Freie Universität Berlin

Diplomarbeit

Sommersemmester 2009

# Charakterisierung des Entwurfsprozesses bei der Paarprogrammierung

Wojciech Polcwiartek

polcwiar@inf.fu-berlin.de

Betreuer: Prof. Lutz Prechelt, Stephan Salinger

28. 05. 2009

Ich versichere hiermit an Eides Statt, dass diese Arbeit von niemand anderem als meiner Person verfasst worden ist. Alle verwendeten Hilfsmittel wie Berichte, 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 Prüfungskommision vorgelegt und auch nicht veröffentlicht.

.....
Wojciech Polcwiartek

Zunächst möchte ich Prof. Dr. Lutz Prechelt und Stephan Salinger für das in mich gesetzte Vertrauen danken, dass sie mir entgegenbrachten, in dem sie mich in ihre Arbeitsgruppe Software Engineering aufnahmen und mir damit die Möglichkeit eröffneten, mich im Rahmen meiner Diplomarbeit einem solch spannenden Thema wie der Paarprogrammierung zu widmen.

Auch während der Arbeit standen sie mir mit Rat und Tat zur Seite - auch wenn einige dieser Hinweise von mir erst nach einer Weile eingesehen wurden.

Dank geht auch an die Mitarbeiter des Fachbereichs Erziehungswissenschaften und Psychologie der FU Berlin Prof. Dr. Michael Niedeggen und Dr. Jens Eisermann für die Konsultation und Hinweise zu den in der Arbeit verwendeten sozialwissenschaftlichen Konzepten.

Außerdem danke ich Christopher Oezbek für die vielen anregenden Diskussionen u.a. über die Methodik und die Hintergründe der Arbeit. Darüber hinaus möchte ich allen anderen Mitgliedern der AG Software Engineering zum einen für die Unterstützung aber auch besonders für die freundliche Aufnahme in ihr Team danken, die die Zeit der Diplomarbeit zu etwas besonderem machten.

Zuletzt danke ich ganz herzlich Catharina Thüre, die mir beim Verfassen dieser Arbeit und der Durchsicht des Geschriebenen in den letzten Monaten immer zur Seite stand.

Vielen Dank!

.....

Wojciech Polcwiartek

# Inhaltsverzeichnis


**1 Einleitung** **3**

**2 Grundlagen** **4**
2.1 Paarprogrammierung . . . 4
2.2 Softwareentwurf . . . 5
2.3 Sozialwissenschaftliche Hintergründe . . . 6
2.3.1 Lernen . . . 6
2.3.2 Verteilte Kognition . . . 7
2.3.3 Gruppenleistung . . . 8
2.3.4 Motivation . . . 10
2.3.5 Problemlösen . . . 12
2.3.6 Konfliktforschung . . . 13
2.4 Grounded Theory . . . 14
2.5 Basiskonzeptmenge . . . 15

**3 Forschung** **19**
3.1 Verwendete Datenquellen . . . 19
3.2 Verwendete Werkzeuge . . . 21
3.3 Verwendete Kodiersyntax . . . 22
3.4 Verlauf der Forschung . . . 23

**4 Ergebnisse** **25**
4.1 Konzepte . . . 25
4.1.1 DesignEpisode . . . 25
4.1.2 DesignSequenz . . . 26
4.1.3 Actor . . . 26
4.1.4 Proposal . . . 26
4.1.5 Agreement . . . 27
4.1.6 Disagreement . . . 28
4.1.7 Amendment . . . 28
4.1.8 Challenge . . . 28
4.1.9 DesignElement . . . 28
4.1.10 IntroductionToDesignContext (ITDC) . . . 28
4.2 Designepisoden . . . 29
4.2.1 Episodentyp: Single-Handed . . . 29
4.2.2 Episodentyp: Guidance . . . 32
4.2.3 Episodentyp: DirectConsensus . . . 36
4.2.4 Episodentyp: Complement . . . 40
4.2.5 Episodentyp: Conflict . . . 43

**5 Zusammenfassung und Diskussion der Ergebnisse** **59**

**6 Grenzen der Arbeit und Ausblick** **69**

**7 Zusammenfassung** **71**

# 1 Einleitung

Die Paarprogramierung ist eine Arbeitstechnik, in der zwei Programmierer an einem Artefakt gemeinsam arbeiten. Sie ist vor allem aus Extreme Programming, einem agilen Entwicklungsprozess, bekannt. In diesem Prozess ist vor der Implementierung keine explizite Entwurfsphase vorgesehen, in der die Software in Gänze entworfen wird. Der Entwurf der Software wird in Extreme Programming iterativ in Paaren erstellt und umgesetzt.

In der Literatur zum Thema Paarprogramierung gibt es widersprüchliche Behauptungen über die Effekte dieses Vorgehens. Diese Behauptungen basieren oft auf quantitativen Untersuchungen.

In der Arbeitsgruppe Software Engineering an der Freien Universität zu Berlin wurde dieses Problem zu einem Forschungsschwerpunkt. Das Ziel dieser Foschung ist es, u.a. die Gründe solch unterschiedlicher Ergebnisse zu erfassen. Aus diesem Grund wurde eine qualitative Forschungsmethode zum Betrachten der inneren Mechanismen der Paarprogrammierung gewählt.

Einer dieser Aspekte ist die Erstellung des Entwurfs. Dieser Aspekt wurde zum Thema dieser Diplomarbeit.

Im Laufe der Untersuchung kristallisierte sich langsam die Forschungsfrage heraus, ob es beim Entwurfsprozess bei der Paarprogrammierung zu wiederkehrenden, klassifizierbaren Vorgehensweisen kommt und wie sich diese praktisch anwenden lassen.

# 2 Grundlagen

## 2.1 Paarprogrammierung

Die Paarprogrammierung (PP) ist eine Techik (auch Praktik genannt), die ihre Bekanntheit vor allem der Bekanntheit des *Extreme Programming*(XP), einer der Methoden agiler Software Entwicklung [4], verdankt.

Laurie Williams definiert das Pair-Programming wie folgt [50]:

> Pair programming is a practice in which two programmers work side-by-side at one computer, continuously collaborating on the same design, algorithm, code, or test.

Die Idee, technische Entwicklung in Paaren durchzuführen, kam bereits vor dem Jahr 1997 (erste Anwendung vom XP [2]) auf [37]. So wird von Tom De Marco der Begriff *jelled teams* für eine eng zusammenarbeitende Gruppe, in der synergetische Effekte auftreten, geprägt. Im Jahr 1996 veröffentlicht James O. Coplien das organisatorische Muster zur Anwendung bei der Softwareentwicklung „Developing in Pairs“ [33]. Coplien erwähnt ebenso den Begriff Synergie als eine der Auswirkungen der Entwicklung in Paaren.

Im Jahr 2001 veröffentlichten Laurie Williams und Alistair Cockburn die Ergebnisse ihrer umfassenden Untersuchungen [9]. Angeregt durch u.a. Untersuchungen auf dem Gebiet der Paarprogrammierung, analysierten die Autoren acht Faktoren, die den softwaretechnischen und organisatorischen Aspekt der PP beinflussen sollen. So waren für sie folgende Faktoren von Interesse:

- Wirtschaftlichkeit (*economics*)
- Zufriedenheit (*satisfaction*)
- Qualität des Entwurfs (*design quality*)
- Kontinuierliche Durchsicht (*continous rewiev*)
- Problemlösung (*problem solving*)
- Lernen (*learning*)
- Arbeitsgruppenbildung und Kommunikation (*team building and communication*)
- Verwaltung der Belegschaft und des Projekts (*staff and project management*)

Bis zu diesem Zeitpunkt war die PP ein Objekt von vor allem quantitativer Untersuchungen, die oft einen oder mehrere der unten genannten Faktoren untersuchten. So sind in dieser Arbeit mehrere Ergebnisse anderer Arbeiten (u.a. [49], [28]) verglichen und analysiert zusammengefasst worden.

Die Autoren haben anhand ihrer Forschungsarbeit folgende Thesen bzgl. der PP gestellt:

- viele Fehler werden bereits beim Tippen korrigiert
- der endgültige Anteil der Defekte im Quellcode ist niedriger
- die Qualität des Entwurfs ist besser und der dazugehörige Quellcode kürzer
- das Team löst die Probleme schneller
- die Programmierer lernen mehr über das System und die Softwareentwicklung
- am Ende des Projekts kennen mehrere Menschen unterschiedliche Teile des Systems
- die Teilnehmer des Projekts lernen besser zusammen zu arbeiten und zu kommunizieren, was zu einem besseren Informationsfluss führt
- die Zufriedenheit während der Arbeit ist höher

## 2.2 Softwareentwurf

Die Softwaretechnik, ein junger Bereich der Informatik, beschäftigt sich seit ihren Anfängen mit dem Problem des Softwareentwurfs. Während der ersten Softwaretechnikkonferenz (1968, Garmisch, Deutschland) gehört der Bereich Softwareentwurf zu den Hauptbereichen, mit denen sich diese Konferenz befasste [27]. Um sich diesem bisher noch wenig ausdifferenzierten Bereich zu nähren, wurde dieses Thema während dieser Konferenz erst allgemein und dann auch unter den Aspekten „Kriterien für Softwareentwurf“ und „Softwareentwurf: Strategien und Techniken“ behandelt. P. Naur beschreibt während dieser Konferenz das Profil eines Softwaredesigners:

> Software designers are in a similar position to architects and civil engineers, particularly those concerned with the design of large heterogeneous constructions, such as towns and industrial plants.

Zur Abgrenzung dieses Profils gibt er auch das Profil eines Programmierers an:

> The first thing to be done by the software producer about to start writing a piece of software, is the writing of a synopsis of what his piece of program is supposed to be doing. The next level may consist of a description of a few pages describing the essential data structures and the major processes to which they will be subjected.

Die heutige Softwaretechnik betrachtet den Softwareentwurf als einen der fünf grundlegenden Wissensgebiete der Softwaretechnik [1].

Im Laufe der Zeit wurden weitere Definitionen geprägt. Einige davon sind nachfolgend dargestellt.

Die IEEE definiert den Begriff „Software Design“ im Standard IEEE610.12 [31]. In diesem Standard wird der Softwareentwurf wie folgt definiert:

1. The process of defining the architecture, components, interfaces, and other characteristics of a system or component.
2. The result of the process in 1.

Zum anderen setzt der Standard ISO/IEC12207 den Entwurf in den Kontext der Softwareentwicklung [14]. Hier wird der Begriff Design sogar drei Mal als Teil des Entwicklungsprozesses erwähnt.

So zählt die ISO folgende, auf den Begriff Design bezogene, Aktivitäten auf:

- der architektonische Systementwurf (System architectural design)
- der architektonische Softwareentwurf (Software architectural design)
- der detaillierte Softwareentwurf (Software detailed design)

Unter dem Begriff des architektonischen Systementwurf versteht die ISO12207 die Erstellung und Beschreibung der Systemarchitektur auf dem höchsten Abstraktionslevel. Während des architektonischen Systementwurfs sollten Hardware- und Softwareteile so wie manuelle Operationen im gesamten System identifiziert werden.

Mit dem Begriff architektonischer Softwareentwurf bezeichnet die ISO12207 eine Reihe von Aktivitäten, die das Ziel haben, die Anforderungen an die Software so zu transformieren, dass eine Beschreibung der Softwarearchitektur auf dem höchsten Abstraktionslevel entsteht und die nötigen Softwarekomponenten und ihre externe Schnittstellen identifiziert sind.

Der detaillierte Softwareentwurf ist laut ISO12207 der Vorgang, die Softwarekomponenten so zu beschreiben, dass anhand dieser Beschreibung diese Komponenten kodiert, übersetzt und getestet werden können. Wenn nötig, sollten bei diesem Schritt die externen Schnittstellen einer Softwarekomponente ausführlicher beschrieben werden, so dass ihre Anwendung ohne zusätzliche Informationen erfolgen kann.

Im Rahmen dieser Diplomarbeit wird nur der Softwareentwurf betrachtet, der detaillierte sowie der architektonische.

## 2.3 Sozialwissenschaftliche Hintergründe

Um sich der PP von der qualitativen Seite zu nähern, müssen zunächst die einzelnen Aspekte und Begriffe näher betrachtet werden. Die in dieser Arbeit untersuchte Thematik lässt sich den sozialwissenschaftlichen Hintergründen des Lernens, der verteilten Kognition, der Gruppenleistung und der Motivation sowie den Problemlösungsaspekten als auch der Konfliktforschung zuordnen.

