen, habe ich Problembild-Äußerungen genannt. Sie werden im Folgenden genauer erläutert.

6.1 Problembild-Äußerungen

Um genauer festzulegen wann im Folgenden von einer Problembild-Äußerung gesprochen wird ist hier meine Arbeitsdefinition angegeben.

Problembild-Äußerung

Eine Problembild-Äußerung ist eine verbale Äußerungen eines Entwicklers, die ein Hindernis für das Voranschreiten im (Arbeits-)Prozess referenziert.

An Hand dieser Definition kann man im vorgestellten Szenario mehrere Problembild-Äußerungen identifizieren. Zum Beispiel die folgenden (abgesehen von der einführenden Äußerung „cURL kann sich immer noch nicht connecten“):

> „Auflösen des Rechnernamen localhost“
> „Du hast keine Parameter mit angegeben in der get_<**Entität**><*.php*>“
> „Vielleicht hat der ja auch nur am Anfang die AutoCreate nicht oder so, das wissen wir ja alles nicht, wie weit er überhaupt kommt“
> „Schön wäre Syntax-Highlighting“
> „Ah, ich glaube er erkennt das hier gar nicht als .php-Projekt an“
> „Wir haben so wie so ein Problem, dass wir die Uhren synchron halten“
> „Ich sehe immer noch ein bisschen ein Problem mit der Uhrzeit.“

Da ich wie oben beschrieben zuerst die Episoden mit den Basiskonzepten versehen haben, konnte ich die folgenden Überschneidungen feststellen. Problembild-Äußerungen können also in *Basis-Konzepten* gesprochen wie folgt annotiert werden:
explain_finding, challenge_finding
explain_knowledge, ask_knowledge
propose_strategy, amend_strategy, ask_strategy
propose_step, decide_step, ask_step
disagree_design, challenge_design, ask_design
propose_todo

think aloud_activity
remember_requirement

Zusammenfassend lässt sich also feststellen, dass Problembild-Äußerungen eine Vielzahl von Annotationsmöglichkeiten in der Basisschicht haben können. Da ich an dieser Stelle lediglich Überschneidungen aufzeigen möchte, werde ich hier nur Bezug auf die Objekte (also den hinteren Teil der Bezeichnung eines Konzepts) der oben beschriebenen Konzepte nehmen. Eine Problembild-Äußerungen kann demnach zum Beispiel Erkenntnisse bzw. Wissen die entweder zum Bestandswissen eines Entwicklers gehören oder aber soeben entstanden sind (finding & knowledge) betreffen. Ebenfalls können Problembild-Äußerungen kurzfristige so wie auch längerfristige Planungen im Arbeitsprozess (step & strategy) referenzieren. Darüber hinaus können Problembild-Äußerungen eine Gestaltungsoption des vorliegenden Software-Artefakts betreffen, so wie an Vorgaben die zu selbigen gemacht worden sind in Erinnerung zurück rufen (design & requirement). Äußerungen die Problembilder verbalisieren können zudem einen Arbeitschritt betreffen, der nicht aktuell, sondern in Zukunft durchgeführt werden soll. Außerdem werden Problembilder oft in Form eines think-alouds geäußert.
Aufgrund der soeben dargestellten Vielfältigkeit ist eine *dimensionale Betrachtung* von Problembild-Äußerungen sinnvoll. Während meiner Studien habe ich eine Vielzahl von Eigenschaften entdeckt entlang derer sich die verschiedenen Problembild-Äußerungen dimensionalisieren ließen. Als Beispiel sei hier die Gedankentiefe angeführt. Die Gedankentiefe lässt sich an Hand der dimensionaler Reichweite aufzeigen die von „ohne Denken“ bis zu „gedankenintensiv“ reicht. Eine andere Eigenschaft von Problembild-Äußerungen ist ob es sich um eine Bewertung einer vorhergegangenen Aktion im PP-Prozess handelt. Bereits an der Formulierung und dem Vorkommen des Worts „ob“ ist zu erkennen, das es sich hierbei um eine binäre Eigenschaft handelt.
Eine weitere solche Eigenschaft konnte ich feststellen, nämlich ob es sich um ein „geschlussfolgertes Problembild“ handelt oder nicht. Unter einem geschlussfolgerten Problembild verstand ich eine Verbalisierung eines Problembild, das sich ohne erkennbare äußere Einflüsse im Kopf des Entwicklers unter Einsatz von kognitiver Leistung gebildet haben musste.

Ähnliche Eigenschaften konnte auch Salinger feststellen. Er unterteilt das Konzept finding in „perceived event“, „discovered issue“ und „thought“.

> „Eine Äußerung ist vom Typ P (perceived event), wenn in ihr die Wahrnehmung (in der Regel durch Sehen oder Hören) von sich zeitnah in der Umgebung der Person abgespielten Ereignissen bzw. Geschehnissen verbalisiert wird. [. . .] 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.“ ([34], S. 252).

Dies ist in etwa mit dem zu vergleichen, was ich als „Bewertung einer vorhergegangenen Aktion“ bezeichnet habe.
Äußerungen vom Typ D „discovered issue“ sind

> „Äußerungen, die einen direkten Bezug zu auf dem Display bzw. Monitor oder in einem Dokument dargestellten und offensichtlich auch gerade betrachteten Informationen [. . .] besitzen.“ ([34], S. 253).

Der Typ D ist also in etwa mit dem zu vergleichen was ich bei meiner Dimensionalisierung von Problembild-Äußerungen mit „ohne Denken“ bezeichnet habe.
Äußerungen vom Typ T „thought“ sind

> „Äußerungen, mit denen die Ergebnisse von Überlegungen oder spontanen bzw. spontan erscheinenden Einfällen und Einsichten verbalisiert werden.“ ([34], S.254).

Dies ist wiederum grob mit dem zu vereinbaren was ich ein „geschlussfolgertes Problembild“ genannt habe.
An dieser Stelle sei angemerkt, dass Salinger die hier vorgestellten Kategorien noch weiter unterteilt hat. Die soeben vorgestellte Granularität der Unterteilung ist jedoch hinreichend um die Ähnlichkeit der Erkenntnisse aufzuzeigen.
Abgesehen von den bisher genannten, konnte ich eine grundsätzlichere Eigenschaft ausmachen, die ich Explizitheit genannt habe. Damit ist gemeint wie explizit - falls überhaupt - das Problembild was ein Entwickler gerade im Kopf hat verbalisiert wird. Die Dimension über die sich die Explizitheit also erstreckt reicht von „implizit“ bis „explizit“. Ein Beispiel einer expliziten Problembild-Äußerung ist die bereits mehrfach referenzierte Aussagen „cURL kann sich immer noch nicht connecten“. Dementgegen als mittelmäßig explizites Problembild-Äußerung ist die Äußerung „Fehlgeschlagen. Hm, was ist denn da los?“. Ein implizite Problembild-Äußerung ist genau genommen gar keine Äußerung, sondern eine Tätigkeit aus der man das Vorliegen eines Hindernisses im Arbeitsprozess erkennen kann.
Die bisher vorgestellten Eigenschaften haben sich jedoch im weiteren Verlauf meiner Studie als zu feingranular erwiesen bzw. sie schienen nicht das relevante Unterscheidungsmerkmal zwischen den verschiedenen Problembild-Äußerungen widerzuspiegeln. Erst nach vielen Überlegungen kam ich zu der Erkenntnis, dass das (für meine Studie) relevante Merkmal der *zeitliche Bezug* der Äußerungen ist. Die Dimension über die sich der zeitliche Bezug erstreckt erstreckt sich dabei von Vergangenheit bis Zukunft. Jedoch kann man die Vergangenheit bis

inklusive der Gegenwart zusammenfassen, sodass sich an Hand dieser Eigenschaft zwei verschieden Problembild-Äußerungen unterscheiden lassen: potentielle Problembild-Äußerungen und effektive Problembild-Äußerungen.
Potentielle Problembild-Äußerungen sind dabei Äußerungen, die ein Problembild adressieren welches noch in der Zukunft liegt, dessen Auftreten also noch nicht bestätigt ist. Hingegen sind effektive Problembild-Äußerungen solche, die ein Problem adressieren, dessen Auftreten bereits festgestellt werden konnte.
Aus der soeben beschriebenen Dimensionalisierung ergeben sich die folgenden Konzepte:

- **explain_effective problem**
  Als Beispiel hierfür kann die Äußerung:
  „cURL kann sich immer noch nicht connecten“ angeführt werden.

- **explain_potential problem**
  Hier kann als geeigneter Repräsentant die Äußerung:
  „Hm, wir haben so wie so ein Problem, dass wir die Uhren synchron halten und gerade wenn man jetzt einen Timestamp macht, dann könnte das echt sein, dass es um ein paar Sekunden, Millisekunden, (....) ne Sekunden, aber dass es eben um ein, zwei Sekunden hinterherhinkt und dass dann unnötig etwas verschickt wird.“ angeführt werden.

Wie bei der Definition von Problembild-Äußerungen beschrieben dienen sie der Verbalisierung von Hindernissen im Arbeitsprozess. Um diese Hindernisse aus dem Weg zu räumen müssen sie entweder gelöst werden oder aber umgangen werden (work around). Es ist wohl naheliegend zu unterstellen, dass die Entwickler zuerst versuchen werden das Problem zu lösen, bevor sie einen work around versuchen würden zu finden. So überrascht es nicht, dass ebenfalls in dem Beispiel kurz auf die Problembild-Äußerung die folgende Äußerung getätigt wird: „Aber cURL ist schon noch drauf oder [...]“ Diese Äußerung beschreibt eine Möglichkeit wie das zuvor geäußerte effektive Problembild zu Stande gekommen sein könnte. Solche Äußerungen habe ich mit dem Konzept Lösungsbild-Äußerungen annotiert.

6.2 Lösungsbild-Äußerungen

Um auch hier genauer festlegen zu können wann im Folgenden von einer Problembild-Äußerung gesprochen wird ist hier meine Arbeitsdefinition angegeben.

Lösungsbild-Äußerung

Eine Lösungsbild-Äußerung ist eine verbale Äußerungen eines Entwicklers, die eine Lösung(-sidee) eines vorher geäußerten Problembilds referenziert.

An Hand dieser Definition kann man im eingangs dieses Kapitels vorgestellten Szenario erneut eine Mehrzahl an Lösungsbild-Äußerungen lokalisieren. Als weitere Beispiele seien hier die folgenden Äußerung angegeben:

„Versuch' mal ein wget"
„Mach' mal auf localhost statt dev-intern"
„Firewall?"
„Vielleicht nur die (∼) Ne, die hat er ja aufgelöst"
„(∼) ist okay"
„Wir könnten allerdings jetzt den MimeType herausnehmen zum Entwickeln."
„Appearance? ganz oben"
„Ah, vielleicht müssen wir den Chat enablen"
„Ok, gehen wir mal davon aus wir haben eine Stunde Differenz."

Auch bei Problemlösungsbild-Äußerungen möchte ich die Überschneidungen zwischen diesem Konzept und den Konzepten aus der *Basisschicht* darstellen. Problemlösungsbild-Äußerungen habe ich durch die folgenden Basis-Konzepte versehen können:
propose_step, amend_step, disagree_step
ask_strategy, amend_strategy, agree_strategy, propose_strategy
propose_design, disagree_design, amend_design
explain_finding, amend_finding, agree_finding
explain_knowledge, ask_knowledge
propose_hypothesis, disagree_hypothesis
think aloud_activity
remember_requirement

Auch hier kann man sehen, dass Problemlösungsbild-Äußerungen in vielerlei Basis-Konzepte unterschieden werden können. Ebenfalls sollen hier erneut nur die Objekte der Basiskonzepte (also der zweite Teil der Bezeichnung) zur Analyse herangezogen werden um die Überschneidung anzudeuten. Demnach können

Lösungsbild-Äußerungen sowohl eine kurzfristige als auch längerfristige Planungen im Arbeitsprozess (step & strategy) referenzieren. Außerdem können Äußerungen die ein Lösungsbild referenzieren eine Gestaltungsoption des vorliegenden Software-Artefakts betreffen, so wie an Vorgaben die zu selbigen gemacht worden sind in Erinnerung zurück rufen (design & requirement). Ferner können Lösungsbild-Äußerungen soeben erlangte Erkenntnisse, so wie bereits vor der Sitzung in der sie geäußert werden vorhandenes Wissen betreffen (finding & knowledge). Es ist außerdem festzustellen, dass Lösungsbild-Äußerungen oft in Form eines think-alouds geäußert werden und dass sie auch hypothetische Aussagen sein können (hypothesis).