### 2.3.1 Lernen

Das Phänomen des Lernens gehört zu den ersten, mit dem sich die damals noch junge Wissenschaft der Psychologie im 19. Jahrhundert beschäftigt hat. Zu den Pionieren der

Forschung auf diesem Gebiet gehörte Hermann Ebbinghaus. Seine Arbeiten resultierten in der Formulierung von Gesetzen, die die Grundlagen der Lernpsychologie erschaffen haben.

So z.B. sind folgende empirische Untersuchungen und Ergebnisse zu finden [12]:

- Abhängigkeit des Lernerfolgs von der Zeit (sog. Lernkurve)
- Abhängigkeit des Vergessens von der Zeit (sog. Vergessenskurve)

Die Arbeit von Ebbinghaus führte auch zu der Erkenntnis, dass man sich beim Lernen vor allem die ersten und die letzten Elemente besonders effektiv merkt. Dieser Effekt wurde der serielle Positionseffekt (auch Primacy-Recency-Effekt oder Primäreffekt/Rezenzeffekt) gennant [51].

In dieser Diplomarbeit wird auch der Aspekt des Lernens als eine der Auswirkungen von PP betrachet, und aus diesem Grund wurden hier die grundlegenden Eigenschaften des Lernprozesses vorgestellt.

Die weitere Entwicklung der Psychologie führte zu einem bedeutendem Paradigmenwechsel. Das behavioristische Stimulus-Response-Paradigma [46] wurde in seiner Popularität durch das Stimulus-Organism-Response-Paradigma ersetzt, dessen Grundlagen von E. Tolman geschaffen wurden [24]. Laut dem S-R-Paradigma (auch Reiz-Reaktion-Paradigma) hängen die Reaktionen eines Tiers direkt und nur von den Reizen ab, denen das Tier ausgesetzt ist. Bei dem S-O-R-Paradigma (auch Reiz-Organismus-Reaktion-Paradigma) beeinflussen die Eigenschaften eines Tiers die Auswirkungen der Reize.

Tolmans Arbeiten können als Basis für die heutzutage bedeutende Forschungsrichtung der kognitiven Psychologie betrachtet werden. Diese Forschungsrichtung betrachtet den Menschen ganzheitlich in seinem Denken und Fühlen. Der Begriff der Kognition umfasst sämtliche mentalen Prozesse, u.a. Empfindung, Wahrnehmung, Mustererkennung, Aufmerksamkeit, Bewusstsein, Lernen, Gedächtnis, Erinnern, Sprache und Emotionen [39]. Das kognitive System wird als ein informationsverarbeitendes System aufgefasst. Die Forschung auf dem Gebiet der Kognition ist häufig interdisziplinär. Es lassen sich Einflüsse der Neurowissenschaft, der Linguistik oder der Informatik finden [3].

Der Begriff der Kognition und seine Bedeutung im Kontext der sozialwissenschaftlichen Forschung wurde in dieser Arbeit eingeführt, damit der nächste eingeführte Begriff, die verteilte Kognition, im sozialwissenschaftlichen Kontext und besser nachvollzogen werden kann.

### 2.3.2 Verteilte Kognition

Kollaborative Softwareentwicklung kann als ein Prozess der verteilten Kognition angesehen werden, was bereits von Flor und Hutchins untersucht wurde [16]. Im PP werden Probleme gemeinsam gelöst. Die Verteilung dieses Prozesses legt das Modell der verteilten Kognition nahe.

Die älteren Untersuchungen zum Thema der Kognition nahmen an, dass die Grenze der Untersuchung der einzelne Mensch bildet. Dies kam daher, dass kognitive Prozesse nur innerhalb einer Person, ohne Betrachtung des Kontextes, gesehen werden können.

Jedoch behauptete M. Halbwachs bereits in der ersten Hälfte des 20. Jh., dass das menschliche Gedächnis nicht in Gänze zu erklären sei, wenn es als Eigenschaft eines einzelnen Menschen betrachtet wird [18]. Ebenso wiesen March und Simon Mitte des 20. Jh. darauf hin, dass Organisationen als ein kognitives System betrachtet werden können [11].

In den achziger Jahren des 20. Jh. erschienen in der westlichen Welt zwei Arbeiten, die sich direkt dem Problem der Grenzen der Kognition widmeten. Sie wurden unter dem Namen „Society of Mind“ von M. Minsky [26] und „Mind in society“ von L.S. Vygotsky [44] veröffentlicht. Minsky behauptete, dass bereits die Kognition jedes einzelnen ein verteilter Prozess sei, da für die Verarbeitung der Informationen im Gehirn mehrere voneinander unabhängige Teile zuständig seien. Weiter folgerte er, dass sich solche kognitiven Systeme in einer Gesellschaft ähnlicher Systeme entwickeln. So gibt es innerhalb der menschlichen Kognition Teile, die mit externen kognitiven Systemen interagieren. Diese Interaktion beeinflusst die interne Kognition. Vygotsky stellte in seinem Werk die These auf, dass die kognitive Aktivität zwei Mal auftritt, zuerst als ein interpsychologischer und nur dann als ein intrapsychologischer Prozess. Er argumentierte, dass das kognitive System eines einzelnen Kindes seine Form und Funktion durch die Interaktion mit anderen Personen oder Artefakten entwickeln kann.

Die verteilte Kognition kann dreierlei verstanden werden [21] :

- kognitiver Prozess kann unterhalb der Mitglieder einer sozialer Gruppe verteilt werden
- kognitiver Prozess kann Koordination zwischen der internen und der externen (kognitiven) Struktur beinhalten
- kognitiver Prozess kann zeitlich so verteilt werden, dass die Ergebnise der früheren Ereignisse die späteren beeinflussen können

### 2.3.3 Gruppenleistung

Mit dem Begriff der Gruppenleistung ist die gemeinsame Fähigkeit, eine Aufgabe in einer bestimmten Zeit zu erledigen, gemeint. Dies ist ein Thema, mit dem sich die Sozialwissenschaft bereits eine lange Zeit befasst hat. Das Ziel dieser Untersuchungen war es, die Phänomene auf diesem Gebiet zu entdecken, die unabhängig von der konkreten Art der Aufgabe auftreten. Die Leistung des Paars war einer der Faktoren, die in der Arbeit Cockburn und Williams [9] bei der Untersuchung der Wirtschaftlichkeit der Paarprogrammierung betrachtet wurde. Im Rahmen dieser Diplomarbeit wurden Modelle und Begriffe aus diesem Gebiet verwendet.

Die ersten Untersuchungen auf diesem Gebiet sind auf M. Ringelmann zurückzuführen [40]. Er untersuchte Anfang des 20. Jahrhunderts die Leistung von Zugvieh (Pferde,

Ochsen). So bildete er Viehgruppen und untersuchte die Fähigkeit, unterschiedliche Gewichte zu ziehen. Es fiel ihm auf, dass die Leistungfähigkeit der Viehgruppen nicht linear von der Anzahl des Viehs abhängt. Je grösser die Tiergruppen waren, desto niedriger war die Leistungsfähigkeit pro Tier. Diesen Effekt nennt man heute den Ringelmann-Effekt.

Dieser Effekt zeigt, dass unabhängig vom Willen, selbst bei einfachsten Aufgaben, die Gruppenleistung in der Abhängigkeit von der Gruppengröße nicht unbedingt linear skaliert.

Im Jahr 1975 formulierten Hackman und Morris die Leistung einer Gruppe wie folgt [22] [43]:

> Gruppenproduktivität = potenzielle Produktivität + Prozessgewinne - Prozessverluste

Die Prozessgewinne und Verluste beziehen sich hauptsächlich auf Änderungen in der Gruppenleistung, die wegen Änderungen der Motivation, der Koordination und der individuellen Fertigkeiten zustande kommen.

Im Jahr 1972 untersuchte Steiner die Gruppenleistung in Bezug auf die Art der Aufgabe [42]. Er gliederte fünf Aufgabentypen heraus. Dazu gehören u.a.

1. additive Aufgaben
2. konjunktive Aufgaben
3. disjunktive Aufgaben

Das Verständnis für die weiteren zwei Aufgabentypen (kompensatorische und mit Ermessenensspielraum) wurde im Kontext dieser Arbeit nicht als relevant betrachtet und daher nicht weiter berücksichtigt.

Das Ergebnis der additiven Aufgaben einer Gruppe ist die Summe der Ergebnisse der einzelnen Personen (z.B. Schneeschaufeln). Somit ist hier die potenzielle Leistung der Gruppe die Summe der potenziellen Leistungen der Mitglieder.

Das Ergebnis der konjunktiven Aufgaben hängt von der Leistung des schwächsten Mitglieds ab. Dies ist z.B. beim gemeinsamen Bergsteigen der Fall. Die Bergsteigergruppe muss aus Sicherheitsgründen immer zusammen bleiben, und die schnelleren Bergsteiger müssen dementsprechend ihr Tempo dem langsamsten anpassen.

Bei den disjunktiven Aufgaben hängt das Ergebnis der Aufgabe von der Qualität der von der Gruppe gewählten Lösung ab (z.B. gemeinsames Problemlösen). Die kollaborative Softwareentwicklung kann daher als disjunktive Aufgabe gesehen werden.

Die potenzielle Leistung der Gruppe bei konjunktiven und disjunktiven Aufgaben erfolgt nicht durch Summierung der Potenziale der Mitglieder. Wenn $N$ die Gruppengröße und $0 \leq Q \leq 1$ das durchschnittliche Potenzial der Mitglieder der Gruppe ist, so ist das approximierte Gruppenpotenzial $P$

- $P = Q^N$ bei konjunktiven Aufgaben

- $P = 1 - (1 - Q)^N$ bei disjunktiven Aufgaben

Die Gruppenmitglieder können bei der Erledigung der gemeinsamen Aufgaben unterschiedlich handeln, wenn es um die Nutzenorientierung ihres Handelns geht [15]. Von besonderem Interesse sind zwei spezielle Handlungsarten, zum einen die kooperative und zum anderen die kompetitive Nutzenorientierung. Das kooperative Handeln wird wie folgt von Ertinger beschrieben „Entscheide dich für die Handlungsmöglichkeit, die den größten gemeinsamen Gewinn nach sich zieht". Das kompetive Handeln beschreibt Ertinger als „Entscheide dich für die Handlungsmöglichkeit, die den größten Unterschied in den Gewinnen zu Gunsten deiner Person nach sich zieht". Sollte der Nutzen der einen Person in einer Dyade als $X$ beschrieben werden und der Nutzen der anderen Person als $Y$, so handelt die Dyade kooperativ, wenn beide Personen das $max(X + Y)$ anstreben. Sollte eine der Personen bei ihrem Handeln $max(X - Y)$ anstreben, so handelt sie kompetitiv. Ertinger erwähnt in seiner Arbeit noch sechs weitere Nutzungsorientierungsfunktionen: altruistisch, individualistisch, aggressiv, destruktiv, auto-aggressiv und selbstaufopfernd. Diese Nutzungsorientierungsfunktionen wurden im Rahmen dieser Diplomarbeit nicht beachtet.

Die Nutzenorientierung der erbrachten Leistung wurde bei der Betrachtung der Konflikte berücksichtigt.

### 2.3.4 Motivation

Da die Gruppenleistung von motivationalen Faktoren beeinflusst wird, wird hier der Begriff des Motivs und der damit verbundenen Motivation eingeführt. Es folgt hier erst eine allgemeine Einführung in die Problematik der Motivation und dann eine Beschreibung eines konkreten Models zum Thema Motivation während der Arbeit.

Das Wahrig-Wörterbuch [45] kennzeichnet den Begriff der Motivation als Gesamtheit der Motive. Ein Motiv wird wiederum u.a. als Leitgedanke oder Beweggrund beschrieben.

Einer der frühen Motiv(ation)forscher D. McClelland beschreibt im 1951 den Begriff des Motives ausführlicher. Ein Motiv sei „eine starke affektive Assoziation, die durch eine antizipatorische Zielreaktion gekennzeichnet ist und auf vergangene Assoziationen bestimmter Hinweisreize mit Genuss und Schmerz beruht" [19].

Die Motivationsforschung beschäftigt sich mit Aktivitäten, die das Verfolgen eines angestrebten Ziels erkennen lassen und unter diesem Aspekt eine Einheit bilden [19]. Im weiten Feld der Motivationsforschung hat die Erforschung der Arbeitsmotivation eine gesonderte Stellung eingenommen. Sie beschäftigt sich mit der Untersuchung der Motive und der Motivation während der Arbeit.