Auch bei Lösungsbild-Äußerungen ist aufgrund der soeben dargestellten Vielfältigkeit eine *dimensionale Betrachtung* unabdingbar. Während meiner Studien habe ich eine Vielzahl von Eigenschaften entdeckt entlang derer sich die verschiedenen Lösungsbild-Äußerungen dimensionalisieren ließen. Als Beispiel ist die Eigenschaft der Erkenntnishöhe anzuführen, die eine Äußerung hat. Entlang dieser Eigenschaft ergibt sich eine Dimension die sich von gar keiner Erkenntnis, über eine Ursachenbeschreibung, hin zu einer detaillierten Handlungsanweisung erstreckt. Als Beispiel für eine Lösungsbild-Äußerung mit geringster Erkenntnishöhe wäre das hypothetische Beispiel „Ich habe keine Ahnung woran es liegt“ als Antwort auf die Frage warum sich beispielsweise cURL nicht verbinden kann. Als weitere Beispiele dienen hier wieder die oben identifizierten realen Äußerungen. „Firewall?“ ist eine Lösungsbild-Äußerung deren Erkenntnishöhe ich als mittel bis gering einstufen würde, da sie zwar einen Vorschlag liefert, dieser ist aber nicht begründet, denn es ist zu bedenken ist dass der Sprecher in dieser Situation der Observer ist und auch keine Handlungen stattgefunden haben, die diesen Vorschlag als begründet erscheinen lassen. Die Lösungsbild-Äußerung „Mach' mal auf localhost statt dev-intern“ würde ich als eine mit hoher Erkenntnishöhe beurteilen, da sie einerseits die vermutete Ursache des Problems in der angegeben URL vermutet und andererseits eine konkrete Lösung vorschlägt.
Abgesehen von der Erkenntnishöhe habe ich eine Eigenschaft entdecken können, die ich Realismus benannt habe. Die Dimension des Realismus von Lösungsbild-Äußerungen erstreckt sich von effektiv nach hypothetisch. Eine Äußerungen, die ich als hypothetisch einstufen würde ist die folgenden: „Ok, gehen wir mal davon aus wir haben eine Stunde Differenz.“ Alle anderen Äußerungen bewegen sich eher im Bereich von realen Lösungsbild-Äußerungen.
Eine Eigenschaft, die ich jedoch nicht in den Daten aufgefunden habe, welche mir aber sinnvoll erscheint, ist ob eine Problembild-Äußerung den möglichen Lösungsraum eines Problems einschränkt oder vergrößert. Da ich diese aber nicht in den Daten finden konnte, habe ich diese Eigenschaft fallen gelassen.

Eine weitere Eigenschaft von Lösungsbild-Äußerungen ist die Explizitheit. Diese ist ähnlich wie bei Problembild-Äußerungen dimensionalisierbar von „implizit“ bis „explizit“. Eine explizite Lösungsbild-Äußerung ist beispielsweise „Versuch' mal ein wget“, wohingegen eine implizite Lösungsbild-Äußerung genau genommen wiederum gar keine Äußerung ist, sondern eine Handlung, beispielsweise das eigenständige Bearbeiten des Quellcodes durch einen Entwickler. Das Problem impliziter Lösungsbild-Äußerungen ist, dass sie den Partner ausschließen und dieser allenfalls durch aufmerksame Beobachtung Schritt halten kann.
Die eben durchgeführte Dimensionalisierung von Lösungsbild-Äußerungen führt also zu den folgenden Konzepten:

- **propose_solution**
  Ein Beispiel für dieses Konzept aus dem Beispielszenario ist:
  „Mach' mal auf localhost statt dev-intern“

- **propose_scenario**
  Ein Repräsentant dieses Konzepts ist die Äußerung:
  „Ok, gehen wir mal davon aus wir haben eine Stunde Differenz.“

Wie im zweiten Abschnitt des Beispiels zu sehen folgen auf Problembild-Äußerungen aber nicht zwangsläufig Lösungsbild-Äußerungen. Ich konnte in den Daten noch ein weiteres Vorgehen erkennen, nämlich das „Zurückweisen“ einer Problembild-Äußerung durch das Zuweisen von geringer Relevanz zu dem zuvor geäußerten Problem. Solche Äußerungen habe ich Relevanz-Äußerungen genannt.

6.3 Relevanz-Äußerungen

Auch hier soll wieder mit meiner Arbeitsdefinition von Relevanz-Äußerungen begonnen werden, um im Folgenden wissen zu können wann von einer Relevanz-Äußerung gesprochen wird:

Relevanz-Äußerung

Eine Relevanz-Äußerung ist eine verbale Äußerung eines Entwicklers, die die Relevanzeinschätzung des aktuellen Themas aus Sicht des Sprechers beschreibt.

An Hand dieser Definition kann man im Beispielszenario erneut mehrere Relevanz-Äußerungen finden, beispielsweise die folgenden:

> „Nö, finde ich schon gut so. (∼)“
> „Ja gut, pf, das nehmen wir jetzt erst einmal in Kauf würde ich sagen.“

„Wollen wir es so lassen und uns dran gewöhnen, dass wir nur“

Ich konnte feststellen, dass Relevanz-Äußerungen ebenfalls in mehrere Basis-Konzepte unterschieden werden können. Dieses waren in meinen Daten die folgenden:
propose_strategy, ask_strategy
disagree_step
disagree_design
propose_todo

In der gleichen Manier wie bei den vorherig vorgestellten Konzepten möchte ich hier die Überschneidungen mit der Basisschicht darstellen. Demnach können Relevanz-Äußerungen sowohl eine kurzfristige als auch längerfristige Planungen im Arbeitsprozess (step & strategy) referenzieren. Außerdem können Äußerungen die eine Relevanzeinschätzung zum Ausdruck bringen eine Gestaltungsoption des vorliegenden Software-Artefakts betreffen. Ferner ist festzustellen, dass Relevanz-Äußerungen oft einen Arbeitschritt betreffen, der nicht aktuell, sondern in Zukunft durchgeführt werden soll betreffen (todo).

Bei der Betrachtung der *Dimensionalität* erschien mir die relevante Eigenschaft zur Unterscheidung für meine Betrachtung relativ schnell gefunden zu sein, nämlich die geäußerte Relevanz selber. Diese erstreckt sich dimensional von niedrig bis hoch. Alle oben identifizierten Relevanz-Äußerungen sind Äußerungen, die eine niedrige Relevanz aufzeigen. Es gibt aber auch Äußerungen, die eine hohe Relevanz zum Ausdruck bringen, zum Beispiel in **CA2.2 35.49 - 36.00**
Kurzbeschreibung der Situation:
Die beiden Entwickler diskutieren darüber ob sie eine zusätzliche Abstraktionsschicht zwischen zwei verschiedene Modelle einziehen wollen.

C1	„Die Frage ist ob ich mich da jetzt rantrauen will, ob wir uns da rantrauen wollen“ ⇒ niedrige Relevanz
C2	„Ich würde mich da schon gerne rantrauen wollen weil das eine Chance ist das leichtgewichtiger zu machen über diesen Arbeitspunkt“ ⇒ hohe Relevanz

An diesem Beispiel lässt sich eine weitere Eigenschaft demonstrieren, nämlich die, ob der Partner mit der geäußerten Relevanz übereinstimmt oder nicht. In dem soeben aufgezeigten Fall war dies offensichtlich nicht der Fall. Oft ist es aber zu beobachten, dass ein Partner mit der verbalisierten Relevanz-Einschätzung seines Gegenübers übereinstimmt.

Aus der soeben beschriebenen Dimensionalisierung ergeben sich die folgenden Konzepte:

- **explain_low_relevance**
  Als Beispiel sei hier die folgende bereits weiter oben angeführte Äußerung: „Nö, finde ich schon gut so. (∼)“
- **explain_high_relevance**
  Als Beispiel sei hier ebenfalls die bereits weiter oben angeführte Äußerung: „Ich würde mich da schon gerne rantrauen wollen weil das eine Chance ist das leichtgewichtiger zu machen über diesen Arbeitspunkt“

Im Folgenden möchte ich eine weitere Beobachtung meiner Studie der gemeinsamen Wissensproduktion aufzeigen. Dabei handelt es sich um das Thema einer PRODUCE-Phase. Wie in Kapitel 5.2 gesagt, empfand ich die Untergliederung in Episoden als zu eng, da ich bei genauerer Betrachtung beobachten konnte, dass man das Thema in unterschiedliche Ebene unterteilen kann und dass gewisse Äußerungen zu einer Veränderung der aktuell besprochenen Themenebene führen. Diese Veränderungen des Themas habe ich TOPIC-Veränderung genannt.

6.4 TOPIC-Veränderungen

Auch bei TOPIC-Veränderungen soll damit begonnen werden, dass meine Arbeitsdefinition von TOPIC-Veränderungen angegeben wird, damit im weiteren Verlauf der Arbeit klar ist wann von einer solchen gesprochen wird.

TOPIC-Veränderung

Eine TOPIC-Veränderung ist eine verbale Äußerungen eines Entwicklers, die das aktuelle Thema in ihrem thematischen Umfang verändert, also entweder fokussiert oder divergiert.

Auch hier sollen zum besseren Verständnis ein weiteres Beispiel aus dem einführenden Szenario angegeben werden, jedoch muss dazu in diesem Fall ein wenig mehr Kontext angegeben werden als bei den vorherigen Konzepten, da eine Veränderung ja immer nur festzustellen ist wenn man der vorherigen Zustand (in diesem Fall das Thema) kannte:

B1	„Wir können uns auch etwas ganz anderes überlegen was der dann zurückgibt. <*bezieht sich auf den Rückgabewert*>“
B2	„Nö, finde ich schon gut so. (∼)“
B1	„Letzte Änderung 0. Das Problem ist halt wenn der den aktuellen Timestamp zurückgeben würde, hm, wir haben so wie so ein Problem, dass wir die Uhren synchron halten und gerade wenn man jetzt einen Timestamp macht, dann könnte das echt sein, dass es um ein paar Sekunden, Millisekunden, (. . . .) ne Sekunden, aber dass es eben um ein, zwei Sekunden hinterherhinkt und dass dann unnötig etwas verschickt wird.“
B2	„Ja gut, pf, das nehmen wir jetzt erst einmal in Kauf würde ich sagen.“

In dieser Sequenz von Äußerungen führt die Aussage „wir haben so wie so ein Problem, dass wir die Uhren synchron halten [. . .]“ zu einer Veränderung des aktuell besprochenen Themas, denn davor ging es um den möglichen Rückgabewert des Timestamps und danach geht es um das potentielle Problembild der Synchronität der Uhren der beiden miteinander kommunizierenden Systeme.

Auch bei TOPIC-Veränderungen sollen die gefundenen Annotationen aus der *Basisschicht* dargestellt werden. Es handelt sich dabei um die folgenden:
propose_design, amend_design, challenge_design, ask_design
propose_step, decide_step, challenge_step
ask_knowledge, challenge_knowledge
propose_strategy
explain_gap in knowledge
think aloud_activity
agree_state
explain_finding
propose_todo

Als Schlussfolgerung dieser Aufzählung denke ich, dass es genügt festzustellen, dass vielerlei verschiedene Tätigkeiten (nach Basisschicht) zu einer Veränderung der Themenebene führen können, weswegen TOPIC-Veränderungen nun genauer *dimensionalisiert* werden.
Als Eigenschaften bei Veränderungen der Themenebene konnte ich mehrere ausmachen. Hier sei als Beispiel die Unterscheidung angeführt ob es sich bei dem Themenwechsel um eine kurze Zwischenfrage handelt, von der der Fragende ausgeht, dass sein Partner sie schnell beantworten kann (im Sinne eines ask_know-

ledge), oder ob es eher eine Aufforderung ist über das ins Gespräch gebrachte Thema zu kommunizieren. Diese Eigenschaft nenne ich Absicht. Sie erstreckt sich von unabsichtlich bis absichtlich und beschreibt die willentliche Intention die aktuelle Themenebene zu verändern.
Das zentrale Merkmal einer thematischen Veränderung ist es aber meiner Meinung nach nicht ob sie absichtlich herbeigeführt wurde oder nicht, da das Thema dann gegebenenfalls ohnehin bereits verändert wurde. Als das relevante Unterscheidungsmerkmal für meine Studie hat sich jedoch die Veränderung der Extension herausgestellt. Als Extension (oder dem Umfang eines Begriffs) versteht man in der Begriffslogik die Gesamtheit der Dinge, auf die er sich erstreckt, also die unter ihn fallen oder die er umfasst [22]. In der vorliegenden Arbeit soll die Extension den Umfang des aktuell vorliegenden Themas denotieren. Es gibt zwei Möglichkeiten die Extension des aktuell bearbeiteten Themas zu verändern:
Diese sind zum Einen eine Verschmalerung der Extension, also die Fokussierung auf beispielsweise einen Teilaspekt des Themas (Unterthema), oder dem entgegengesetzt die Verbreiterung der Extension auf beispielsweise einen übergeordneten Aspekt des aktuell vorliegenden Themas (darüberliegendes Thema), im Folgenden Divergenz genannt.
Aus der soeben beschriebenen Dimensionalisierung ergeben sich die folgenden Konzepte:

- **propose_topic focus**
  Ein Beispiel für einen thematischen Fokus ist die folgende Äußerungssequenz, genau genommen nur die zweite Äußerung, da sie die Extension der Diskussion fokussiert:
  „Ich sehe immer noch ein bisschen ein Problem mit der Uhrzeit. Wenn man das jetzt irgendwie simulieren will, dann bräuchte man die gleiche Uhrzeit wie das <**Systemname**>-System, damit das vernünftig funktioniert."
  „Ok, gehen wir mal davon aus wir haben eine Stunde Differenz."
  „Na gut, dann wäre es nicht so schlimm. Dann ist natürlich auch die Frage in welche Richtung der Versatz ist. [...]"

- **propose_topic divergence**
  Ein Beispiel für eine thematische Divergenz ist die folgenden Äußerungssequenz. Hier ist ebenfalls die entscheidende Äußerung in kursiv dargestellt:
  „Wie könnte man so etwas feststellen?"
  „Wie bei denen die Uhr tickt?"
  „mhm <*Zustimmung*>"
  „Im Idealfall könnten sie einfach einen Timestamp von jetzt mitschicken, aber die können ja nicht jedes mal einen Timestamp mitschicken nur damit wir gucken, äh, geht eure Uhr jetzt anders. Die könnten uns jede Minute oder jede 10 Sekunden"

„Die könnten ein keep-alive <*senden*>“
„Das wäre eine Idee. (.) Also ich meine es gibt bestimmt auch irgendwelche Mittel und Wege die Uhr von zwei Servern synchron zu halten.“
„Ich notiere das einmal als Problem und dann können wir ja mal versuchen das zu lösen“
„Also eigentlich müssen sie doch nur an den gleichen TimeServer angeschlossen sein, dann ist doch alles gut, also mit dem verbunden sein, von dem die ihre Uhrzeit beziehen und daran werden die wohl denken“
„Es ist ja auch unser Server, daher die gleiche Farm und daher die gleiche Uhrzeit würde ich mal annehmen, aber das müssen wir mal klären.“
„Ich glaube auch, dass die den gleichen Zeitserver haben“

6.5 Funktionen der Konzepte

Bisher wurden die genannten Konzepte trotz ihrer Einbettung in das einführende Beispiel relativ isoliert betrachtet. Dies soll sich in diesem Abschnitt ändern. Es werden also nun die beobachteten Funktionen, die die Konzepte auf die gemeinsame Wissensproduktion haben, dargestellt. Dazu wird in der selben Reihenfolge vorgegangen wie im vorherigen Kapitel. Die Funktionen der einzelnen Konzepte sind teilweise auch gut durch die Überschneidung, die die Konzepte untereinander haben darzustellen, was im Folgenden ebenfalls erläutert wird.

6.5.1 Funktionen von Problembild-Äußerungen

Problembild-Äußerung ⇔ TOPIC-Veränderung
Problembild-Äußerungen referenzieren nach Definition (Kapitel 6.1) Hindernisse, die dem Paar bei ihrem Arbeitsprozess im Weg stehen. Sie beschreiben somit ebenfalls auch das Thema mit dem sich das Paar beschäftigen muss um eben dieses Hindernis zu bewältigen. Den beiden Partnern wird durch eine effektive Problembild-Äußerung eben aufgezeigt in welchem Bereich sie noch Arbeit investieren müssen um ihren Arbeitsprozess vorantreiben zu können. Dies kommt einer TOPIC-Veränderung bereits nahe. Dies entspricht auch den Beobachtungen die ich machen konnte: Problembild-Äußerungen sind oftmals TOPIC-Veränderungen. Ich vermute, dass dies absichtlich geschieht und eine Problembild-Äußerung somit einer absichtlichen TOPIC-Veränderung gleich kommen kann.

Wiederholte Problembild-Äußerungen
Die nächste Funktion von Problembild-Äußerungen, die ich hier angeben möchte

ist, dass Problembild-Äußerungen wiederholt geäußert werden können wie auch im einführenden Beispiel zu sehen ist

> „wir haben so wie so ein Problem, dass wir die Uhren synchron halten [...]“
> eine Weile später thematisiert der selbe Entwickler das gleiche Thema noch einmal:
> „Ich sehe immer noch ein bisschen ein Problem mit der Uhrzeit. Wenn man das jetzt irgendwie simulieren will, dann bräuchte man die gleiche Uhrzeit wie das <**Name des Systems**>-System, damit das vernünftig funktioniert.“

Die erste Äußerung des Entwicklers ist ein *explain_potential problem.*
Diese wurde jedoch vom Partner schnell mittels seiner Äußerung: „Ja gut, pf, das nehmen wir jetzt erst einmal in Kauf würde ich sagen.“ abgetan und danach nicht mehr besprochen.
Da jedoch wie oben gesagt Problembild-Äußerungen als TOPIC-Veränderungen fungieren können vermute ich, dass der andere Entwickler im weiteren Verlauf der Sitzung noch einmal versucht sein potentielles Problembild zur Sprache zu bringen und mit dem Partner darüber kommunizieren möchte. Dies gelingt ihm auch, wie man am weiteren Verlauf nach seiner Äußerung: „Ich sehe immer noch ein bisschen ein Problem mit der Uhrzeit. [...]“ sehen kann (siehe Beispielsszenario). Daraus schließe ich also, dass wiederholte Problembild-Äußerungen die Funktion haben ein Thema zur Sprache zu bringen, das bereits vorher erfolglos geäußert wurde und als TOPIC-Veränderung intendiert war.

6.5.2 Funktionen von Lösungsbild-Äußerungen

Die Funktion von Lösungsbild-Äußerungen ist nach der Arbeitsdefinition von oben ein durch ein Problembild referenziertes Hindernis versuchen zu lösen. Außer dieser offensichtlichen Funktion ein zuvor erkanntes Hindernis aus dem Arbeitsprozess zu schaffen ist mir bisher keine weitere Funktion aufgefallen.

6.5.3 Funktionen von Relevanz-Äußerungen

Niedrige Relevanz ⇔ TOPIC-Veränderung rückgängig machen
Bei den Funktionen von Problembild-Äußerungen ist gesagt worden, dass sie einer TOPIC-Veränderung gleichkommen können. An dieser Stelle ist jedoch zu sagen, dass dies komplexer ist als oben angedeutet. Betrachten wir noch einmal die erste potentielle Problembild-Äußerung und deren weiteren Verlauf:

> „Letzte Änderung 0. Das Problem ist halt wenn der den aktuellen Timestamp zurückgeben würde, hm, wir haben so wie so ein Problem,

dass wir die Uhren synchron halten und gerade wenn man jetzt einen Timestamp macht, dann könnte das echt sein, dass es um ein paar Sekunden, Millisekunden, (....) ne Sekunden, aber dass es eben um ein, zwei Sekunden hinterherhinkt und dass dann unnötig etwas verschickt wird."
„Ja gut, pf, das nehmen wir jetzt erst einmal in Kauf würde ich sagen."
„Ok, ich würde mal sagen [...] die ganze Testerei lagern wir jetzt gleich mal hierhinein aus <*<**Entität**>.lib.php*> [...]"

Die zweite Äußerung in diesem Abschnitt ist eine Relevanz-Äußerung, genauer genommen ein *explain_low relevance*. In der nächsten Äußerung kann man erkennen, das es nicht weiterhin um das zuvor geäußerte potentielle Problembild geht, sondern um etwas, das das Design des Codes betrifft. Das *explain_low relevance* hat also als TOPIC-Veränderung gewirkt. Die Beobachtung, dass eine niedrige Relevanz-Äußerung eine soeben vorgenommene TOPIC-Veränderung rückgängig macht, konnte ich mehrfach im Datenmaterial machen. Dies könnte man eventuell mit einem sofortigen Rücksprung durch ein return der aufgerufenen Funktion in der Programmierung vergleichen. Relevanz-Äußerungen scheinen also die TOPIC-Veränderungen steuern zu können.

Implizite Relevanz-Äußerungen
Soeben wurde gesagt, dass Relevanz-Äußerungen TOPIC-Veränderungen steuern können. Dies gilt natürlich nur für explizit verbalisierte Relevanz-Äußerungen. Oft scheint es jedoch der Fall zu sein, dass Relevanz-Äußerungen implizit getätigt werden, also gar nicht verbalisiert werden. Dies gilt insbesondere für *explain_high relevance*, denn diese Äußerungen habe ich während meiner Studie nur gesehen wenn zuvor ein *explain_low relevance* geäußert wurde. Es scheint also so als würden TOPIC-Veränderungen per default implizit akzeptiert werden. Ist dies nicht der Fall, versucht der Partner die TOPIC-Veränderung durch ein explizites *explain_low relevance* abzuwehren. Ebenfalls die Reaktion auf ein *explain_low relevance* scheint im Allgemeinen implizit von Statten zu gehen, da zumeist die TOPIC-Veränderung rückgängig gemacht wird und an der alten Stelle weiter gearbeitet wird. Ein einziges Mal konnte ich einen „Streit" über die Relevanz eines Themas in den Aufzeichnungen finden, in
CA2.2 35.49 - 36.00

C1	„Die Frage ist ob ich mich da jetzt rantrauen will, ob wir uns da rantrauen wollen" ⇒ niedrige Relevanz
C2	„Ich würde mich da schon gerne rantrauen wollen weil das eine Chance ist das leichtgewichtiger zu machen über diesen Arbeitspunkt" ⇒ hohe Relevanz

6.5.4 Funktionen von TOPIC-Veränderungen

TOPIC-Veränderungen dienen zunächst einmal offensichtlicherweise dazu das aktuell besprochene Thema zu verändern. Darüberhinaus vermute ich aber, dass sie die Funktion erfüllen den auf dem aktuellen Stand zu halten und auf diese Art und Weise seine Aufmerksamkeit und seine kognitiven Fähigkeiten nutzen zu können. Außer diesen offensichtlichen Funktion ist mir bisher keine weitere Funktion von TOPIC-Veränderungen aufgefallen.
Anzumerken ist jedoch, dass es Überschneidungen zwischen den Lösungsbild-Äußerungen, Relevanz-Äußerungen und TOPIC-Veränderungen gibt. So kann beispielsweise ein *propose_topic focus* gleichzeitig ein *propose_scenario* sein:

> „Ok, gehen wir mal davon aus wir haben eine Stunde Differenz."

Außerdem kann beispielsweise ein *propose_topic divergence* ebenfalls ein *explain_low relevance* sein:

> „Das wäre eine Idee. (.) Also ich meine es gibt bestimmt auch irgendwelche Mittel und Wege die Uhr von zwei Servern synchron zu halten."

6.6 Zusammenspiel der Konzepte

Da nun ein grobes Verständnis davon vorliegt wie PP-Partner gemeinsam Wissen produzieren ist eine weitere offene Frage der ich mich gerne annehmen möchte die folgende:

> Produziert das Paar das neue Wissen *systematisch*? Und falls ja, *wie genau*?