Wenn man die heutige Literatur zu den Theorien der Arbeitsmotivation betrachtet, so stellt man fest, dass sie sich hauptsächlich mit dem Erforschen der Antworten auf zwei Fragen beschäftigen:

- Welche Faktoren motivieren zur Arbeit? (Inhaltstheorien)
- Wie kann das Arbeitsverhalten von der Motivation beeinflusst werden? (Prozesstheorien)

In der Definition der Motive von McClelland ist seine Zugehörigkeit zur Schule des Behaviosismus u.a. durch die Verwendung der Begriffe „Reize“ und „Assoziation“ sichtbar, die vor allem im Kontext der klassischen Konditionierung verwendet werden [29].

Zu den ersten allgemeinen Motivationstheorien gehörte die Theorie von Maslow [25], den man ebenso zu den behavioristischen Forschern zählen kann. Sie ging in die Geschichte der Wissenschaft als die sog. „Maslow'sche Bedürfnispyramide“ ein. Die Theorie von Maslow behauptet, dass der Mensch motiviert ist, hierarchische Bedürfnisse zu befriedigen. Die eindimensionale Bedürfnishierarchie sieht nach Maslow wie folgt aus:

1. Körperliche Existenzbedürfnisse (z.B. Nahrung, Schlaf, Atmung)
2. Sicherheit (z.B. Schutz vor Gefahren, fester Arbeitsplatz)
3. Soziale Beziehung (z.B. Familie, Kommunikation)
4. Soziale Wertschätzung (z.B. Soziale Anerkennung, Wohlstand)
5. Selbstverwirklichung (z.B. Individualität, Talententfaltung)

Die ersten drei Stufen der Hierarchie sind sog. Defizitbedürfnisse. Diese müssen befriedigt sein, damit ein Mensch den Zustand der Zufriedenheit erreichen kann. Laut Maslows Theorie versucht der Mensch, zuerst die erste Stufe der Hierarchie zu befriedigen. Wird diese befriedigt, versucht er, die nächste Stufe zu befriedigen. So handelt der Mensch, bis die letzte Stufe erreicht wird. Die letzte Stufe (Bedürfnis der Selbstverwirklichung) kann nicht vollständig gestillt werden.

Eine weitere Theorie zur Motivation formulierte F. Herzberg [20]. Seine Theorie bezieht sich auf die Arbeitsmotivation, ist empirisch untermauert und betrachtet die Motivation zweidimensional. So nimmt Herzberg zwei Arten der Bedürfnisse an:

1. Motivationsbedürfnisse (auch Zufriedensteller oder intrinsische Faktoren)
2. Hygienebedürfnisse (auch Unzufriedensteller oder extrinsische Faktoren)

So gehören u.a. Erfolg und Anerkennung zu den wichtigsten Motivationsfaktoren. Die Organisationspolitik und -verwaltung sowie die Kompetenz des Vorgesetzten sind wiederum die wichtigsten Hygienefaktoren.

Sollten die Hygienefaktoren nicht erfüllt sein, führt das zu einer hohen Unzufriedenheit. Die Erfüllung der Hygienefaktoren resultiert in einem Zustand, in dem die Unzufriedenheit nicht vorhanden ist.

Die Motivationsfaktoren können Zufriedenheit schaffen. Sind diese nicht erfüllt, kann ein Mitarbeiter nicht zufrieden sein. Die Erfüllung dieser Faktoren führt zu einer hohen Zufriedenheit.

So beinhalten die Ergebnisse seiner Arbeit die Zusammenstellung der Motivations- und Hygienefaktoren in der Arbeitsumgebung.

### 2.3.5 Problemlösen

Der Enwicklungsprozess bei der PP kann im allgemeinen als ein Problemlösenprozess aufgefasst werden. Es ist ein kognitiver Prozess und kann somit verteilt werden.

Die Art und Weise, wie der Mensch im allgemeinen Probleme lösen kann, war ebenso ein Bereich, mit dem sich die Sozialwissenschaft beschäftigt hat.

Der Begriff des Problem(lösen)s wurde von Dunker bereits im Jahr 1935 wie folgt definiert [13]:

> Ein Problem entsteht z.B. dann, wenn ein Lebewesen ein Ziel hat und nicht weiss, wie es dieses Ziel erreichen soll.

Ein prominentes Modell auf diesem Gebiet wurde von March und Simon geschaffen [38]. Dieses Modell ist im kognitiven Paradigma ansäßig, worauf sein ursprüngliches Ziel hindeutet. Das Modell sollte dazu taugen, das menschliche Problemlösen als ein Informationsverarbeitungsprozess auf einem Computer zu simulieren [24].

Der Prozess des Problemlösens ist laut March und Simon ein Suchvorgang im Problemraum [34].

Der Problemraum wird von March und Simon wie folgt definiert:

- Eine Menge der Elemente $U$, die den Wissenszustand über die Aufgabe repräsentieren.
- Eine Mange der Operatoren $Q$, die auf den Elementen aus $U$ angewendet werden können, wodurch ein neuer Wissenszustand erzeugt wird.
- Ein initialer Wissenszustand $u_0 \in U$, der das Wissen über das Problem am Anfang des Lösens repräsentiert.
- Ein Problem, das durch die Angabe der Zielzustände $G \subseteq U$ angegeben wird. Das Erreichen der Zielzustände $G$ wird durch die Anwendung der Operatoren aus $Q$ realisiert.
- Sämtliches Wissen, das dem Löser beim Problemlösen zugänglich ist.

Der Prozess des Lösens (das Anwenden der Operatoren auf Wissenszustände laut March und Simon) kann nach unterschiedlichen Strategien erfolgen. Bei manchen Problemen ist es möglich, einen genauen Algorithmus anzugeben. Andere Probleme lassen sich durch die Anwendung der Heuristiken und "Versuch und Irrtum"-Verfahren, oder durch systematische Variation lösen. Eine Allgemeinstrategie zur Lösung von Problemen wurde als Algorithmus im Jahr 1974 von Putz und Osterloh verfasst [38].

### 2.3.6 Konfliktforschung

Das Wort Konflikt wird umgangssprachlich vor allem in zwei Bedeutungen verwendet ([6], [48]):

- Zwiespalt, (innerer) Widerstreit
- Zerwürfnis, Streit, Auseinandersetzung.

Beide dieser Interpretationen betonen eine negativ wahrgenommene Situation, die mit dem Namen Konflikt beschrieben werden kann. Die Konfliktforschung ist interdiziplinär. Sie liegt u.a. im Interessenfeld der Politikwissenschaften, Soziologie und Psychologie, wobei jede dieser Wissenschaften sich mit unterschiedlichen Konfliktaspekten beschäftigt. In dieser Arbeit werden vor allem die (sozial-)psychologischen Aspekte des Begriffs Konflikt betrachtet.

In den Sozialwissenschaften werden somit mehrere Definitionen vom Konflikt verwendet. Die einen beziehen sich auf die subjektive Wahrnehmung des Konfliktes ([30]), andere auf den objektiven Gegenstand eines Konflikes ([8]).

Eine breite Definition des Begriffs gibt Becker [5] an:

> In der Psychologie, aber auch in den Sozialwissenschaften allgemein, spricht man von einem Konfikt dann, wenn zwei Elemente gleichzeitig gegensätzlich oder unvereinbar sind.

Diese Deutung des Konfliktbegriffs wird in dieser Arbeit verwendet.

Die Grundlagen der soziopsychologischen Konfliktforschung wurden von Kurt Lewin gelegt. Seine Feldtheorie resultierte in einer Typisierung der Konflikte [47]. So fand Lewin drei Konfliktarten: Annäherung - Annäherung, Vermeidung - Vermeidung und Annäherung - Vermeidung. Eine Konfliktsituation der Art Annäherung - Annäherung (genannt auch Appetenzkonflikt) tritt auf, wenn eine Person von mindestens zwei Kräftefeldern mit positiver Valenz beeinflusst wird. Ein Beispiel für solch einen Konflikt ist der intrapersonelle Konflikt in einer Eisdiele: „Welche Eissorte soll ich wählen: Vanille- oder Schokoladeneis?“ Solch ein Konflikt tritt in Situationen auf, in denen es mindestens zwei attraktive Ziele gibt.

Ein Aversionskonflikt (Konfliktart: Vermeidung-Vermeidung) tritt beispielsweise dann auf, wenn der Einfluss auf eine Person von mindestens zwei Kräften mit negativer Valenz stammt. Diese Form kann zum Beispiel bei der Planung eines knappen Haushalts vorkommen: „Auf welche Freizeitaktivität muss ich verzichten: Kino oder Tennis? “.

In den Appetenz-Aversion-Konflikten (Konfliktart: Annäherung-Vermeidung) wird eine Person von mindestens zwei Käftefeldern beeinflusst, die durch Ambivalenz gekennzeichnet sind. Sie haben für die Person sowohl positive als auch negative Valenz. Eine Entscheidungssituation zwischen einer riskanten Kapitalanlage mit hohem möglichen Gewinn und einer, die wenig Gewinn bringt, dafür aber nicht so riskant ist, wäre ein Beispiel einer solchen Konfliktart. Beide Alternativen haben sowohl Vor- und Nachteile, weswegen der Einfluss auf die Person ambivalent sein kann.

Weitere bedeutende Konfliktforscher sind u.a. Friedrich Glasl und Gerhard Schwarz [7].Beide beschäftigten sich mit Konflikten und deren Behandlung in einer organisationalen Umgebung. F. Glasl ist u.a. das Phasenmodel der Konflikteskalation zu verdanken. G. Schwarz erarbeitete sechs wiederkehrende Konfliktstile. Es sind generelle Muster im Konfliktverlauf, die u.a. durch die angewandte Handlungsstrategie im Konflikt determiniert sind. Er beschreibt folgende Konfliktstile:

- Flucht
- Vernichtung
- Unterwerfung
- Delegation an dritte Instanz
- Kompromiss
- Konsens

Die Untersuchung der *Conflict*-Episode im Rahmen dieser Diplomarbeit wurde vom Schwarzschen Modell angeregt.

## 2.4 Grounded Theory

Die Untersuchung der Paarprogrammierung in der AG Software Engineering an der FU Berlin wurde anhand der "Grounded Theory" Methodik betrieben.

Diese Methodik wird ebenso bei der Untersuchung für diese Diplomarbeit angewendet.

Grounded Theory (GT)ist eine qualitative Forschungsmethodik, die von den zwei Soziologen B. Glaser und A. Strauss in den sechziger Jahren des 20 Jh. entwickelt wurde [17]. Die Methode wurde auf der theoretischen Basis des Pragmatismus und des Symbolischen Interaktionismus begründet [10]. Eine explizite Beschreibung des Begriffes der GT ist in [41] zu finden:

> What do Strauss and Corbin mean when they use the term „grounded theory"? They mean theory that was derived from data, systematically gathered and analyzed through the research process. In this method, data collection, analysis, and eventual theory stand in close to one another. A researcher does not begin with a preconceived theory in mind /.../. Rather, the researcher begins with an area of study and allows the theory to emerge from the data. Theory derived from data is more likely to resemble the „reality" then is theory derived by putting together a series of concepts based on experience or solely through speculation /.../.

Die Literatur zum Thema GT enthält eine Anzahl von Prozeduren, die es zum Ziel haben, eine Menge von Konzepten zu schaffen, die eine gründliche Erklärung eines sozialen Phänomens ermöglichen [10].

Die wichtigsten dieser Prozeduren werden werden von Strauss und Corbin unter dem Namen *Coding* (Kodieren) zusammengefasst. Mit diesem Begriff ist der Prozess gekennzeichnet, Daten zu analysieren.

Strauss und Corbin nennen in ihren Arebiten drei Arten vom Kodieren:

1. *Open Coding* (offenes Kodieren) - ein interpretativer Prozess, in dem die Daten analytisch betrachtet werden. Das Ziel des offenen Kodierens ist es, dem Forscher neue Einblicke in das untersuchte Phänomen zu erlauben. Beim offenen Kodieren werden Ereignisse/Aktionen miteinander verglichen und mit konzeptuellen Namen beschrieben. Konzeptuell ähnliche Namen werden in Kategorien zusammengefasst. Die Kategorien sollten dann auf das Vorhandensein von Eigenschaften untersucht werden.

2. *Axial Coding* (axiales Kodieren) - eine Menge von Prozeduren, in denen die Daten nach dem offenen Kodieren durch die Untersuchung von Relationen innerhalb der Kategorien mit neuen Relationen versehen werden. Die Relationen innerhalb der Kategorien sollten dadurch ebenso gegen die Datenbasis überprüft werden. Die Kategorien können hier anhand des *paradigm model* (Paradigma-Modell) [41] analysiert werden. Dieses Model untersucht die Bedingungen und Auswirkungen eines Phänomens, so wie seinen Kontext und die Strategien, die im Rahmen dieses Phänomens unternommen werden.