Im Rahmen dieser Überlegung war es zunächst naheliegend ein Problem zu konzeptualisieren und dann die Abarbeitung dieses Problems zu verfolgen, um eventuell einen musterhaften Ablauf erkennen zu können. Um ein Problem zu konzeptualisieren kam mir schnell der Gedanke dies als Baum (im mathematischen Sinne) zu tun, da dies in der Algorithmik ein beliebtes Vorgehen ist [17]. Dabei sollte ein Problembaum wie folgt aufgebaut sein:
An der Wurzel des Baums steht der Ausgangspunkt der Produktionsphase – also das Problembild. Alle Kinder des Baums sind Teilaspekte der Lösung des Problems. Dabei wird der Baum von links nach rechts gefüllt. Ein Kindeskind des Wurzelknotens wäre somit ein Teilaspekt eines Teilaspekts der Lösung des Problembilds. Ein beispielhafter Problembaum ist in Abbildung 14 gezeigt.
Als mögliche systematische Abarbeitung eines solchen Baums drängten sich natürlich die Algorithmen der Breiten- bzw. Tiefensuche [25] auf. In der Hoffnung

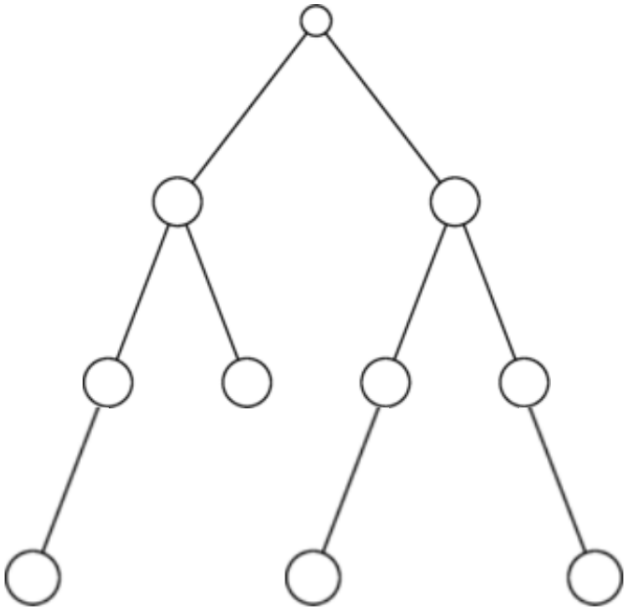

Abbildung 14: Beispiel eines Problemlösungsbaums. An der Wurzel steht der Ausgangspunkt der Episode – das Problembild. Kinder stellen einen Teilaspekt der Lösung des Problems dar.

solche Muster entdecken zu können, begann ich nun die ersten Problembäume an Hand des Sitzungmaterials aufzuzeichnen.

Dabei fiel mir zunächst auf, dass es schwierig nachzuvollziehen ist welche Lösungsbild-Äußerung an welcher Stelle des Baums einzutragen ist (abgesehen von der ersten Lösungsbild-Äußerung die immer das erste Kind zur linken Seite ist). Die einzig verlässliche Methode schien zu sein die Lösungsbild-Äußerungen alle als Kinder des Problembilds einzutragen, was natürlich zu einem breiten Baum der Höhe eins führt. Dies brachte mich ins Zweifeln darüber ob der gewählte Ansatz überhaupt geeignet ist.
Um diesen Zweifeln zu begegnen transkribierte ich (unter anderen) das Beispielszenario und las den erstellten Text mehrmals aufmerksam durch. Ich stellte fest, dass sich auch bei genauester Betrachtung keinerlei Hinweise darauf finden lassen, dass die Entwickler ein Baum-Modell des Problems im Kopf haben - zumindest wurde dies in den mir vorliegenden Daten nicht verbalisiert. Was hätte es also für einen Sinn eine Baumstruktur zu postulieren und diese aus den Daten zu rechtfertigen wenn den Entwicklern diese Struktur gar nicht bewusst ist. Wie eingangs dieser Arbeit angesprochen liegt ja das langfristige Ziel der Forschung der AGSE darin den Entwicklern Verhaltens-Muster an die Hand zu geben. Wenn aber die Entwickler sich dieser Baum-Struktur gar nicht bewusst

sind, erscheint ein solches Unternehmen ungerechtfertigt.

Im Zuge dieser Überlegungen wurde mir schließlich bewusst, dass mein Vorgehen für den Aspekt der Systematik von Problemlösungen nach GTM nicht zulässig war, weil ich versucht hatte, etwas in den Daten zu sehen, von dem ich überzeugt war, dass es vorhanden sein sollte. Wie in Kapitel 3.2 gesagt sollen ja aber die Erkenntnisse aus den Daten kommen, da die Theorie am Ende sonst nicht gegrounded sein kann. Ich war also auf dem Weg die Daten zu forcen (3.2.5). Also trat ich einen Schritt zurück und überdachte meinen Ansatz.
Nun stand ich also wieder am Anfang bei der Frage

> Gibt es eine *Systematik* dahinter wie die PP-Partner gemeinsam Wissen produzieren?

Im Sinne der GT entschloss ich mich dazu die transkribierten Daten unvoreingenommen graphisch zu repräsentieren, ohne dabei etwas in die Daten zu interpretieren was nicht intersubjektiv nachvollziehbar dort vorhanden ist. Dies habe ich vorgenommen, in dem ich ein Diagramm erstellt habe, in dem die x-Achse die Sitzungszeit repräsentiert und die y-Achse die aktuell diskutierten Themen der Sitzung darstellt. Dabei kommt dem y-Achsenabschnitt natürlich keinerlei Bedeutung zu, er stellt lediglich die zeitliche Abfolge der besprochenen Themen des Paares dar. Diese Themen habe ich versucht aus den transkribierten Sätzen zu gewinnen, in dem ich ein zentrales Wort (manchmal unter Einbeziehung von Beschreibungen) als Thema eines Satzes festgelegt habe. Oft fallen diese zentralen Worte mit dem grammatikalischen Objekt des Satzes zusammen. Falls das Objekt eines Satzes nicht die primäre Itention des Satzes ausdrückt (beispielsweise durch non-verbale Kommunikation, Ironie oder ähnliches) habe ich versucht die primäre Itention des Satzes durch paraphrasieren herauszubekommen.
Ein solches Diagramm ist für das oben aufgezeigte Beispielszenario in Abbildung 15 gezeigt.
Die Erkenntnis, die sich aus Abbildung 15 ziehen lässt ist jedoch vor allem, dass sich exakte Themenwiederholungen nur selten ereignet haben, denn insgesamt kann man ein tendenziell geradlinig fortlaufendes Gespräch erkennen. Dies mag verwundern wenn man sich das Beispielszenario vor Augen führt, denn auf mich machte es keinen derartig stringenten Eindruck. Ich zog aus dieser Abbildung den Schluss, dass eine Betrachtung auf solch niedrigem Niveau vermutlich zu detailliert ist und ich die Themen vermutlich zu Themengruppen zusammenfassen sollte um so mehr darüber herausfinden zu können wie Paarprogrammierer gemeinsam Wissen produzieren. Diesen Ansatz habe ich dann auch weiterverfolgt.
Zu der Einteilung der Themengruppen ist natürlich anzumerken, dass die gewählte Einteilung sicher einen Einfluss auf das Ergebnis haben wird. Deswegen bemühte ich mich diese Gruppen so objektiv wie möglich einzuteilen. Dies tat

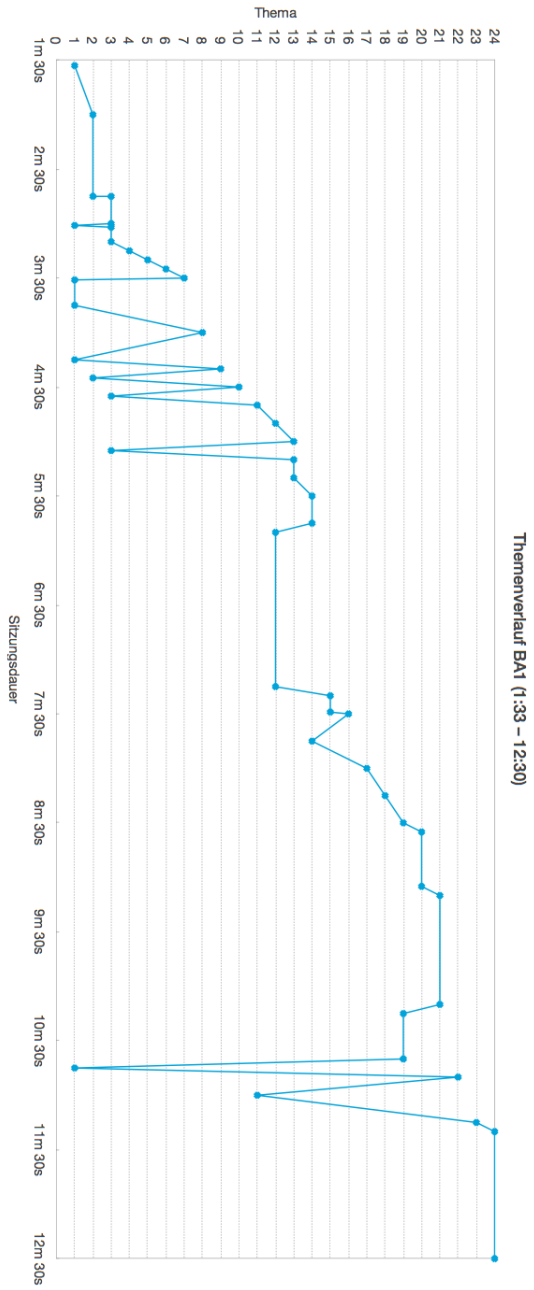

Abbildung 15: Thematischer Gesprächsverlauf des Szenarios aus BA1 – (1:33-12:30). Die x-Achse zeigt die Sitzungsdauer. Die y-Achse repräsentiert die aktuell diskutierten Themen der Sitzung. Dabei kommt dem y-Achsenabschnitt keinerlei Bedeutung zu, er stellt lediglich die zeitliche Abfolge der besprochenen Themen des Paares dar.

ich in dem ich lediglich die Teilaspekte des Problems die die Entwickler selber nennen etwas generalisiert habe zu den folgenden Kategorien:

1. **Effektive Problembild-Äußerung: cURL could not connect**
   Diese Kategorie fasst Äußerungen zusammen, die das effektive Problembild, dass sich cURL nicht verbinden kann, zusammen.

2. **Alternativen**
   Diese Kategorie fasst Äußerungen zusammen, die Alternativen zur aktuell versuchten Vorgehensweise aufzeichnen.

3. **Rechner-Bedienung**
   Diese Kategorie fasst Äußerungen zusammen, die sich auf die Bedienung des in der Sitzung vorhandenen Rechners beziehen.

4. **Tool**
   Unter dieser Kategorie werden Äußerungen zusammengefasst, die sich auf die Benutzung oder das Vorhandensein von gewissen Software Werkzeugen beziehen.

5. **URL & Skript**
   Diese Kategorie fasst Äußerungen zusammen, die sich auf die angegebene URL oder dem Pfad zur bearbeiteten Datei beziehen. Damit ist das Verifizieren, ob die bearbeitete Datei wirklich die in der IDE geöffnete Datei ist, mit eingeschlossen. Ebenso wie Äußerungen, die das Skript selber betreffen.

6. **Zugriff**
   Diese Kategorie fasst Äußerungen zusammen, die sich auf Zugriffsrechte beziehen, also beispielsweise den klassischen Zugriffsrechten auf eine Datei, ebenso wie Firewall-Einstellungen.

7. **Konfiguration**
   Unter dieser Kategorie werden Äußerungen subsummiert, die sich auf Einstellungen innerhalb der vorhandenen Codebasis beziehen, also beispielsweise das Umsetzen von Werten, das Modifizieren von (Header)-Ausgaben oder Ähnlichem.

8. **IDE (Integrated Development Environment**)
   Unter dieser Kategorie werden Äußerungen zusammengefasst, die sich auf die IDE beziehen, also beispielsweise das Einstellen der Schriftgröße, der assoziierte Editor für eine Quelltextdatei und das Erstellen von Arbeitsbereichen (working sets).

Die Kategorisierung (inkl. der Nummerierung) dieses Schemas brachte den Gesprächsverlauf hervor, der in Abbildung 16 dargestellt ist.

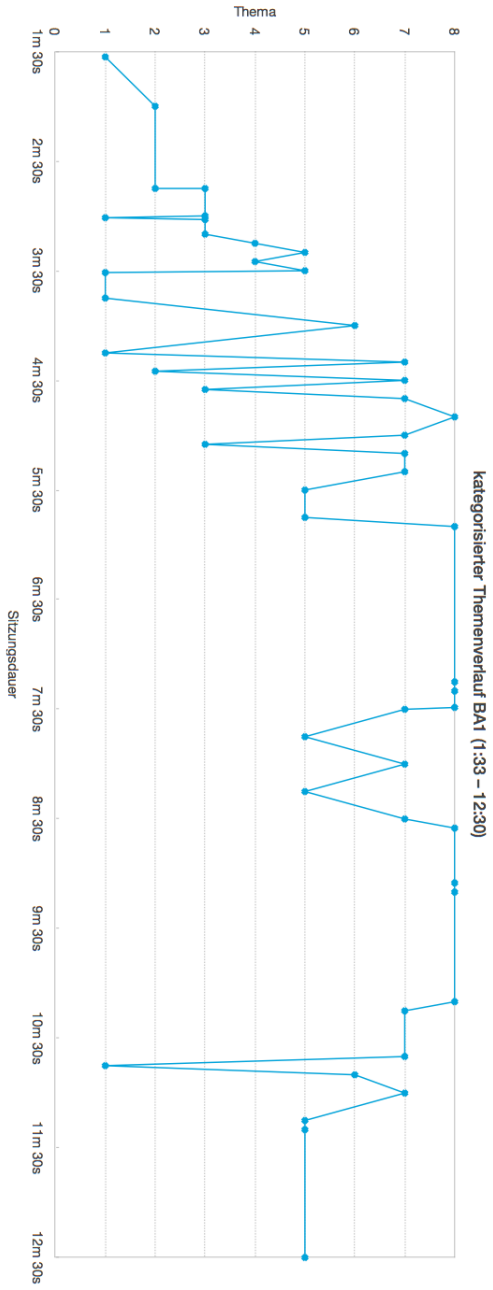

Abbildung 16: Kategorisierter Gesprächsverlauf des Szenarios aus BA1 – (1:33-12:30). Die x-Achse zeigt unverändert die Sitzungsdauer. Die y-Achse repräsentiert die kategorisierten Themen der Sitzung.

In Abbildung 16 sind wiederkehrend referenzierte Themenkategorien zu erkennen. Zum Beispiel die Themenkategorie *URL & Skript* wird mehrfach referenziert, ebenso wie die Kategorie *Konfiguration*. Zwischen diesen beiden Kategorien finden mehrfache (teilweise sogar direkte) Wechsel statt (Minute 7:30 – 8:30 in Abbildung 16). Ich vermute, dass eine systematische Lösung eines Problems wenige Rücksprünge auf bereits besprochene Themenkategorien aufweist, da die verschiedenen Aspekte vollständig geklärt sind bevor die Entwickler an einer neuen Stelle gedanklich ansetzen. Folglich würde ich die vorliegende Klärung aus Abbildung 16 also als eher ungeordnet einstufen.

An dieser Stelle seien einige Anmerkungen gemacht, hinsichtlicher welcher Aspekte sich ein solches Diagramm weiterhin untersuchen ließe abgesehen vom ersten Aspekt, der soeben durchgeführt wurde:

- **Sprunghaftigkeit zwischen Themenkategorien**
  Mit der Sprunghaftigkeit von Themenkategorien ist der soeben untersuchte Aspekt gemeint, ob zwischen gewissen Themenkategorien hin und her gesprungen wird, oder ob eine Themenkategorie nach einmaliger Referenzierung nicht wiederkehrt.

- **Reihenfolge der Themenkategoriewechsel**
  Ein weiterer Aspekt der an der vorliegenden Abbildung untersucht werden könnte, ist die Reihenfolge der Themenwechsel, also in etwa ob auf gewisse Themenkategorien stets eine andere folgt, oder ob keinerlei Zusammenhang erkennbar ist.

- **Entsprechen Themenkategoriewechsel einem Unteraufruf**
  An der vorliegenden Abbildung ließe sich ebenfalls untersuchen ob die Entwickler wenn sie von einer Themenkategorie zur nächsten gesprungen sind und diese dann beendet haben, zu der ursprünglichen Themenkategorie zurückkehren oder nicht. Dies ist vergleichbar mit einem Unteraufruf (oder eventuell sogar mehreren Unteraufrufen) einer Funktion in der Programmierung.

- **zeitliche Verteilung zwischen den Themenkategorien**
  Zuletzt ließe sich die vorliegende Abbildung auch darauf hin untersuchen welcher Themenkategorie wie viel Gesprächszeit zukommt.Eventuell lassen sich auch über diesen Aspekt Einsichten gewinnen.

Im Folgenden wird ein anderes Szenario aus dem Datensatz analysiert werden. Es handelt sich um die Sitzung **CA2.2 – 32:57 - 36:36**.
Auch hier wird bevor die Transkription des Szenarios folgt eine Kurzbeschreibung der Situation angegeben, damit der Rahmen in dem der Szenario stattfindet verständlich wird:

Das Ziel der Sitzung besteht darin eine neue Funktionalität für ein bereits bestehendes Geoinformationssystem zu implementieren. Die zu entwickelnde Funktionalität ist sogenannte „virtuelle Attribute“ in einer GUI des Systems hinzuzufügen. Um dies erreichen zu können muss eine Klasse „TableModel“ erweitert werden. Das gesamte System ist in Java implementiert. Der Einstiegspunkt für das vorgestellte Szenario ist als die Entwickler bereits ein Refactoring[3] des vorhandenen Codes durchgeführt und getestet haben. Es ist ebenfalls damit begonnen worden die gewünschte neue Funktionalität zu implementieren und die Änderungen zu testen. Es folgt eine Phase in der das Paar über das Design der vorliegenden Codestelle diskutiert:

C1	„Wir können überlegen ob wir das in zwei TableModels aufspalten“ (33.05)
C2	„Also du willst dem TableModel hier noch einmal ein TableModel drüber flanschen“ (33.09)
C1	„Oder nicht dem, sondern zwei separate die wir dann irgendwie miteinander verbinden“ (33.14) Da gibt es sicher schon irgendwelche Klassen mit denen man aus zwei TableModels eins machen kann“ (33.17)
C2	„aaaaahhhhhhh <*ist nicht überzeugt*>“
C1	„Würde bedeuten, dass wir die Zuständigkeiten der beiden TableModels ein bisschen besser aufspalten“ (33.27)
C2	„Also ich würde... Lass uns mal das TableModel angucken wie das arbeitet, wie das die Werte abholt (...) Also ich bin der Meinung, dass das TableModel gar nicht wissen darf oder sollte, dass das gegen virtuelle oder nicht virtuelle Spalten arbeitet“ (33.52)
C1	„Okaaaay, das ist die Abstraktion an die ich vorher gedacht hatte“ (33.57)
C2	„Der sollte im Grunde aus der Konfiguration noch einen Value Provider oder was weiß ich abholen“ (34.05)
C1	„Hast du im Prinzip nicht Unrecht ja - das ist die Abstraktion an die ich vorher gedacht hatte. (..) Das heißt, ok“ (34.16)

[3]Refactoring ist das Umstrukturieren von Quellcode ohne dessen beobachtbares Verhalten zu verändern. Es dient zumeist der besseren Lesbarkeit und Verständlichkeit des vorliegenden Codes.

C2	„Dass wir die Attributkonfiguration dann noch um so einen Value Provider erweitern und der gibt dann immer das FeatureProxy rein um die Werte abzuholen und dahinter passiert dann der Rest.“ (34.40)
C1	„Mmmmmmm <*überlegt*> (∼∼) featureProxySet („ „ „)“
C2	„Das ist ja das blöde. Das featureProxySet implementiert halt mehr oder weniger andere TableModels. Das könnte man herausziehen dann.“ (34.53)
C1	„(„ „)Ich bin mir nicht sicher ob ich es da herausziehen würde („ „ ,)“ (35.08)
C2	„Ich würde nur noch das featureProxy abholen aus dem Set und dann (...) gegen etwas anderes arbeiten“ (35.16)
C1	„(..) Wie jetzt?“ (35.18)
C2	„Also ich würde, im Endeffekt würde ich das so umstellen, dass das featureProxySet diese TableModel-Eigenschaften verliert. Dass du aus dem featureProxySet nur noch dieses feature abholst.“ (35.31-)
C1	„Hm <*Zustimmung*> (...)“ (35.34)
C2	„Und zusätzlich aus dem“ (35.36)
C1	„Das ist übrigens keine TableModel-Eigenschaft übrigens. Das ist das mit den Namen“ (35.40)
C2	„Ja, also getValue(). Der hat aber auch getValueAdd - mit Index kannst du auch“ (35.46)
C1	„Hm ich weiß. Das ist furchtbar“ (35.47)
C2	„Du hast das ja hier mit RowIndex“
C1	„Hm <*Zustimmung*> (.) Die Frage ist ob ich mich da jetzt rantrauen will, ob wir uns da rantrauen wollen“ (35.55)
C2	„Ich würde mich da schon gerne rantrauen wollen weil das eine Chance ist das leichtgewichtiger zu machen über diesen Arbeitspunkt“ (36.01)
C1	„Dann sollten wir das zuerst machen oder danach, aber nicht gleichzeitig“ (36.06)
C2	„Hm <*Zustimmung*>“ (36.10)
C1	„Ich fürchte halt, dass wir bei dem Arbeitspunkt so wie so die Zeit überschreiten werden und das relativ komplex wird (....) vielleicht dass man es danach macht? Ich bin mir da aber auch nicht sicher“ (36.25)
C2	„Also ich würde das, hm, weil wenn wir das danach machen wollen, dann passiert das nicht.“ (36.31)

C1 | „Ok (...) gut“ (36.36)

Auch dieses Szenario wird in Abbildung 17 dargestellt. Es ist mit den gleichen Regeln erstellt worden wie auch Abbildung 15. Die x-Achse repräsentiert also die Sitzungszeit, während die y-Achse die aktuell diskutierten Themen der Sitzung darstellt. Auch hier wurden die Themen an Hand der oben beschriebenen Methode bestimmt. Dem y-Achsenabschnitt kommt also erneut keinerlei Bedeutung zu, er stellt lediglich die zeitliche Abfolge der besprochenen Themen des Paares dar.
Abbbildung 17 zeigt einen relativ geradlinigen Gesprächsverlauf auf, da thematische Wiederholungen nur selten vorkommen. Auch die in Abbildung 17 aufgezeigten Themen werden wie im vorherigen Szenario zusammengefasst und zwar in die folgenden Kategorien:

1. **Aufspaltung des TableModels**
   Diese Kategorie fasst Äußerungen zusammen, die die Idee verbalisieren das vorhandene TableModel in zwei unterschiedliche TableModels aufzuspalten. Darin mit eingeschlossen sind konkrete Vorschläge wie dies implementiert werden könnte.

2. **Abstraktionsschicht**
   Unter dieser Kategorie werden Äußerungen subsummiert, die das Einziehen einer zusätzlichen Abstraktionsschicht referenzieren.

3. **TableModel Eigenschaften**
   Diese Kategorie fasst Äußerungen zusammen, die zum Thema haben was TableModel Eigenschaften sind und was nicht.

4. **Reihenfolge der Bearbeitung**
   Diese Kategorie fasst Äußerungen zusammen, die die Reihenfolge der Bearbeitung für zukünftige Arbeitsschritte referenzieren.

Der aus der soeben beschriebenen Kategorisierung resultierende Gesprächsverlauf ist in Abbildung 18 dargestellt.
In Abbildung 18 ist ein kategorisierter Gesprächsverlauf zu sehen, der wenig sprunghaft zu sein scheint. Themenwechsel werden (mit Ausnahme von der Stelle bei 34:05) nicht rückgängig gemacht. Einer der Gründe dafür könnte sein, dass die Themen beendet worden sind bevor zu einem nächsten Thema gesprungen wird. Den vorliegenden Verlauf klassifiziere ich also folglich als eine systematische Problemlösungsstrategie.

In diesem Kapitel ist also eine Methode vorgestellt an Hand derer sich eine systematische Problemlösungsstrategie von einer scheinbar unsystematischen Problemlösungsstrategie unterscheiden lässt.

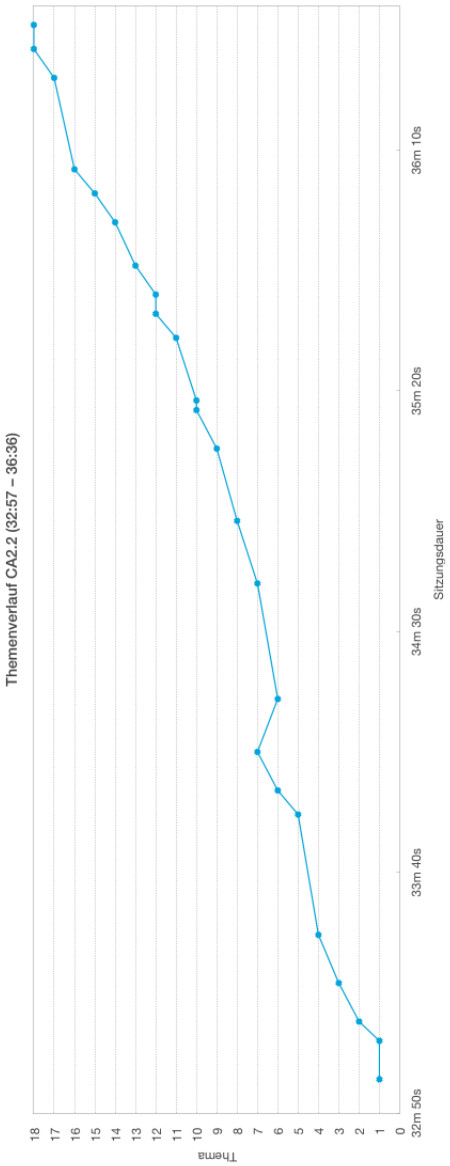

Abbildung 17: Gesprächsverlauf des Szenarios aus Sitzung CA2.2 – (32:57 - 36:36). Die Abszisse zeigt die Sitzungsdauer. Die Ordinate repräsentiert die aktuell diskutierten Themen der Sitzung. Dabei kommt dem y-Achsenabschnitt keinerlei Bedeutung zu, er stellt lediglich die zeitliche Abfolge der besprochenen Themen des Paares dar.