3. *Selective Coding* (selektives Kodieren) - ein Prozess, eine Kategorie zur Kernkategorie zu ernennen und diese mit anderen Kategorien in Beziehung zu setzen. Die Kernkategorie repräsentiert das zentrale untersuchte Phänomen.

Das offene und axiale Kodieren sind zwar unterschiedliche Kodierprozeduren, sie sollten aber abwechselnd angewendet werden. Das selektive Kodieren führt zur Erstellung einer Theorie, und es kommt vor allem in den späteren Phasen der Untersuchung vor [10].

## 2.5 Basiskonzeptmenge

Die Diskrepanz in den Ergebnissen verschiedener quantitativer, empirischer Untersuchungen im Bereich der Paarprogrammierung war einer der Gründe für das verstärkte Interesse der Arbeitsgruppe Software Engineering im Fachbereich Mathematik und Informatik an Freien Universität Berlin für das PP. Die Arbeiten von Laura Plonka, Lutz Prechelt und vor allem von Stephan Salinger führten zur Erstellung einer Menge der Konzepte zur Beschreibung der grundlegenden Aktivitäten, die bei der Paarprogrammierung vorkommen können [36]. Diese Menge heißt Basiskonzeptmenge (BKM).

Die Menge wurde durch die Anwendung der Grounded-Theory-Methodik anhand ergiebiger Daten erarbeitet. Die dort verwendete Form der Daten gleicht der, die in dieser Arbeit benutzt wurde (u.a. Videoaufnahmen der Programmierer; siehe Kap. Verwendete Datenquellen). Der Verlauf dieser Forschungsarbeit wird teilweise in [35] beschrieben.

Die Basiskonzeptmenge enthält drei disjunkte Untermengen:

- HCI-Konzepte (Human-Computer-Interaction)
- HEI-Konzepte (Human-Enviroment-Interaction)
- HHI-Konzepte (Human-Human-Interaction)

Die HCI-Menge enthält Konzepte, die zur Beschreibung der Interaktionen eines Menschen mit dem Computer verwendet werden. In dieser Menge sind folgende Konzepte zu finden:

- *write_sth* - Tippen auf der Tastatur
- *search_sth* - Suche nach einem konkreten Zielobjekt
- *explore_sth* - Explorieren vorhandener Artefakte; Suche, ohne ein konkretes Zielobjekt
- *do_sth* - Sonstige Aktivitäten, die die Maus oder Tastatur in Anspruch nehmen

Die Interaktion eines Programmierers mit der Umwelt beschreiben Konzepte aus der HEI-Menge. Diese Menge beinhaltet folgende Konzepte:

- *show_sth* - Zeigen einer bestimmten Stelle, z.B. mit dem Finger
- *verify_sth* - Überprüfen eines Arbeitsergebnisses, z.B. durch Testen oder Durchsicht
- *examine_sth* - Lesen eines unbekannten Textes mit dem Ziel, diesen nachvollziehen zu können
- *read_sth* - Lautes (vor)lesen, z.B. einer Fehlermeldung
- *sketch_sth* - Skizzieren, z.B. auf dem Papier
- *read_requirement* - Lesen von Anforderungen

Zusätzlich werden bei der Anwendung der Basiskonzeptmenge Hilfskonzpte verwendet (*auxiliary concepts*). Diese sind streng genommen nicht ein Teil der BKM, sind jedoch behilflich beim Verstehen einer kodierten Sitzung. Zu den Hilfskonzepten können folgende zugeordnet werden:

- *P¡X¿.become_driver* -P¡X¿ wird zum Driver
- *divide_work* - Das Paar fängt an, separat zu arbeiten

- *interrupt* - Unterbrechung von extern
- *wait* - Untätiges Warten, z.B. beim Quellcodeübersetzen

Die HHI-Menge ist die größte Untermenge der BKM. Sie wird zusätzlich in drei disjunkte Untermengen aufgeteilt. So gehören zu dieser Menge produktorientierte, prozessorientierte und generische Konzepte.

Eine Auflistung der zur HHI-Menge zugehörigen Konzepte ist in der Abbildung 1 zu finden.

Innerhalb der generischen Konzepte ist noch eine zusätzliche Untermenge herausgegliedert. Diese Untermenge umfasst Konzepte, die einen vereinfachten Blick auf eine Situation erlauben. Diese Konzepte werden bei einer verbalen Äußerung, die während einer HCI oder HEI Aktivität gestellt wird, verwendet.

Die BKM erhebt nicht den Anspruch auf Vollständigkeit. Die BKM bildet nur eine Basisschicht (*foundation layer*), die die grundlegenden Aktivitäten beschreibt. Weitere Schichten, mit weiteren Konzepten, sind denkbar.

Abbildung 1: Die HHI-Konzepte. Eine Kopie der Abbildung aus [36]

# 3 Forschung

## 3.1 Verwendete Datenquellen

In der Arbeit wurden als Datenquellen drei Programmiersitzungen verwendet. Diese Sitzungen wurden aus Sitzungen ausgewählt, die von Frau Laura Plonka mit dem Zweck aufgenommen wurden, diese für wissenschaftliche Arbeiten in der Arbeitsgruppe Software Engineering im Bereich Paarprogrammierung zu verwenden.
Die Aufnahmen beinhalten:

- Bildschirmvideo - ein hochaufgelöstes Video mit dem Inhalt des Bildschirms, der vom Paar zum Programmieren in der Sitzung verwendet wird
- Personenvideo - ein niedrigaufgelöstes Video mit einer frontalen oder seitlichen Aufnahme des Paars
- Personenaudio - eine Audioaufnahme des Paars während der Sitzung (verfügbar als eine im Video eingebettete Monospur und externe Stereospur)
- Paarumfrage - eine von Frau Plonka gestaltete Umfage, die aus den beiden Fragebögen „Beschreibung der zu bearbeitenden Aufgaben“ (ausgefüllt vor der Sitzung) und „Verlaufsbeschreibung Paarprogrammiersitzung“ (ausgefüllt nach der Sitzung) besteht; die Fragebögen wurden separat von jedem Enwickler ausgefüllt; die Umfagen enthalten persönliche Angaben der Programmierer sowie zusätzliche Beschreibungen der Sitzung (u.a. Art der Sitzung)

Die Videoquellen und die Audioquelle mussten aufgearbeitet werden. Das Video musste mit der Tonspur synchronisiert und geschnitten werden. Diese Arbeiten wurden bereits teilweise von S. Salinger erledigt.

Die drei Sitzungen stammen aus professionellen Umgebungen. Sie fanden in einem in Deutschland ansässigen Unternehmen im Jahr 2008 statt. Daran beteiligten sich nur professionelle Entwickler. Die Zusammensetzung des Paars, die Art der zu bearbeitenden Aufgabe und die verwendete Technologie waren in den Sitzungen unterschiedlich.

In der Paarumfrage wurde u.a. die Art der Sitzung erfragt. Diese konnte jeder Programmierer aus folgender Menge wählen:

- Neuentwicklung Funktionalität
- Weiterentwicklung Funktionalität
- Refactoring
- Testfallentwicklung
- Debugging
- Sonstiges (hier konnten eigene Beschreibungen angegeben werden)

Die beiden Programmierer stimmten in jeder der verwendeten Sitzungen in der Kennzeichnung der Sitzungsart überein. Diese Kennzeichung wurde für die Beschreibung der Aufgabe der Sitzung übernommen. Darüberhinaus gaben die Programmierer ihre Berufserfahrung in Jahren und Monaten an, was als Grundlage für die Zuordung zu der Menge "Experte" oder "Anfänger" diente.

**Sitzung FA02** Das Paar arbeitet an einem umfangreichen Projekt. Aus diesem Projekt werden mehrere konkrete Produkte erstellt. Aus einer Codebasis werden Applikationen mit unterschiedlichen Mengen der vorhandenen Funktionalitäten erzeugt. Das Ziel des Projekts ist es, ein Geoinformationssystem zu erstellen.

In der hier bearbeiteten Sitzung FA02 befasste sich das Paar mit einem Refactoring. Beide Programmierer haben in ihren früheren Sitzungen separat Funktionalitäten geschaffen, die in dieser Sitzung aneinander angepasst werden sollen. Das Refactoring betrifft Funktionalitäten, die, nach dem „Model-View-Controller" (MVC)-Architekturmuster [32], der „Präsentation"-Komponente zuzuordnen sind. So beschäftigen sich die Programmierer mit Klassen wie z.B. `VirtualColumns` oder `VirtualAttributes`. Die genaue Bedeutung der verwendeten Namen und Begriffe kann nicht eindeutig festgestellt werden. Die führende Person (im Sinne der Meinungsführung) in der Sitzung ist P1. Das Paar kann als „Experte-Experte"-Konstellation betrachtet werden, da beide Programmierer mehrere Jahre Arbeitserfahrung aufweisen können (P1: 20 Jahre Berufserfahrung; P2: 9 Jahre Berufserfahrung). Das Paar verwendet die „Java"-Programmiersprache in der „Eclipse"-Entwicklungsumgebung.

**Sitzung FA03** Das Paar arbeitet an einem Projekt, das vermutlich einen Teil eines grösseren Projektes bildet. In der Sitzung wird die neue Klasse `FerryPolygonChecker` erstellt. Diese basiert auf der bereits vorhandenen Klasse `FerryIDChecker`. Die neue Klasse wurde durch das Kopieren der alten Klasse erzeugt und zusätzlichen Anpassungen unterzogen. Das bearbeitete Projekt ist ein Teil eines Geoinformationssystems. Die Aufgabe dieser Sitzung betrifft, nach dem MVC-Architekturmuster, die „Steuerung"-Komponente. Die vom Paar verwendeten Begriffe wie z.B. `FerryPolygonChecker` oder `RtgMapData` können nicht genau nachvollzogen werden. Die führende Person (im Sinne der Meinungsführung) der Sitzung ist P2. Die Zusammensetzung des Paars deutet auf eine „Anfänger-Experte"-Konstellation hin (P1: 6 Monate Berufserfahrung; P2: 10 Jahre Berufserfahrung). Das Paar verwendet die „C++"-Programmiersprache in der „Microsoft VisualStudio"-Entwicklungsumgebung mit der „VisualAssist X"-Erweiterung.

**Sitzung FA04** Das Paar arbeitet am gleichen Projekt wie das Paar aus der Sitzung FA02. Die Aufgabe der Sitzung ist es, neue geometrische Operationen auf geographischen Objekten zu implementieren. So bezieht sich die Aufgabe auf die „Steuerung"-Komponente nach dem MVC-Architekturmuster. Die Aufgabe wird von den Programmierern in mehreren, miteinander verbundenen Tasks erledigt. Das Paar besteht aus

| Name der Sitzung | FA02 | FA03 | FA04 |
|---|---|---|---|
| Zusammensetzung des Paars | Experte - Experte | Anfänger - Experte | Experte - Experte |
| Verwendete Programmiersprache | Java | C++ | Java |
| Aufgabe in der Sitzung | Refactoring | Entwicklung neuer Funktionalität | Entwicklung neuer Funktionalität |

Tabelle 1: Vergleich der Programmiersitzungen

zwei erfahrenen Programmierern (P1: 12 Jahre Berufserfahrung; P2: 10 Jahre Berufserfahrung), so kann man hier von der Konstellation „Experte-Experte“ sprechen. Das Paar verwendet die „Java“-Programmiersprache in der „Eclipse“-Entwicklungsumgebung.

**Sitzung FA05** Die Aufgabe dieser Sitzung war es, einen Defekt zu beseitigen. So handelte es sich um eine Sitzung des Typs „Debugging“. Aus Mangel an identifizierbaren Tätigkeiten, die als Teil des Entwurfsprozesses aufgefasst werden könnten, wurde diese Sitzung nicht weiter betrachtet.

## 3.2 Verwendete Werkzeuge

Die Analyse der Daten erfolgte mit Hilfe der folgenden EDV-Werkzeuge.

**Camtasia Studio** Die Daten jeder Sitzung waren als eine „Camtasia-Projekt“-Datei verfügbar. Dabei handelt es sich um eine Datei, die das Paar- und Bildschirmvideo und eine Monoaudiospur von der Sitzung enthielt. Diese Datei liess sich mit der Software „Camtasia Studio 4“ (Hersteller: „TechSmith Corporation“) öffnen und bearbeiten. Die Aufgabe dieser Software war es, die Filme zu schneiden, Bild mit Ton zu synchronisieren und die Daten in einem gängigen Videodateiformat (Windows Media Video) zu speichern.

**Atlas.ti** Zur Analyse und Beschreibung der Daten wurde die Software „Atlas.ti 5“ (Hersteller: „ATLAS.ti Scientific Software Development GmbH“) verwendet. Die Software ermöglicht es, unterschiedliche Medien (statische Bilder, Text, Audio und Video) zu verarbeiten. Daten, die eine dieser Medienformen aufweisen, können damit beschrieben und analysiert werden. Die in dieser Software verwendeten Konzepte ähneln diesen, die in „Grounded Theory“ verwendet werden. So nennt „Atlas.ti“ die Phänomene aus der GT „Quotations“. Die Konzepte aus GT werden teilweise durch „Codes“ im „Atlas.ti“ abgebildet. Die Memos aus Grounded Theory sind als „Memos“ verfügbar.
So ermöglicht diese Software zeitlich zusammenhängende Teile der Videos als eine

„Quotation“ zu markieren. Einer „Quotation“ wird automatisch eine Kennung zugewiesen, die aus zwei natürlichen Zahlen besteht. Die Kennung wird als [X:Y] kodiert, wobei X für die fortlaufende Nummer des Filmes im „Atlas.ti“-Projekt und Y für die fortlaufende Nummer der „Quotation“ im Film steht. Beide Nummern werden automatisch und inkrementell erzeugt (vergleichbar mit der Funktion von automatisch- inkrementellen Feldern bekannt aus Datenbanken). Eine „Quotation“ kann daraufhin mit einer (vernünftig) beliebigen Anzahl von „Codes“ und „Memos“ verknüpft werden. Zwischen den „Memos“ und „Codes“ einerseits und den „Quotations“ andererseits kann eine „N-zu-M“-Beziehung erstellt werden. Die Software bildet noch weitere Teile der „Grounded Theory“ ab (z.B. Kategorien, Beziehungen innerhalb der Kategorien). Diese Möglichkeiten der Software wurden nicht verwendet. Dies hat den Grund, dass sich die Software allein als wenig geignet zum Verarbeiten der Videodaten und Analysieren von zeitlichen Verläufen herausstellte. Die Video- und Audiodaten sind sehr ergiebige Daten - wenige Sekunden der Audio- und Videodaten können in einer ausführlichen Beschreibung resultieren. Die Software gibt jedoch nur eingeschränkt die Möglichkeit, die Beschreibung zu betrachten.
„Atlas.ti“ verfügt über die Möglichkeit, die Beschreibungen in Form einer XML-Datei zu exportieren.

**Atlas-prozessor** Zur Analyse der im „Atlas.ti“ erstellten Beschreibungen wurde eine eigene Software im Rahmen dieser Diplomarbeit entwickelt. Dies hatte das Ziel, die Schwächen von „Atlas.ti“ bei der Visualisierung der Beschreibungen der Audio- und Videodateien auszugleichen. Die wichtigste Funktionalität des „Atlas-prozessors“ ist es, die aus „Atlas.ti“ exportierte XML-Datei zu verarbeiten und die Daten (z.B. Beschreibung) in eine CSV-Datei zu speichern. Die Anwendung wurde in Java entwickelt. Die CSV-Dateien konnten dann mit weiteren Mitteln, die zum sog. Tool-Chain von „Atlas-prozessor“ gehören, verarbeitet werden. So wurden mehrere GNU-Werkzeuge zum Verarbeiten der CSV-Dateien verwendet, sowohl interaktiv, als auch in Shell-Skripten. Zum Visualisieren der Daten wurde die Software “Calc” aus dem “Openoffice.org”-Paket benutzt. Diese konnte die CSV-Dateien einlesen und in Form einer Tabelle anzeigen. Eine weitere Verarbeitung mittels einer Makroanwendung fand im “Calc” statt. Diese diente der übersichtlicheren Visualisierung. Die „Openoffice.org“ Makroanwendung hat die Tabelle sortiert und mit Farben versehen.

## 3.3 Verwendete Kodiersyntax

Es wurde eine Methode entwickelt, mit deren Hilfe sich der Entwurfsprozess beschreiben liess. Dies geschah mit Codes, die einer gewissen Syntax unterliegen. Diese Syntax ist an die von S. Salinger [35] [36] verwendete angelehnt, da sich mit ihrer Hilfe Aussagen, die aus einem Subjekt, Objekt und Verb bestehen, einfach und hinreichend genau kodieren lassen.

Die Syntax baut sich wie folgt auf:

```
<DesignCode> ::= <Subject>.<Predicate>
<Subject> ::= <P1> | <P2>
<Predicate> ::= <Proposal> | <Agreement> | <Disagreement> |
<Amendment> | <Challenge> | <ITDC>
```

Zum Beispiel würde ein Vorschlag bzgl. eines Entwurfs, der von der Person 1 kommt, wie folgt kodiert werden: `P1.Proposal`. Die weitere Beschreibung der verwendeten `<Predicate>` erfolgt im Kapitel „Ergebnisse“.

## 3.4 Verlauf der Forschung

Die Diplomarbeit begann mit der Einarbeitung in das Gebiet der GT, um die Grundlagen der Methodik nachzuvollziehen, sowie dem Erkunden der bisherigen Arbeiten auf dem Gebiet der Paarprogrammierung in der Arbeitsgruppe Software Engineering (AGSE) an der FU Berlin. Von besonderem Interesse waren Arbeiten von S. Salinger und L. Plonka. Die von S. Salinger erarbeitete Basiskonzeptmenge diente als direkte Grundlage (sowohl methodisch als auch fachlich) für diese Arbeit [36].

Die erste Aufgabe bestand darin, sich zunächst das Anwenden der BKM anzueignen. Dies erwies sich als eine nur scheinbar einfache Aufgabe, da die korrekte Vergabe der Codes aus der BKM ein möglichst genaues Verstehen der angetroffenen Situation voraussetzt, vor allem in Bezug auf den Kontext der Situation. Dazu mußte ein sytematisches Interpretieren der Situationen vorgenommen werden. Die Korrektheit der Verwendung der BKM-Codes wurde durch S. Salinger zwei Mal in längeren, gemeinsamen Sitzungen überprüft.

Darüber hinaus erfolgte die Vorstellung der Arbeit im Forschungsseminar der AGSE, was zu mehreren Hinweisen und Ideen, die vor allem die methodische Seite der Arbeit betrafen, führte. Hier erwiesen sich zwei Hinweise als besonders hilfreich:

- von Anfang an den Blick auf das Design zu richten und die Situationen unter diesem Blickwinkel zu interpretieren
- so schnell wie möglich übergeordnete Konzepte“ zu finden

Anschliessend wurde über einen längeren Zeitraum die Datenbasis offen kodiert. Eine Filmquelle wurde vorab bereits von S. Salinger kodiert. Dies erfolgte mit der von ihm entwickelten allgemeinen Kodierung (BKM). Diese Quelle und die Kodierung wurden überprüft und, wenn nötig, mit Blick auf das Softwaredesign nachgebessert. Während der Phase der offenen Kodierung wurden Indizien gesammelt, die behilflich sein könnten, übergeordnete Konzepte zu erstellen, die sich für den Entwurfsprozess in einer Paarprogrammiersitzung als relevant herausstellen könnten. Dies erfolgte durch Anwendung von Techniken, die die theoretische Sensibilität erweitern sollten.

Um die Nachvollziehbarkeit durch Dritte und die Reproduzierbarkeit der Daten zu gewährleisten, wurde am Anfang die Aufmerksamkeit auf objektive und deskriptive Merkmale gerichtet. So wurde u.a. die Form der verwendeten Sätze (z.B. Satzart,

Modus, Person) untersucht. Mit Hilfe dieser deskriptiven Konzepte wurde die Datenbasis kodiert, um u.a. die Meinungsführerschaft zu untersuchen. Aus der rein deskriptiven Beschreibung ergaben sich u.a. jedoch die folgenden Probleme. Zum einen war die erzeugte Datenmenge sehr gross, und ihre Erstellung nahm viel Zeit in Anspruch. Zum anderen fehlte die Möglichkeit, die Daten einfach zu visualisieren. Darüber hinaus liessen sich anhand der Beschreibung mit Hilfe der rein deskriptiven Konzepte keine Regelmäßigkeiten im Entwurfsprozess erkennen.

Diesem Schritt folgte der Versuch, die für den Softwareentwurf relevanten Sitzungsteile zu identifizieren. Zunächst wurde die Annahme erstellt, dass die Sitzungsteile, die dem selben Entwurfszweck dienen, einer „Episode“ zugeordnet werden können.

Diese Episoden wurden mit Hilfe eines „top-down“-Vorgehens gefunden. Dem folgte, nach den Anweisungen des Betreuers, eine „bottom-up“-Untersuchung der relevanten Sitzungsfragmente, um die methodische Korrektheit zu gewährleisten. Diese Untersuchung wurde anhand der offenen Kodierung mit den BKM-Codes durchgeführt, wobei das Kriterium der gleichen Zweckmäßigkeit weiter eingehalten wurde. Dank dieses Vorgehens konnte das Konzept *DesignEpisode* entwickelt werden.

Zur selben Zeit wurde die Software „Atlas-prozessor“ zum Zweck der Datenvisualisierung entwickelt. Diese ermöglicht es, die Daten aus „atlas.ti“ automatisch so zu bearbeiten, dass die Episoden und deren Struktur (BKM-Codes, deskriptive Codes) gleichzeitig sichtbar wurden.

Dank dieser Visualisierungsoftware ließen sich einfacher Strukturen erkennen, die die Forschungsarbeit entscheidend weiterbrachten. So fiel es z.B. auf, dass es einseitige und interaktive Episoden gibt, da in jeder dieser Untermengen ähnliche Strukturen auftauchten.

Diese zwei Untermengen wurden dann weiter ausdifferenziert. Bei diesem Schritt trat die Schwierigkeit auf, mit Hilfe der BKM- und der eigenen, deskriptiven Codes die Episoden weiter voneinander zu unterscheiden. Dies war der Anlass dafür, andere, interpretative Codes zu verwenden. Dabei wurden die Namen der bereits verwendeten Codes meistens beibehalten. Der Blick änderte sich jedoch von einem rein deskriptiven auf einen, der den situativen Kontext für die Beschreibung der Situation verstärkt berücksichtigt. Dabei ging es vor allem um die Interpretation der einzelnen Ereignisse, um ihre Rolle für die Entstehung des Entwurfs besser zu charakterisieren.

Der Übergang von der einen in die andere Kodierung war weich, und die Konzepte wurden nach und nach weiterentwickelt. Das Ziel war es, eine Beschreibung zu entwickeln, die der verwendeten Datenbasis gerecht wäre. Parallel mit der Entwicklung der Konzepte wurde die Ausdifferenzierung der Episoden betrieben. Die beiden Tätigkeiten wurden iterativ durchgeführt, da die beabsichtigte Definition der Episodentypen die Gestalt der Konzepte beeinflusste und andersrum die Definition der Konzepte auch die Form der Episoden beeinflusste. Auf diesem Wege entstand der Kern der Ergebnisse dieser Arbeit: Episodentypen.

# 4 Ergebnisse

## 4.1 Konzepte

Zur Beschreibung des Entwurfsprozess bei der Paarprogrammierung wurden mehrere Konzepte entwickelt und verwendet. Diese Konzepte sind angelehnt an die der Basiskonzeptmenge. Die entwickelten Konzepte haben teilweise einen interpretativen Charakter. Alle beziehen sich auf die Entwicklung des Softwareentwurfs und dienen zur Beschreibung von Designepisoden, welche auch ein Konzept darstellen. Um diese Konzepte von denen der Basiskonzeptmenge eindeutiger zu trennen, wurde die Konzeptmenge *DesignIssue* genannt, und die Konzepte wenn von Nöten zusätzlich mittels Namensraum getrennt. Damit eine Beschreibung mit den *DesignIssue* Konzepten möglich ist, müssen die Sitzungen vorab mit den Codes aus der Basiskonzeptmenge beschrieben werden. Dies sollte im Zweifelsfall mit Blick auf den Softwareentwurf durchgeführt werden. Mitunter muss die Beschreibung mehrmals, iterativ, durchgeführt werden, um eine präzisere Beschreibung mit *DesignIssue*-Codes zu bekommen. Dies hat den Grund, dass beide Konzeptmengen aus interpretativen Codes bestehen, und die Interpretation der Phänomene nicht eindeutig ist. Die Interpretation der Phänomene, die bei der Beschreibung mit BKM-Codes durchgeführt wird, hilft, die Beschreibung mit *DesignIssue*-Codes durchzuführen. Diese wiederum kann dazu führen, eine geeignetere Interpretation der Phänomene mit den BKM-Codes zu finden. So kann dieser Vorgang iterativ wiederholt werden, bis eine konvergente Beschreibung mit den Codes aus beiden Mengen gefunden wird.