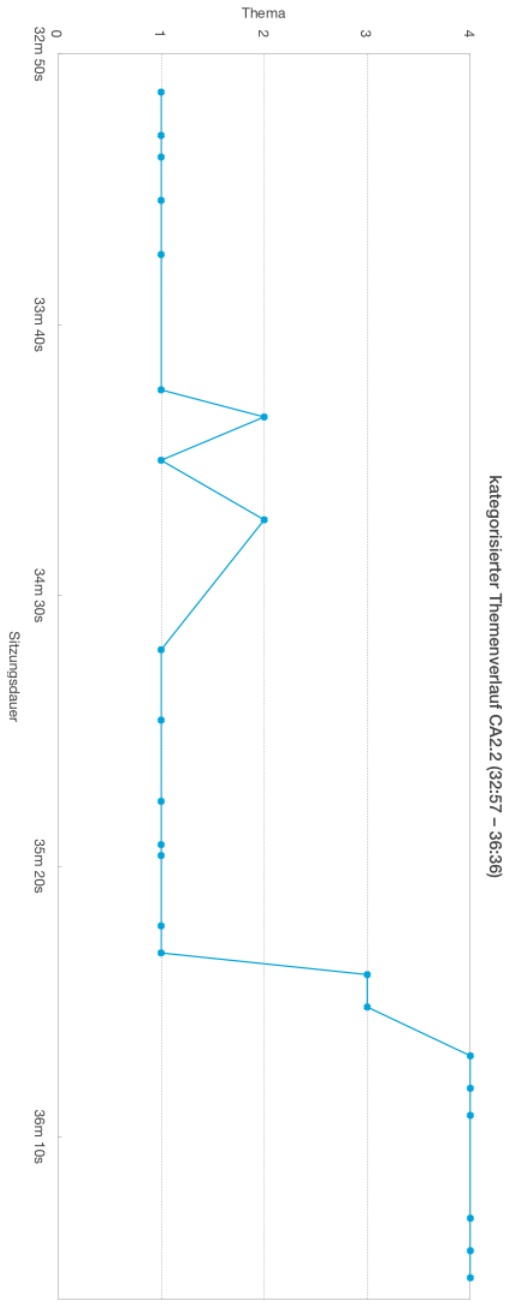

Abbildung 18: Kategorisierter Gesprächsverlauf des Szenarios aus Sitzung CA2.2 – (32:57 - 36:36). Die Abszisse zeigt die Sitzungsdauer. Die Ordinate repräsentiert die kategorisierten Themen der Sitzung.

6.7 Sonstige Konzepte

In diesem Abschnitt sollen Konzepte angeführt werden, die ich zu Beginn meiner Studie an Phänomene annotiert hatte, im weiteren Verlauf dieser Arbeit aber wieder verworfen habe, welche mir aber dennoch erwähnenswert erscheinen, da ich vermute, dass auch sie Teilaspekte der gemeinsamen Wissensproduktion konzeptionalisieren. Nach der Nennung und der Beschreibung eines solchen Konzepts wird ebenfalls der Grund für dessen Aufgabe genannt werden.

- **Gedankenschritt**
  Ein Gedankenschritt ist ein Konzept, das den kognitiven Vorgang eines der beiden Partner kodiert. Es fügt sich zwischen eine Problembild-Äußerung und eine Lösungsbild-Äußerung ein, da ja offensichtlich ein kognitiver Prozess bei demjenigen Entwickler stattgefunden haben muss, der als Konsequenz ja in der Lage ist einen Lösungsvorschlag verbalisieren zu können. Die Problematik bei diesem Konzept ist, dass es in einer behavioristischen Sichtweise auf die Daten (Kapitel 3.2.6) schwierig ist zu lokalisieren, da ja der Prozess an sich nicht sichtbar ist. Eine Möglichkeit dieses Problem zu lösen wäre eventuell gewesen einen Gedankenschritt durch sichtbare Phänomene derart „umzingeln“ zu können, dass es trotz mangelnder Operationalisierbarkeit intersubjektiv nachvollziehbar geworden wäre. Davon habe ich jedoch Abstand genommen, da die Konsequenz eines Gedankenschritts wesentlich einfacher zu operationalisieren ist, nämlich eine Lösungsbild-Äußerung (Kapitel 6.2)

- **Verbindung zwischen den Partnern**
  Die Verbindung zwischen zwei Partnern ist ein Konzept, dass die gelegentlich auftretenden Äußerungen konzeptionalisiert, die dem Zweck dienen nach dem aktuellen kognitiven Zustand des Partners zu fragen, also beispielsweise:

  „[...] weißt du was ich meine?“ (**BA1 1:26:18**)
  „[...] kannst du mir folgen gerade?“ (**BA1 49:20**)
  „Guckst du noch oder <*kann ich weiterblättern*>?“ (**CA3.1 8:41**)
  „Das hast du vorher gemeint oder?“ (**CA4.1 7:27**)

  Ich hatte den Eindruck, dass solche Äußerungen bei eng kooperierenden Paaren oft verbalisiert werden, während sie bei scheinbar weniger eng kooperierenden Paaren eher selten zu finden sind. Dies ist aber in der vorliegenden Arbeit nicht näher untersucht worden, da es dem Prozess der

gemeinsamen Wissensproduktion nicht in wesentlichem Maße beschreiben kann. Ich vermute jedoch, dass dies ein Aspekt sein könnte, der mit der Qualität der gemeinsamen Wissensproduktion korreliert.

7 Validität und Güte

In diesem Kapitel sollen nun die Güte und die Gefahren für die Validität der Ergebnisse aus Kapitel 6 dargestellt werden. Da es sich ja bei diesen Ergebnisse um Ergebnisse einer qualitativen Forschungsmethode handelt, können hier nicht einfach die Maßstäbe quantitativer Forschung übernommen werden. Mayring gibt für die Beurteilung qualitativer Forschung sechs allgemeine Gütekriterien an ([28], S. 140):

1. **Verfahrensdokumentation:** Quantitative Forschung kann sich in vielen Fällen eines standardisierten „Werkzeugkastens“ bedienen um nachweislich valide Ergebnisse zu erzielen. Bei qualitativer Forschung hingegen ist es notwendig, dass das angewandte Vorgehen „bis ins Detail dokumentiert [wird]“
2. **Argumentative Interpretationsabsicherung:** Interpretationen gehören zu qualitativer Forschung (Kapitel 3.2). Problematischerweise lassen sich solche aber „nicht beweisen [...] [sondern müssen] argumentativ begründet werden“. Um dieses zu tun sollen Interpretationen erklärt und Alternativdeutungen überprüft werden.
3. **Regelgeleitetheit:** Qualitative Forschung ist kein Synonym für „völlig unsystematisches Vorgehen“. Eben drum muss sie sich auch an bestimmte Verfahrensregeln halten und „systematisch ihr Material bearbeiten.“ Diese Regelgeleitetheit muss nachvollziehbar gemacht werden.
4. **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.“
5. **Kommunikative Validierung:** „Die Gültigkeit der Ergebnisse, der Interpretationen kann man auch dadurch überprüfen, indem man sie den Beforschten nochmals vorlegt, mit ihnen diskutiert.“
6. **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.“

Wenn man die vorliegenden Arbeit bezüglich der soeben genannten Punkte überprüft, kann man zu den folgenden Ergebnissen kommen:

- Das Analyseverfahren ist genau dokumentiert worden. Es wurde sowohl die Methodik (Kapitel 3.2) als auch deren Erweiterungen (Kapitel 3.2.6) dokumentiert. Ferner wurde das Datenmaterial und die Datenerhebungsmethode dargelegt (Kapitel 4). Somit ist auch festzustellen, dass die Regelgeleitetheit nachvollziehbar gemacht worden sein sollte.
- Interpretationen von Phänomenen wurden hauptsächlich dazu benutzt um zu Klassifizieren. Diese Klassifizierungen wurden versucht durch Definitionen und Beispiele intersubjektiv nachvollziehbar zu machen. Zur weiteren Interpretationsabsicherung wurde zu jedem Konzept auch dessen Überschneidungen mit der Basisschicht aufgezeigt. Ferner sollte die Dimensionalisierung eines jeden Konzepts als ein Versuch die Extension eines Konzepts aufzuzeigen betrachtet werden.
- Wie in Kapitel 4 dargestellt sind alle von mir analysierten Paarprogrammierungssitzungen im professionellen Umfeld entstanden. Ferner handelte es sich bei den betrachteten Sitzungen um reale Aufgaben, die im Unternehmen ohnehin zur Bearbeitung anstanden. Darüber hinaus wurden die Entwickler in ihrer natürlichen Arbeitsumgebung gefilmt. Dies kann nur bedeuten, dass die Nähe zum Gegenstand als hoch beurteilt werden sollte.

- Auf eine kommunikative Validierung wurde aus zeitlichen Gründen und der Kürze der Erstellungsdauer dieser Arbeit verzichtet. Ferner würde eine solche Validierung auch den Rahmen einer Masterarbeit sprengen.
- Eine Triangulierung hat wenn dann nur in sehr leichter Form stattgefunden, nämlich in der Form, dass verschiedene Paarprogrammierungssitzungen betrachtet wurden. Dies ist mit einer Datentriangulation zu bezeichnen. Eine wesentlich größere Triangulation erscheint mir aber auch den Rahmen der hier vorliegenden Arbeit ein zu ambitioniertes Ziel zu sein.

Natürlich muss an dieser Stelle noch einmal explizit darauf hingewiesen werden, dass die hier präsentierten Ergebnisse nicht vollständig und lückenlos sein können. Dies ist mit der Bearbeitungsdauer und der Komplexität des Vorgehens zu begründen. Zielsetzung dieser Arbeit war es ja aber auch einen Teil zu dem aktuellen Promotionsvorhaben in der AGSE beizutragen.

8 Zusammenfassung & Ausblick

Die vorliegenden Arbeit hatte zum Ziel die gemeinsame Wissensproduktion in der Paarprogrammierung auf konzeptioneller Ebene zu erhellen. Dazu wurde ein qualitativer Forschungsstil angewendet, nämlich die Grounded Theory Methode.
Die folgenden Ergebnisse hat diese Arbeit zu Tage gebracht:
Es wurden zunächst vier Klassen von Konzepten vorgestellt: Problembild-Äußerungen, Lösungsbild-Äußerungen, Relevanz-Äußerungen und TOPIC-Veränderungen.
Problembild-Äußerungen wurden als verbale Äußerungen eines Entwicklers definiert, die ein Hindernis für das Voranschreiten im Prozess darstellen. Diese konnten im Verlauf der Arbeit untergliedert werden in effektive Problembild-Äußerungen – also solche die ein bereits zu Tage getretenes Problem adressieren – und potentielle Problembild-Äußerungen, die ein Problem adressieren, dessen Auftreten noch nicht sicher ist.
Lösungsbild-Äußerungen wurden als solche verbalen Äußerungen eines Entwicklers des Paares definiert, die eine Lösungs(-idee) eines zuvor geäußerten Problembilds referenzieren. Diese wurden im Verlauf dieser Arbeit unterteilt in solche die eine konkrete Lösung vorschlagen und solche, die ein Szenario skizzieren.
Als *Relevanz-Äußerungen* wurden verbale Äußerungen eines Entwicklers definiert, die die persönliche Relevanzeinschätzung bezüglich des aktuell besprochenen Themas angeben. Diese wurden untergliedert in solche die dem aktuellen Thema eine hohe Relevanz zuweisen und solche, die diesem eine niedrige Relevanz geben.
Zuletzt wurden *TOPIC-Veränderungen* als verbale Äußerungen die das aktuell besprochene Thema verändern definiert. TOPIC-Veränderungen wurden in solche unterteilt, die die Extension (also den Umfang aller Dinge, die das aktuell besprochene Thema referenzieren könnte) verringern, also einen thematischen Fokus legen und in solche die die Extension vergrößern, thematisch also zu einer Divergenz führen.
Abbildung 19 zeigt die vier Konzepte inklusive ihrer Unterkategorien in graphischer Form.
Bis zu einem gewissen Punkt sind die Konzepte (trotz ihrer Einbettung in reale Paarprogrammierungssitzungen) tendenziell isoliert betrachtet worden. Aufschlussreich ist jedoch das Zusammenspiel dieser vier Konzepte, welches sich wie folgt in einer Geschichte (nach GTM Methodologie) erzählen lässt:

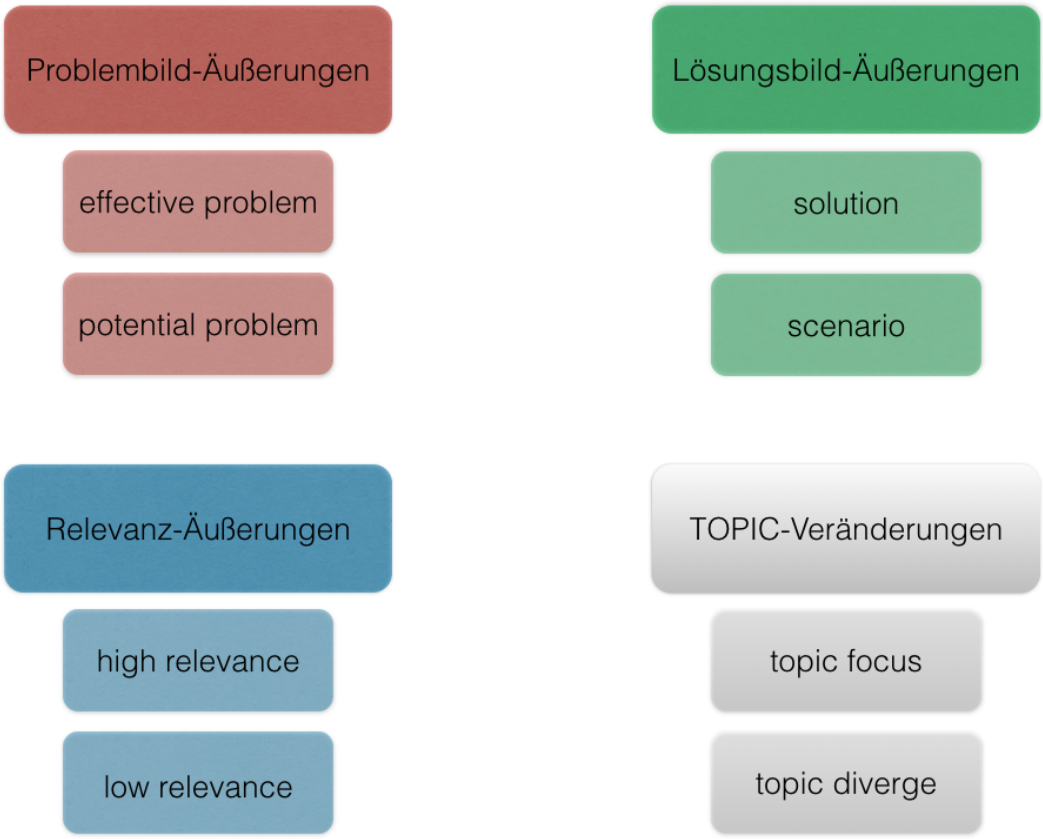

Abbildung 19: Zusammenfassung der vier in der vorliegenden Arbeit hergeleiteten Konzepte gemeinsamer Wissensproduktion. Die Kategorien von Konzepten sind jeweils in höherer Farbdeckung dargestellt als die dazugehörigen Konzepte selber.

Problembild-Äußerungen fungieren oftmals als thematische Steuerung der Diskussion. Ein geäußertes effektives Problembild führt oft dazu, dass dieses in der Folge in den Mittelpunkt der Diskussion rückt. Dies entspricht einem thematischen Fokus. Lösungsbild-Äußerungen fungieren offensichtlicherweise als Versuch ein zuvor fokussiertes Problem zu lösen. Problembild-Äußerungen und damit einhergehende thematische Fokussierungen können jedoch auch abgelehnt werden. Dies ist besonders dann oft zu beobachten wenn es sich bei der Problembild-Äußerung um eine die ein potentielles Problem referenziert handelt. Es kann also geschlussfolgert werden, dass Relevanz-Äußerungen auf die thematischen Veränderungen innerhalb der gemeinsamen Wissensproduktion Einfluss nehmen kön-

nen.
Außerdem ist zu beobachten, dass potentielle Problembild-Äußerungen mit der Skizzierung eines Szenarios beantwortet werden können. Ich vermute, dass diese unter anderem dazu dienen dass potentielle Problembild zu explizieren um auf diese Weise eine Lösung vorschlagen zu können.

Auf diesem Zusammenspiel basierend ist die Überlegung angestellt worden, ob sich eine Systematik hinter der gemeinsamen Wissensproduktion erkennen lässt. Diese war zuerst auf das Erkennen von algorithmischen Strukturen während der Produktionsphasen ausgerichtet. Im weiteren Verlauf ist von dieser Idee jedoch Abstand genommen worden, da es sie zu einem Forcen der Daten geführt hätte. Deshalb ist eine Methode vorgestellt worden, die auf transkribierten Szenarien aus den Daten basiert und mittels Kategorisierung der Thematiken eine systematische Problemlösungsstrategie von einer weniger systematisch anmutenden Strategie unterscheiden kann. Ebenfalls sind weitere Kriterien, die man an Hand der vorgestellten Methode untersuchen könnte, genannt worden. Es ist kritisch angemerkt worden, dass die Wahl der Kategorien natürlich einen Einfluss auf das Ergebnis haben wird.

Diese Ergebnisse wurden im weiteren Verlauf dieser Arbeit auf ihre Validität und Güte hin überprüft und diskutiert. Selbstverständlich können die hier vorgestellten Konzepte keinen Anspruch auf Vollständigkeit erheben. Dies kann sich unter anderem darin äußern, dass auch andere Kombinationen von Verben und den von mir beobachteten Objekten zu finden sind. Dennoch wurde natürlich versucht die vorhandenen Daten möglichst vollständig und sinnvoll zu konzeptualisieren. Dabei wurde natürlich Rücksicht darauf genommen, dass in den Daten nicht vorhandene Phänomene auch nicht konzeptualisiert werden.

Die dargelegten Konzepte sind eine Beschreibung der gemeinsamen Wissensproduktion in der Paarprogrammierung auf hoher konzeptioneller Ebene. In der Dimensionalisierung der einzelnen Konzepte wurden die Eigenschaften mit deren Dimensionen vorgestellt, die ebenfalls eine feingranularere Unterteilung der Konzepte hätte rechtfertigen können. Dies schien aber im Rahmen dieser Arbeit nicht sinnvoll zu sein, da die daraus resultierende Vielzahl von Konzepten sich dann nicht mehr zu einer Geschichte im Sinne der GTM zusammenfügen ließen. Für tiefergehende Studien kann es aber notwendig werden die hier genannten Konzepte an Hand anderer als den von mir gewählten Eigenschaften bzw. an Hand mehrerer solcher Eigenschaften zu unterscheiden und somit in anderer Granularität zu konzeptionalisieren.

Zur Tragweite der hier dargelegten Ergebnisse seien die folgenden Bemerkungen gemacht:
Die Erkenntnisse dieser Arbeit sind nicht alleine für den Softwareprozess Extreme Programming von Bedeutung, sondern vielmehr für alle agilen Prozesse. Darüber hinaus sogar nicht nur für agilen Prozesse, denn im Kern geht es bei Paarprogrammierung ja um das gemeinsame Arbeiten an Softwareartefakten in Zweiergruppen. Anekdotische Berichte weisen darauf hin, dass Paarprogrammierung einerseits bedeutend länger praktiziert wird als es unter diesem Namen bekannt ist und andererseits wesentlich häufiger praktiziert wird als explizit angegeben wird [34]. Dies ist dadurch zu begründen, dass im alltäglichen Arbeitsleben eines Softwareentwicklers schnell Paarprogrammierungskonstellationen auftreten, beispielsweise wenn ein Mitarbeiter seinen Kollegen zu einer spezifischen Stelle im Programmcode etwas fragt und der Kollege sich „kurz dazusetzt“.

Da wie eingangs der vorliegenden Arbeit beschrieben als Ziel der Forschung Verhaltensmuster für Paarprogrammierer stehen, möchte ich abschließend einen in weiteren Arbeiten zu überprüfenden Vorschlag für eine Verhaltensweise, die der gemeinsamen Wissensproduktion in der Paarprogrammierung zuträglich zu sein scheint, machen:

> Es scheint sinnvoll zu sein wenn die Entwickler die thematische Entwicklung ihrer Gespräche (insbesondere während sie Probleme lösen) im Hinterkopf bewahren und beachten, denn es scheint der gemeinsamen Wissensproduktion zuträglich zu sein Themensprünge nur dann zu machen, wenn das aktuelle Thema entweder abgeschlossen ist, oder aber das neue Thema einen Unteraufruf des vorherigen Themas darstellt, also nach Beendigung des dazwischengeschobenen Themas wieder zum alten Thema zurückgekehrt wird.

Um das Gesagte auch graphisch zu veranschaulichen, möchte ich also das Diagramm aus Abbildung 20 als unvorteilhafte kategorisierte thematische Gesprächsentwicklung klassifizieren, also als Anti-Pattern.
Die Diagramme aus Abbildung 21 sehe ich als positive kategorisierte thematische Gesprächsentwicklung an, also als ein positives Muster.

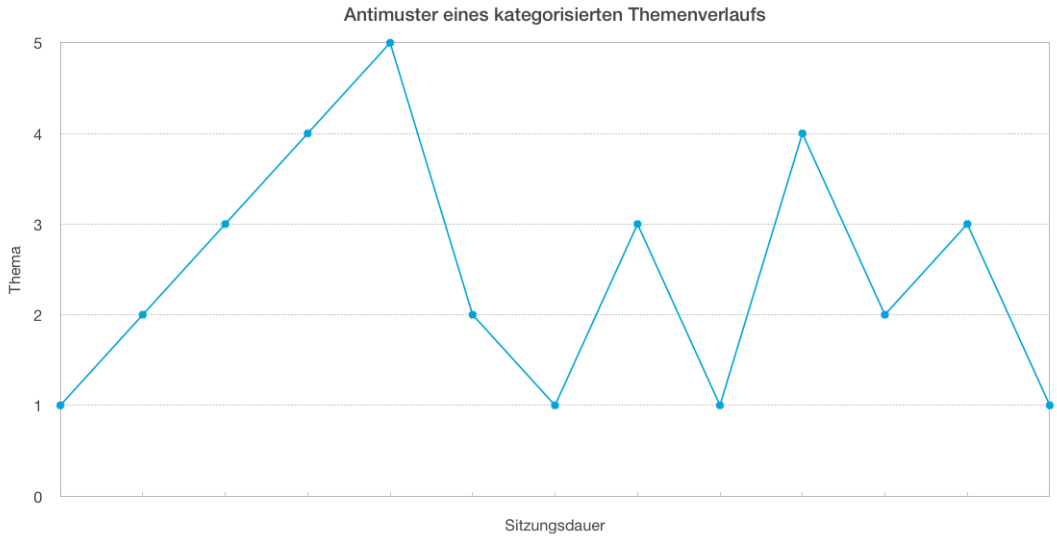

Abbildung 20: Negatives Beispiel eines kategorisierten thematischen Gesprächsverlaufs. Dies ist durch die Sprunghaftigkeit der besprochenen Themen zu begründen, als auch durch den Verlauf der Themen: nach Beendigung eines Themas wird nicht zum ursprünglichen Thema zurückgesprungen. Der Verlauf solcher Themenwechsel stellt also keinen Unteraufruf dar.