### 4.1.1 DesignEpisode

Es handelt sich um den Teil der Sitzung, in der der Entwurf erarbeitet wird. In jeder Episode geht es für das Paar darum, ein Problem zu lösen. Das Problem besteht darin, einen Entwurf für einen Teil des Produkts (*DesignElement*) zu erarbeiten. Dieser Teil kann in einer Episode verbal genannt werden (z.B. Name der bearbeiteten Schnittstelle oder Variable). Es kann aber auch nicht verbal, z.B. durch Zeigen auf den Bildschirm, kommuniziert werden. Die Lösung des Problems wird in den meisten Episoden erreicht.

Je nach ihrer Art (z.B. "Weiterentwicklung der Funktionalität", "Neuentwicklung der Funktionalität") enthält eine Sitzung meistens mehrere Phänomene, die als *DesignEpisode* bezeichnet werden können. So ist der Anteil der gesamten Episodendauer am höchsten in Sitzungen vom Typ „Neuentwicklung der Funktionalität" in Bezug auf die Sitzungsdauer. Die Sitzung der Art „Debugging" enthält wiederum nur einen geringen Anteil an Designepisoden.

*DesignEpisode* ist eine abstrakte Bezeichnung für *DesignSequenz*. Es betont eine zusammenhängende und abgeschlossene Einheit, die eine Episode bildet. Sie ist als Synonym der *DesignSequenz* zu verstehen.

### 4.1.2 DesignSequenz

Als eine *DesignSequenz* wird eine Sequenz von zeitlich nacheinander folgenden Phänomenen bezeichnet, die sich auf einen Teil des Produkts bezieht, welcher in der Sequenz mindestens ein Mal benannt wird. Die einzelnen Phänomene können mit einem BKM-Code beschrieben werden. In jeder Sequenz muss mindestens ein Mal ein neuer Lösungsansatz zu einem Entwurfsproblem gestellt werden. Dieser gestellte Lösungsansatz läßt sich mit Hilfe vom BKM-Code *Px.propose_design* beschreiben. Dieser Name betont die Struktur des Konzeptes.

Die Vorgehensweise zum Aufbauen einer *DesignSequenz* läßt sich wie folgt beschreiben:

1. Finde in der Beschreibung mit den BKM-Codes ein Exemplar vom Code *Px.propose_design* (Vorstellung eines Lösungsansatzes für ein Entwurfsproblem) und markiere dieses als Startpunkt.

2. Ermittele das Objekt bzw. das Entwurfsproblem des Lösungsansatzes.

3. Bewege dich in der Beschreibung mit den BKM-Codes im Verlauf der Sitzung rückwärts, um den Anfang der Sequenz zu ermitteln. Um dies zu realisieren, finde den ersten BKM-Code, der sich auf das Entwurfsproblem bezieht, wobei die Beschreibung zeitlich zusammenhängend sein muss.

4. Gehe zurück zum Startpunkt.

5. Bewege dich in der Beschreibung mit den BKM-Codes im Verlauf der Sitzung vorwärts, um das Ende der Sequenz zu ermitteln. Um dies zu realisieren, finde den letzten BKM-Code, der sich auf das Entwurfsproblem bezieht, wobei die Beschreibung zeitlich zusammenhängend sein muss.

### 4.1.3 Actor

Ein *Actor* (Akteur) bezeichnet einen der Programmierer. In der Datenbasis wurden nur professionelle Programmierer betrachtet. Einen professionellen Programmierer charakterisiert u.a. die berufliche Erfahrung und die Kenntnis des Projekts, das in der Sitzung bearbeitet wird. Die Kenntnis des Projekts ist vielschichtig. So kann der Verlauf einer Sitzung sowohl von der Kenntnis des aktuell bearbeiteten Produktteils abhängen, als auch von der Kenntnis des gesamten Projekts.

Die berufliche Erfahrung wird in zwei Klassen aufgeteilt, die als Experte oder Anfänger gekennzeichnet wurden. Eine Person wird als Experte eingestuft, sobald er über eine Berufserfahrung von mehreren Jahren verfügt.

### 4.1.4 Proposal

Ein *Proposal* (Vorschlag) ist ein vorgeschlagener Lösungsansatz für ein Entwurfsproblem. Der Vorschlag bezieht sich auf ein *DesignElement*, welches ein Entwurfsproblem

abbildet. Ein Proposal wird von einem Akteur in einer *DesignEpisode* gestellt. Das Ziel eines Vorschlags ist es, die Lösung eines Entwurfproblems aktiv zu beeinflussen bzw. zu gestalten. Es wird angenommen, dass bei dem Vorschlagenden die Erwartung besteht, dass sein Vorschlag korrekt ist und akzeptiert wird. Die Nennung eines Vorschlags ist der letzte Schritt auf dem Problemlösungsweg des Vorschlagenden. Der vorgestellte Lösungsansatz ist das Objekt eines Vorschlags. Das Objekt eines Vorschlags ist meist explizit und konkret bestimmt. In solchen Fällen wird der Lösungsansatz verbal genannt und beschrieben. Manchmal muss jedoch das Objekt des Vorschlags aus dem Verlauf der Epsiode interpretiert werden. Dies findet man u.a. in den Situationen, in denen die Programmierer das Objekt des Vorschlags mit Demonstrativpronomen beschreiben.

Ein Vorschlag kann explizit oder implizit gestellt werden. Ein expliziter Vorschlag drückt direkt den Wunsch aus, den vorgestellten Lösungsansatz umzusetzen. Ein Indiz dafür kann u.a. die Verwendung von Modalverben in Aussagesätzen oder das Formulieren von Aufforderungssätzen sein.

Wenn ein Vorschlag implizit geäußert wird (z.B. durch Verwendung einer Aufforderung), ist es mitunter nicht direkt zu merken, dass der Kommunikator die geäußerten Informationen in der Intention eines Lösungsansatzes äußert, und dass beim Kommunikator der Wunsch besteht, die genannten Informationen in Form einer Lösung umzusetzen. Meistens enthalten Äußerungen, die als implizite Vorschläge klassiert werden, Begründungen für einen Lösungsansatz, der implizit vorgeschlagen wird.

Vorschläge, die explizit gestellt werden, ähneln dem BKM-Code *Px.propose_design*, vor allem wenn das Objekt des Vorschlags explizit beschrieben wird.

Ein Sonderfall des Vorschlags bildet der Widerspruch (siehe: *Disagreement*), hier wird eine Lösung vorgeschlagen, die anders ist als die, der widersprochen wird. Der Lösungsansatz selbst wird jedoch nicht genannt.

### 4.1.5 Agreement

Ein *Agreement* bedeutet die Zustimmung des Partners, einen gegebenen Vorschlag umzusetzen. Mit der Zustimmung ist eine zustimmende Rolle eines Phänomens gemeint. Die Zustimmung entweder explizit oder implizit sein, d.h. die Rolle der Zustimmung kann sich explizit in einer Aussage manifestieren oder nicht. Explizite Zustimmungen sind denen ähnlich, die man mit dem BKM-Code *Px.agree_design* beschreiben kann. Sie werden z.B. durch verbale oder nonverbale Bejahung ausgedrückt.

Implizite Zustimmungen sind solche, in denen ein Ereignis keine Zustimmung ausdrückt, wenn der Kontext von diesem Ereignis nicht berücksichtigt wird. Wenn man jedoch seinen Kontext und die Rolle im Entwurfsprozess betrachtet, bestätigt dieses Ereignis die Lösung, und der Ansatz wird umgesetzt. So findet eine implizite Zustimmung spätestens in dem Moment statt, in dem der Vorschlag umgesetzt wird.

### 4.1.6 Disagreement

Ein *Disagreement* (Widerspruch) ist ein Ausdruck für den Lösungsansatz, der in *Proposal* gestellt wurde. Ein Widerspruch kann auch als ein Vorschlag aufgefasst werden (Untermenge von *Proposal*). Er ist in diesem Fall wie folgt zu verstehen: durch einen Widerspruch wird eine Lösung vorgeschlagen, die anders ist als die, die ursprünglich vorgeschlagen wurde. Der Widerspruch kann explizit oder implizit sein. Implizite Widersprüche können u.a. durch einen beabsichtigen Abbruch der Designsepisode geschehen. Eine andere Art von impliziten Widersprüchen bilden Begründungen für Vorschläge, die sich mit dem gegebenen Vorschlag nicht vereinbaren lassen.

### 4.1.7 Amendment

Ein *Amendment* beschreibt einen Vorschlag, der sich auf einen anderen, zuvor genannten Vorschlag bezieht und diesen erweitert. Die Erweiterung widerspricht fachlich dem Leitfaden des ursprünglichen Vorschlags nicht. Sie bestätigt diesen und fügt ein neues Element hinzu. Meistens werden durch die Erweiterung mehr Details vom ursprünglichen Lösungsansatz ausgedrückt. Dies kann entweder durch Verfeinerung des ursprünglichen Lösungsansatzes oder durch das Hinzufügen eines neuen Elements geschehen. Es kann also als gleichzeitiges Auftreten der Codes *Agreement* und *Proposal* mit einem Bezug auf einen anderen *Proposal* gedeutet werden.

### 4.1.8 Challenge

Ein *Challenge* ist ein Vorschlag, der sich auf einen anderen, zuvor genannten Vorschlag bezieht und diesem mindestens teilweise widerspricht. Somit ist ein *Challenge* ein gleichzeitiges Auftreten der Codes *Disagreement* und *Proposal* mit einem Bezug auf ein anderen *Proposal* zu verstehen.

### 4.1.9 DesignElement

Ein *DesignElement* ist ein konkretes produktbezogenes Element, von dem der Entwurf abhängt. Es werden hier hauptsächlich Elemente betrachtet, die aus dem Gebiet der objektorientierten Ansätze stammen (z.B. Klasse `X` oder Methode `a(X param)`).

### 4.1.10 IntroductionToDesignContext (ITDC)

Eine *ITDC* (Einführung in den Entwurfskontext) bezeichnet ein Phänomen, in dem Wissen explizit kommuniziert wird. Es ist eine Sequenz von zeitlich nacheinanderfolgenden BKM-Codes, die das Ziel hat, zusätzliche bzw. ergänzende Informationen über den Kontext des aktuellen Problems der Episode mitzuteilen. So führt der Kommunikator den Partner in den Kontext eines (Entwurfs-)Problems ein. Oft enthalten die *ITDC*s Begründungen für einen Vorschlag bzw. zusätzliche Informationen über das Produkt oder Projekt. Die *ITDC* kann interaktiv sein. Solche Einführungen in den Kontext

bestehen nicht nur aus Aussagesätzen oder anderen informativen Äußerungen, sondern können u.a. Fragen des Partners beinhalten. Zum Subjekt des Codes wird die Person ernannt, bei der die Rolle des Kommunikators der meisten revelanten Informationen festzustellen ist. Innerhalb der BKM-Codes, die eine Einführung bilden, sind vor allem *Px.*_knowledge* und *Px.*_standard of knowledge* zu finden.

## 4.2 Designepisoden

Die Arbeit befasst sich mit der Beschreibung der Teile der Paarprogrammiersitzung, in denen Softwareentwürfe erstellt werden (siehe Konzepte). Diese Sitzungsteile wurden Designepisoden genannt.
Der Beschreibung folgte eine Klassifizierung der Designepisoden. Im Rahmen dieser Arbeit ließen sich fünf Episodenklassen (auch Episodentypen genannt) eruieren, die sich wie folgt gliedern.

### 4.2.1 Episodentyp: Single-Handed

Dieser Episodentyp ist durch ein asymmetrisches Verhältnis beider Programmierer gekennzeichnet, wobei einer die aktive und einer eine stark passive Rolle einnimmt. Es findet sozusagen ein „Alleingang“ („Single-Handedness“) der aktiven Person bei der Problemlösung statt, da nur sie sich aktiv an der Gestaltung des Entwurfs beteiligt. Wer die Rolle der aktiven Person übernommen hat, kann erst nach Beendigung der Episode abschließend festgestellt werden. Die aktive Person schlägt selbständig, meistens ohne jegliche Begründung, einen Lösungsansatz vor. Dieser wird häufig implizit oder auch explizit übernommen. Die zweite (nicht aktive) Person begrenzt sich meistens auf Aussagen, die wenig Bedeutung für die Gestaltung des Entwurfs haben (z.B. flüstern, wiederholen des von der aktiven Person Gesagten).

Dieser Episodentyp läßt sich exemplarisch in den Tabellen 2 bis 5 darstellen. Die aktive Person hat in allen vorgestellten Situationen das Sagen. Der Prozess der Problemlösung wird von ihr (fast) allein durchgeführt. Der passiven Person wird meistens nur das Ergebnis (Lösungsvorschlag) mitgeteilt. So z.B. erwidert die passive P1 auf einen konkreten Ansatz der P2 mit einem undeutlichen Flüstern, welches aufgrund eines fehlenden aktiven Einspruchs als passive Zustimmung aufgefasst werden kann (Tabelle 2; Äußerung 2).

Eine andere Situation stellt sich in der Tabelle 3 dar. Die aktive P2 kennt vermutlich bereits den entsprechenden Lösungsansatz, da sie von sich aus den zu bearbeitenden Abschnitt vorschlägt und gleich die Methode benennt (Tabelle 3; Äußerung 1 und 2). Die passive Person bestätigt die Äußerung durch simples Wiederholen, ohne dass es im Verlauf zu einer weiteren Interaktion käme (Tabelle 3; Äußerung 4 bezieht sich nicht explizit auf Äußerung 3). Da die aktive Person die Methode sofort umsetzt (Tabelle 3; Äußerung 4), läßt sich vermuten, dass die Zustimmung der passiven Person die Lösung nicht beeinflusste, es konnte jedoch dazu führen, dass die aktive P2

den Methodennamen nicht weiter verifizierte. In der Tabelle 4 wird die Lösung ebenso selbständig von P1 erarbeitet. Die anschliessende Äußerung von P2 bestätigt den Lösungsvorschlag. Dies geschieht jedoch erst nach einer kurzen Denkpause von P2, was ein Hinweis dafür sein kann, dass P2 nicht parallel und separat an einem ähnlichen Vorschlag arbeitete, sondern den Lösungsvorschlag von P1 nachvollzogen hat (Tabelle 4; Äußerung 3). In der Tabelle 5 spitzt sich dieser Episodentyp insoweit noch zu, als dass die aktive P2 die gesamte Arbeit übernimmt, ohne dass es von der passiven Person zu irgendeiner Reaktion käme. Dies wird von P2 nicht zum Anlass zur Nachfrage genommen, ob für das Ausbleiben der Reaktion ein fehlendes Verständnis oder eine Zustimmung zum Geschehen zu Grunde liegt. Dieses Beispiel stellt daher eine Maximalvariante des Alleingangs dar.
Die Gründe, diesen Episodentyp zu wählen, können je nach Sitzung unterschiedlich sein.
Einer der Gründe kann die niedrige Einschätzung der Schwierigkeit und/oder Relevanz der Episode für das Erledigen der Ziele der Sitzung sein.
Dies ist in den Tabellen 2, 4 und 5 zu finden. In den Tabellen 2 und 5 initiiert P2 am Anfang der Sitzung die „Alleingang“-Episode. In beiden Beispielen erstellt P2 durch seine Vorschläge einen ungenauen Lösungsansatz. Die weitere Ausarbeitung dieses Ansatzes kann diesen verändern. Somit ist die wahrgenommene Relevanz der Episode für P2 gering. In der Tabelle 2 betont P2 die niedrige Relevanz des Lösungsansatzes durch das Verwenden des Adjektivs „einfach“ als Beschreibung des Schwierigkeitsgrads des Problems in der Episode (Tabelle 2; Äußerung 1). In der Tabelle 4 wird der Rückgabewert einer neuen Methode festgelegt. Die angedachte Funktionalität und die Schnittstelle der Methode wurden gemeinsam entwickelt. Diese Methode ist eine sogenannte Fabrikmethode, deren einzige Aufgabe es ist, ein neues Objekt einer bestimmten Klasse zu erzeugen. Somit ist der Aufwand der Implementierung dieser Methode gering. Nach dem die Funktionalität und die Schnittstelle dieser Methode bestimmt wurden, ist es nachvollziehbar, dass P1 das Problem als einfach eingestuft hat.
In der Tabelle 3 schlägt P2 vor, den Zugriffsmodifikator einer Methode zu ändern. Die betroffene Methode wurde in der Sitzung gemeinsam entwickelt. Diese Änderung führt zur Änderung der Schnittstelle der entwickelten Klasse, was die Architektur der Software potenziell beeinflusst. In seinem Vorschlag betont P2 jedoch den technischen Aspekt seines Lösungsansatzes. Dieser beruht darauf, dass der Zugriffsmodifikator als Teil der Programmiersprache genannt und der architektonische Aspekt in den Hintergrund gestellt wird. Es ist naheliegend, dass für erfahrene Programmierer (wie P2) Programmiersprachen keine besonders grosse Herausforderung bilden. Dieser Lösungsansatz wurde deswegen vermutlich von P2 als wenig relevant und einfach eingestuft.
Die mangelnde Erklärung des Kontexts der Lösung verweist darauf, dass die aktive Person es nicht bedeutend fand, den Kontext mitzuteilen. Das läßt sich anhand der Daten auf zwei Arten deuten. Einerseits geht die aktive Person möglicherweise davon aus, dass die passive den Kontext der Lösung ausreichend kennt, um den Lösungsansatz zu verstehen. Hierbei kann eine explizite Zustimmung der passiven Person ein Hinweis dafür sein, dass diese den Ansatz verstanden hat und dem zustimmt. Eine solche Situ-

ation lässt sich in der Tabelle 4 finden.
Andererseits kann die aktive Person den potenziellen Verständnismangel ignorieren. Solch eine Situation findet man in den Tabellen 2 und 3. Beide Beispiele gehören zu einer Sitzung, in der das Paar aus einem Anfänger (6 Monate Berufserfahrung) und einem Experten (10 Jahre Berufserfahrung) besteht. Daher kann man annehmen, dass die Unterschiede der Fähigkeiten und Kenntnisse des bereits fortgeschrittenen Projekts groß sind. Trotzdem wählt P2 den „Alleingang“ und teilt P1 den Kontext der Lösungsansätze nicht mit.

Im „Alleingang“ wird das Problem allein von der aktiven Person gelöst. Der Weg zur Lösung wird der passiven Person nicht mitgeteilt.
Diese kann bei vergleichbaren oder besseren Fähigkeiten und Kenntnissen der Projektumgebung den Weg erahnen. Bei schlechteren Fähigkeiten und Kenntnissen kann weder der Weg und noch die Lösung nachvollzogen werden. Somit hängt die Nachvollziehbarkeit der Lösung bei der passiven Person von der Sitzung ab. In den Sitzungen, in denen beide Programmierer die Projektumgebung gut kennen und deren Programmierfähigkeiten vergleichbar sind, können beide die Problemlösung sowie deren Hintergründe in der Episode eher nachvollziehen. In Sitzungen mit einer unausgewogenen Konstellation des Paars (unterschiedliche Fähigkeiten/Kenntnisse), in der die Rolle der passiven Person durch den unerfahreneren der beiden Programmierer eingenommen wird, besteht die Gefahr, dass die passive Person den Problemlösungsweg nicht nachvollziehen kann. Diese Episode begrenzt die in der Literatur beschriebenen Effekte der Paarprogrammierung in einer Programmiersitzung, wie es die folgenden Ausführungen näher zeigen.

Die Bewertung der Korrektheit der Lösung (”continuous review”) kann wegen der kurzen Dauer kaum durchgeführt werden. Nur in ausgeglichenen Paaren (ähnliche Fähigkeiten/Kenntnisse), bei denen beide Personen die Relevanz und den Schwierigkeitsgrad des Problems als niedrig einschätzen, kann es zu einer erfolgreichen Code-Durchsicht der Lösung kommen.

Der Lerneffekt in diesem Episodentyp ist ebenso begrenzt. Das Problem wird zu kurz betrachtet, als dass es zu einem bedeutenden Lerneffekt kommen könnte. Darüber hinaus werden die Gründe der Lösung nicht angegeben. In unausgeglichenen Paaren kann die passive Person kaum etwas über das Design im Projekt lernen, wenn die fähigere Person die aktive Rolle annimmt. In solchen Situation werden vor allem kulturelle Aspekte der Organisation vermittelt.

Die Lösung in einer solchen Episode wird schnell erzielt. Die Leistung des Paars bleibt hoch, da u.a. keine Koordinationverluste auftreten. Die Leistung vom Paar in der Episode ist aber vergleichbar mit der einer einzelnen Person. Grund dafür ist die Inaktivität der passiven Person, wenn es um die Entwicklung eines Lösungsansatzes geht. Die *Single-Handed*-Episoden können aber zu motivationalen Problemen bei der passiven Person führen. Sie ist an dem Lösungsansatz kaum beteiligt und kann sich somit wenig für den Erfolg vom Paar verantwortlich fühlen.

Da der vorgestellte Lösungsansatz nicht mehr in der Episode geändert wird, läßt sich darauf schliessen, dass die Qualität der Lösung im Paar vergleichbar ist mit der

einer separat gestellten Lösung. Eine explizite Zustimmung kann ein Hinweis darauf sein, dass die Qualität der Lösung von beiden akzeptiert wird.

| # | Äußerung | *BKM Code* | *DesignIssue Code* |
|---|---|---|---|
| 1 | P2: Und diese... die Methoden... die würden wir einfach ma auskommentieren... | *P2.propose_design* | *P2.Proposal* |
| 2 | P1: flüstert undeutlich | *P1.mumble_sth* | |

Tabelle 2: Es wird eine Klasse und ihre Methoden angeschaut. Anhand der betrachteten Klasse soll eine neue Klasse entwickelt werden. Die Situation findet am Anfang der Sitzung statt. (FA03 [1:585]; Driver: P2; Aktive Person: P2)

| # | Äußerung | *BKM Code* | *DesignIssue Code* |
|---|---|---|---|
| 1 | P2: Jetzt würd' ich gerne, die eine Methode zu..... | *P2.propose_step* | *P2.Proposal* |
| 2 | P2: Wie hieß die noch gleich? `findTeleporter`? | *P2.explain_standard of knowledge* / *P2.explain_knowledge* | |
| 3 | P1: `findTeleporter` | *P1.agree_knowledge* | |
| 4 | P2: Machen wir alles mal `public`! | *P2.propose_design* | |

Tabelle 3: Eine neue Klasse `FerryPolygonChecker` wird entwickelt. Die Entwicklung der Klasse hat bereits angefangen.(FA03 [1:605]; Driver P2; Aktive Person: P2)

### 4.2.2 Episodentyp: Guidance

Die Episodenklasse *Guidance* (Führung) ist ein Episodentyp, in dem eine der Personen einen Lösungsvorschlag gibt, der durch zusätzliche Kontextinformationen vom Vorschlaggeber bereichert wird. Die zusätzlichen Informationen sollen dem Partner den Problemlösungsweg näher bringen.
Die Beziehung im Paar ist in diesem Episodentyp asymmetrisch. Es können die zwei Rollen *aktiv* und *passiv* gefunden werden. Wer welche Rolle übernommen hat, kann erst nach dem Beenden der Episode endgültig festgestellt werden. Der Grund dafür ist der inhaltlicher Bezug der *DesignIssue* Codes *Px.IntroductionToDesignKontext* und *Px.Proposal*.

Die Aktivität der *aktiven* Person beinhaltet zwei Unteraktivitäten:

- Der passiven Person den Designkontext näher bringen (*Px.ITDC*).

| # | Äußerung | *BKM Code* | *DesignIssue Code* |
|---|---|---|---|
| 1 | P1: Dann machen wir einfach dieses hier. | *P1.propose_design* / *P1.amend_activity* / *P1.write_sth* | *P1.Proposal* |
| 2 | P1: Da sind wir ein Stück weiter. | *P1.explain_completion* | |
| 3 | P2: Uhm.. Ja. | *P2.agree_design* / *P2.agree_completion* | |

Tabelle 4: Das Paar bearbeitet eine Klasse, die eine Schnittstelle implementieren soll. Der Rumpf der Methode wurde automatisch von der IDE erstellt. Das Paar möchte den Rückgabewert der Methode setzen. Die hinzugefügte Methode `createSplitObjectAction` gibt lediglich ein neues Objekt der Klasse `SplitObjectAction` zurück. (FA04 [1:21]; Driver: P1; Aktive Person: P1)

| # | Äußerung | *BKM Code* | *DesignIssue Code* |
|---|---|---|---|
| 1 | P2: Also die `FerryIDChecker` - viele Sachen bleiben ja vielleicht auch nicht. | *P2.propose_strategy* / *P2.propose_design* | *P2.Proposal* |
| 2 | P2: Also Statistik kriegen wir vielleicht wieder welche. Wer weiss, das muss ma' noch sehen. | *P2.propose_strategy* / *P2.propose_design* | *P2.Proposal* |

Tabelle 5: Anfang der Sitzung. Die P2 kennt den Code besser und zeigt der P1 die Vorgehensweise. Sie schauen sich eine vorhandene Klasse (`FerryIDChecker`) an, die als Grundlage für die Entwicklung dienen wird. Die P2 ist sich der Details der Lösung nicht sicher, weswegen sie eine grobe Beschreibung der Lösung gibt. Von P1 erfolgt keine Reaktion.(FA03 [1:582]; Driver: P2; Aktive Person: P2)

- Lösungsansatz präsentieren (*Px.Proposal*).