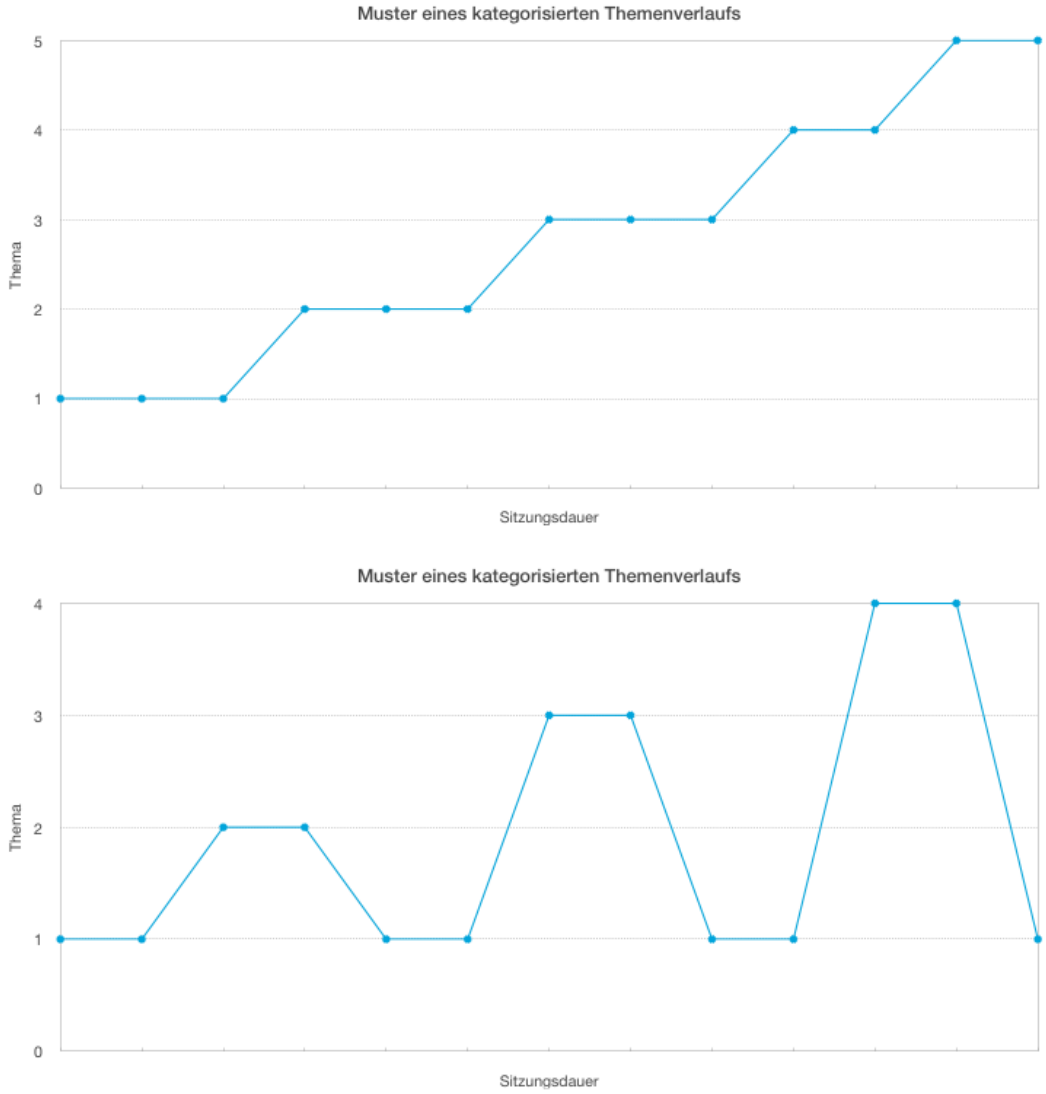

Abbildung 21: Positive Beispiele eines kategorisierten thematischen Gesprächsverlaufs. In der oberen Abbildung ist dies durch den geradlinigen Verlauf und der geringen Sprunghaftigkeit der besprochenen Themen zu begründen. In der unteren Abbildung ist die Abfolge der Themenwechsel als positiv zu beschreiben, da nach Beendigung eines Themas zum ursprünglichen Thema zurückgekehrt wird, die Themenwechsel entsprechen also einem Unteraufruf.

Zum Abschluss dieser Arbeit sei noch ein Ausblick auf weitere mögliche Arbeiten, die mit dem Thema der vorliegenden Arbeit verwandt sind, angegeben: In der vorliegenden Arbeit konnte ein Vorgehen aufgezeigt werden, dass ein erkennbar systematisches Problemlösen von einem scheinbar unsystematischen Vorgehen unterscheiden kann. Ein offen gebliebener Aspekt dieser Betrachtung der sicherlich weitere Aufmerksamkeit verdient ist, ob man innerhalb der systematischen Vorgehensweisen nicht doch algorithmische Vorgehensweisen erkennen kann wie in etwa eine Breiten- oder Tiefensuche.
Weiterhin ist offen geblieben, ob sich hinter der scheinbar unsystematischen Problemlösungsweise nicht doch komplexere Strukturen verbergen, die im Rahmen dieser Arbeit unentdeckt geblieben sind.
Ein weiterer in dieser Arbeit angesprochener Aspekt der ebenfalls mehr Aufmerksamkeit verdient als im Rahmen dieser Arbeit möglich war, sind die in Kapitel 6.6 angeführten weiteren Aspekte nach denen man die vorgestellten Diagramme untersuchen könnte. Insbesondere die Untersuchung ob thematische Veränderungen während einer gemeinsamen Wissensproduktionsphase einem Unteraufruf einer Funktion entsprechen, oder ob vermeintlich weniger systematisch thematisch zurückgesprungen wird, erscheint mir aussichtsreich um die Systematik mit der Paarprogrammierer gemeinsam Wissen produzieren tiefergehend verstehen zu können.

Mit der vorliegenden Arbeit hoffe ich einerseits einen weiteren Anteil an der Arbeit der Arbeitsgruppe Software Engineering geleistet zu haben, so wie andererseits an dem momentan laufenden Promotionsvorhaben.

Anhang

A Transkriptionssystem

Für die Transkription von Szenarien wurde von Salinger ein eigenes Transkriptionssystem entwickelt, die so genannte „pragmatische halbinterpretative Transkription“ (PHIT). Dieses Transkriptionssystem basiert auf der „Halbinterpretative Arbeitstranskription“ (HIAT), die sich insbesondere im deutschsprachigen Raum großer Beliebtheit erfreut [19],[20],[21],[34].

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.	

Abbildung 22: Pragmatisches Halbinterpretatives Transkriptionssystem für Paarprogrammierungssitzungen [34]

B Eidesstattliche Erklärung

Ich versichere hiermit an Eides Statt, dass diese Arbeit von niemand anderem als meiner Person verfasst worden ist. Alle verwendeten Hilfsmittel wie Berichte, Bücher, Internetseiten oder ähnliches sind im Literaturverzeichnis angegeben, Zitate aus fremden Arbeiten sind als solche kenntlich gemacht. Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungskommission vorgelegt und auch nicht veröffentlicht.

Berlin, den 3. November 2014

Yannick Gerhard Pascal Schütt

Literatur

[1] *ATLAS.ti - qualitative data analysis*. `http://atlasti.com/de/`, Abruf: 17. Oktober 2014

[2] *Camtasia*. `http://www.techsmith.de/camtasia.html`, Abruf: 17. Oktober 2014

[3] *cURL*. `http://curl.haxx.se`, Abruf: 29. Oktober 2014

[4] *GNU Wget*. `http://www.gnu.org/software/wget/`, Abruf: 14. Oktober 2014

[5] *Introducing JSON*. `http://json.org`, Abruf: 10. Oktober 2014

[6] *PHP*. `http://php.net`, Abruf: 10. Oktober 2014

[7] *Agility counts*. `http://www.economist.com/node/779429`. Version: 2001, Abruf: 25. Oktober 2014

[8] Alheit, P. : *„Grounded Theory": Ein alternativer methodologischer Rahmen für qualitative Forschungsprozesse*. Universität Göttingen, 1999/2000

[9] 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. 65–68

[10] Baheti, P. ; Gehringer, E. ; Stotts, D. : Exploring the Efficacy of Distributed Pair Programming. In: *Extreme Programming and Agile Methods - XP/Agile Universe 2002*, Springer Verlag, 2002

[11] Beck, K. : *Extreme Programming Explained: Embrace Change*. Addison-Wesley Professional, 1999

[12] Beck, K. : *Extreme Programming Explained: Embrace Change*. Addison-Wesley Professional, 2004 (Second Edition)

[13] Canfora, G. ; Cimitile, A. ; Garcia, F. ; Piattini, M. ; Visaggio, C. A.: Evaluating performances of pair designing in industry. In: *Journal of Systems and Software 80* (2007), Nr. 8, S. 1317–1327

[14] 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, 2005

[15] Chong, J. ; Hurlbutt, T. : The Social Dynamics of Pair Programming. In: *Proceedings of the 29th International Conference on Software Engineering*, IEEE Computer Society, 2007, S. 354–363

[16] Cockburn, A. ; Highsmith, J. : *Crystal Clear. A Human Powered Methodology for Small Teams.* Pearson Education, 2005

[17] Cormen, T. H. ; Leiserson, C. E. ; Rivest, R. L. ; Stein, C. : *Introduction to Algorithms.* The MIT Press, 2009 (3rd edition)

[18] 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

[19] Ehlich, K. ; Rehbein, J. : Halbinterpretative Arbeitstranskriptionen (HIAT). In: *Linguistische Berichte 45* (1976), S. 21–41

[20] Ehlich, K. ; Rehbein, J. : Erweiterte halbinterpretative Arbeitstranskriptionen (HIAT2). In: *Linguistische Berichte 59* (1979), S. 51–75

[21] Ehlich, K. ; Rehbein, J. : Zur Notierung nonverbaler Kommunikation für diskursanalytische Zwecke (HIAT2). In: *Methoden der Analyse von Face-To-Face-Situationen* (1979), S. 302–329

[22] Frege, G. : Über Sinn und Bedeutung. In: *Zeitschrift für Philosophie und philosophische Kritik* 100 (1892), S. 25–50

[23] Gamma, E. ; Helm, R. ; Johnson, R. ; Vlissides, J. : *Design Patterns: Elements of Reusable Object-Oriented Software.* Addison-Wesley Professional, 1995

[24] Glaser, B. G. ; Strauss, A. L.: *The Discovery of Grounded Theory. Strategies for Qualitative Research.* Aldine Transaction, 1967

[25] Gumm, H.-P. ; Sommer, M. : *Einführung in die Informatik.* Addison-Wesley, 2011 (9. Auflage)

[26] Jeffries, R. ; Anderson, A. : *Extreme Programming Installed.* Addison-Wesley Longman Publishing, 2000

[27] Kruchten, P. : *The Rational Unified Process. An Introduction.* Addison-Wesley, 2004

[28] Mayring, P. : *Einführung in die qualitative Sozialforschung: Eine Anleitung zu qualitativem Denken.* Beltz Verlag, 2002

[29] McBreen, P. : *Questioning Extreme Programming.* Addison-Wesley Longman Publishing, 2002

[30] 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

[31] Nosek, J. T.: The case for collaborative programming. In: *Communications of the ACM 41* (1998), Nr. 3, S. 105–108

[32] Rumpe, B. : *Modellierung mit UML.* Springer Verlag, 2011 (2. Auflage)

[33] 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

[34] Salinger, S. : *Ein Rahmenwerk für die qualitative Analyse der Paarprogrammierung*, Freie Universität Berlin, Diss., 2013

[35] Salinger, S. ; Zieris, F. ; Prechelt, L. : Liberating Pair Programming Research from the Oppressive Driver/Observer Regime. In: *ICSE 2013*, 2013

[36] 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: *Proceedings of the 3rd International Symposium on Empirical Software Engineering and Measurement*, IEEE Computer Society, 2009, S. 214–225

[37] Schwaber, K. ; Beedle, M. : *Agile Software Development with Scrum.* Prentice Hall, 2002

[38] Strauss, A. L. ; Corbin, J. : *Basics of Qualitative Research: Grounded Theory Procedures and Techniques.* Sage Publications, 1990

[39] Strauss, A. ; Corbin, J. : *Grounded Theory: Grundlagen Qualitativer Sozialforschung.* Psychologie Verlags Union, 1996

[40] Strübing, J. : *Zwei Varianten von Grounded Theory? Zu den methodologischen und methodischen Dierenzen zwischen Barney Glaser und Anselm Strauss.* Vs Verlag, 2011

[41] Vanhanen, J. : *Effects of Pair Programming at the Development Team Level: An Experiment*, Helsinki University of Technology, Diss., 2005

[42] Williams, L. ; Kessler, R. ; Cunningham, W. : Strengthening the Case for Pair Programming. In: *IEEE Software 17* (2000), Nr. 4, S. 19–25

[43] Williams, L. ; Kessler, R. : All I really need to know about pair programming I learned in kindergarten. In: *Communications of the ACM 43* (2000), May, S. 108–114

[44] Zieris, F. : *Qualitative Untersuchung von Beeinflussungen der Aktivität des Partners bei der Paarprogrammierung*, Freie Universität Berlin, Diplomarbeit, 2012

[45] Zieris, F. ; Prechelt, L. : On Knowledge Transfer Skill in Pair Programming. In: *Proceedings of the ESEM 14* (2014)