Tabelle 6 zeigt eine Episode vom Typ *Guidance*, die nur aus den zwei genannten Aktivitäten besteht, und somit ein minimales Exemplar des genannten Typs ist. In dieser Situation wird nur P2 aktiv. P1 äussert sich nicht verbal. Im Paarvideo ist P1 fast bewegungslos. Er schaut auf den Bildschirm und stützt sich mit dem Ellenbogen auf den Tisch ab. P2 treibt dafür die Entwicklung voran. Er stellt einen Lösungsvorschlag vor (Tabelle 6; Äußerung 1). Die Konstellation der Programmierer der Sitzung FA03 (Anfänger-Experte) lässt vermuten, dass der Vorschlag von P1 nicht verstanden und nachvollzogen werden kann. So folgen dem Vorschlag zusätzliche Informationen (Tabelle 6; Äußerung 2). Diese erklären teilweise den Kontext des Vorschlags. P2 ahnt, warum die genannte Methode nicht notwendig ist; eine bereits vorhandene Komponente (`FerryLayer`) bildet die benötigte Funktionalität ab. Dazu erwähnt P2 kurz, wie diese Funktionalität aussieht. Die Tatsache, dass P2 diese beiden zusätzlichen Informationen erwähnt, läßt vermuten, dass diese für ihn bei der Problemlösung wichtig waren. Die Aufteilung der Rollen ist in dieser Situation eindeutig.

Die verbale Aktivität der passiven Person ist optional. In der Tabelle 6 zeigt die passive Person (P1) keine verbale Aktivität. In den weiteren Beispielen sind Aktivitäten der passiven Person zu finden. Diese tragen aber nicht direkt zum Entwurf der Software bei. Sollten bei der passiven Person verbale Aktivitäten auftreten, auf die sich die aktive Person in ihren Äußerungen bezieht, so kann man von Interaktion sprechen. Wenn diese auftritt, begrenzt sich die passive Person u.a. auf Fragen zum Designkontext oder zum Lösungsansatz, sowie auf das Wiederholen des Teils der Informationen über den Designkontext, die von der aktiven Person stammen. Der präsentierte Lösungsansatz wird übernommen. Exemplarisch für eine verbale Aktivität, der passiven Person werden die folgenden Beispiele vorgestellt.

In der Tabelle 7 begrenzt sich die Aktivität von P2 in der ganzen Episode auf eine einzige Frage (Tabelle 7; Äußerung 3), wenn man von der abschliessenden Äußerung absieht. Damit möchte er vermutlich dem Partner vermitteln, dass er ihm nicht folgen kann. Auch in den beiden weiteren Beispielen (Tabellen 8 / 9) fragt die passive P1 nach (Tabelle 8; Äußerung 1 und Tabelle 9; Äußerung 2). Mit diesen beiden Äußerungen trägt sie ebenfalls nicht konkret zur Entwicklung des Entwurfs bei; es werden weder Lösungsansätze vorgeschlagen, noch für den Entwurf relevantes Wissen kommuniziert. Die Fragen bringen ein konkretes Verständnisproblem während des Entwurfs des Produktes zum Ausdruck. Diese Äußerungen bewirken, dass die aktive Person sich auf diese Problemstellen in ihren darauffolgenden Äußerungen explizit bezieht (Tabelle 7; Äußerung 4 / Tabelle 8; Äußerung 2 / Tabelle 9; Äußerung 3).

Die Form und der Inhalt der verbalen Interaktion können für die Auswirkungen der Episode entscheidend sein.
Der interaktive Teil der Episode kann der aktiven Person helfen, schneller und/oder zu einer besseren Lösung zu kommen. Dies ist der Fall, wenn es die passive Person durch ihre Interaktion schafft, den Suchraum der Problemlösung der aktiven Person einzuschränken. Dies kann z.B. durch eine gezielte Frage geschehen. In der Tabelle 9

ist so eine Situation vorzufinden. P2 fängt die Episode mit einer Einführung in den Designkontext des Problems an. P2 beschreibt P1, wie der Konstruktor der Klasse, auf deren Basis die Entwicklung stattfindet, konzipiert ist. P1 versteht diesen Ansatz nicht ganz und stellt eine Frage, die den Gedankengang von P2 beeinflusst. Die Antwort auf diese Frage fällt P2 nicht leicht, worauf u.a. die unvollständigen Sätze deuten. Der Satz „Oder doch, er braucht für dieses `get`...“ (Tabelle 9; Äußerung 3) weist darauf hin, dass P2 seine Meinung während der Antwort mindestens ein Mal geändert hat. Die genauere Betrachtung des Problems, die von der Frage angestoßen wurde, kann entweder zu einer besseren Qualität vom Entwurf des Konstruktors oder zum schnelleren Erreichen dieser Qualität führen. Eindeutige Beweise für eine bessere Qualität oder ein schnelleres Erreichen dieser Qualität können nicht gefunden werden. Der interaktive Teil der Episode kann andererseits bei der passiven Person den Lerneffekt verstärken. Dies ist vor allem dann der Fall, wenn die konkreten Wissensmängel explizit, z.B. in Form einer Frage, ausgedrückt werden. So kann die aktive Person diesen Wissensmangel bei der passiven Person genauer bestimmen und darauf reagieren.

Die Gründe für das Auftreten dieser Episodenklasse können unterschiedlich sein. Der wichtigste Grund könnte in der Zusammensetzung des Paars bestehen. In einem unausgeglichenen Paar tritt dieser Episodentyp besonders häufig auf. Solch eine Sitzung hat für die unerfahrenere Person hauptsächlich eine Lernfunktion. Aus diesem Grund wählt die fähigere Person diesen Episodentyp aus, um dem Partner die Lösung inklusive der Zusatzinformationen mitzuteilen. Die fähigere Person muss sich jedoch bewusst sein, dass der Partner weniger Kenntnisse über das Projekt und/oder die verwendeten Werkzeuge (u.a. Programmiersprache, Programmierumgebung) verfügt. So ist es z.B. in der Sitzung FA03, aus der die Tabellen 6, 8 und 9 stammen. Am Anfang dieser Sitzung betont P2 die Ungleichheit im Paar. P2 spricht P1 wie folgt an: „Also wichtig wär' jetzt, glaube ich, dass du immer dabei bist. Und dass du sagst, wenn ich mich irgendwie verrenne oder 'was falsch mache.“ So weist P2 der P1 eine Hilfsrolle zu, anstatt sie als ebenbürtig zu erachten. In Paaren mit vergleichbaren, hohen Kenntnissen, kommt die *Guidance*-Episode seltener vor. In solchen Paaren ist der Lerneffekt der Sitzung nicht von kritischer Bedeutung. Hier wählt einer der Programmierer diesen Episodentyp eher aus dem Grund, seinen Problemlösungsweg zu verbalisieren (Tabelle 7 aus der Sitzung FA02 - Experte-Experte). So kann der Partner einfacher den Lösungsansatz verstehen und diesen notfalls korrigieren. In solchen Paaren kommt es häufiger vor, dass die passive Person die Korrektheit der Lösung verbal bestätigt.

Dieser Typ von Episode betont den Lerneffekt der Paarprogrammierung. Dies geschieht dadurch, dass die passive Person einen fundierten Einblick in den Designkontext und die aktuelle Lösung zugleich bekommt. Das Vorkommen interaktiver Teile kann somit den Lerneffekt besonders verstärken, u.a. beim Nachfragen (Abgleich des Informationsstands) oder beim Wiederholen (phonologische Schleife, Rezenzeffekt).

| # | Äußerung | *BKM Code* | *DesignIssue Code* |
|---|---|---|---|
| 1 | P2: `parseName`... Das brauch ma' nicht. | *P2.propose_design* | *P2.Proposal* |
| 2 | P2: Das macht dann der `FerryLayer` wahrscheinlich für uns, weil der muss die alle durchgehen. | *P2.explain_knowledge* | *P2.ITDC* |

Tabelle 6: Die Situation findet am Anfang der Sitzung statt. Das Paar entschied sich, eine neue Klasse anhand einer vorhandenen zu entwickeln. Daher schauen sie sich gemeinsam die vorhandene Klasse an und analysieren diese. `parseName` ist eine Methode in der ursprünglichen Klasse. Mit `FerryLayer` ist wahrscheinlich eine Komponente/ein Subsystem der entwickelten Software gemeint. (FA03[1:587]; Driver: P2; Aktive Person: P2)

### 4.2.3 Episodentyp: DirectConsensus

Die Episodenklasse *DirectConsensus* umfasst Designepisoden, in denen beide Programmierer das Problem der Episode separat lösen und annähernd gleichzeitig zur Lösung kommen. Die von beiden erzielten Lösungen sind inhaltlich deckungsgleich und werden übernommen.

Eine Beispielepisode des *DirectConsensus* ist in der Tabelle 10 zu finden. In diesem Beispiel stellen beide Programmierer Lösungsvorschläge vor (10; Äußerungen 3 und 4). Diese Lösungsansätze stimmen vollständig überein. Die Ansätze werden zu fast gleicher Zeit geäußert, worauf auch die kurze Dauer (ca. 5 Sek.) der Episoden hinweist. Das ist ein Indiz dafür, dass die Problemlösung bei beiden Programmierern separat verlaufen ist. Beide kamen auf die gleiche Lösung.

Der direkte Konsens ist auch in Situationen zu merken, in denen die Lösungsansätze beider Programmierer nicht zeitgleich, sondern zeitnah gestellt werden. So eine Situation ist im Beispiel der Tabelle 11 zu finden. Die Situation beginnt mit einer Einführung in den Designkontext. Die P1 stellt danach ihren Lösungsansatz vor. Diesem folgt eine Bestätigung seitens der P2 - sie schlägt die gleiche Lösung vor. Die P2 scheint vom Vorschlag der P1 nicht überrascht zu sein. Ihre Stimme bleibt bei dieser Äußerung verhältnismäßig tief, was einen Eindruck der (Selbst-)Sicherheit und Überzeugung vermittelt. Dies führt zu der Vermutung, dass die P1 das Problem in der Episode separat gelöst hat. Der kurze zeitliche Abstand (¡ 1 Sek.) zwischen den Äußerungen 2 und 3 führt zu der Vermutung, dass der Problemlösungsprozess bei beiden Programmierern parallel verlief.

Ein direkter Konsens ist auch in Situationen zu vermuten, in denen ein Lösungsansatz vorgestellt wird, der aus Äußerungen beider Programmierer besteht. Solch eine Situation ist im Beispiel der Tabelle 12 zu sehen. In diesem Beispiel kommen die Lösungsvorschläge von der P1, die P2 ergänzt diese jedoch. Beim ersten Vorschlag rechnet die P1 nicht damit, dass dieser Vorschlag ergänzt wird. Dies ist an der Intonation der Äußerung der P1 und an der sehr kurzen zeitlichen Entfernung der Äußerungen 1 und 2 zu merken.

| # | Äußerung | *BKM Code* | *DesignIssue Code* |
|---|---|---|---|
| 1 | P1: Ok, so... Dann muss ich ma', wenn wir die aufrecht erhalten, die `Factory` rüberziehen. | *P1.propose_design* | *P1.Proposal* |
| 2 | P1: Aber das ist dann das Problem. | *P1.explain_knowledge* | *P1.ITDC* |
| 3 | P2: Ja? | *P2.ask_knowledge* / *P2.explain_standard of knowledge* | |
| 4 | P1: Und zwar das Problem sind, wenn du die `Columns` benutzen willst, auch in `All`, musst du für einheitliche Attributspalten haben... sorgen. Und dafür musst du an der Stelle... | *P1.explain_knowledge* | |
| 5 | P1: Ach so! Nee. Halt! Die haben wir ja hier. | *P1.challenge_knowledge* | |
| 6 | P1: Das heisst, die können wir in der `Factory` auch reinmachen. | *P1.propose_design* | *P1.Proposal* |
| 7 | P1: Wir sind hier in der Konfiguration. Da müssten wir auch ein `getAllColumnAttriutes` machen können. | *P1.explain_knowledge* | *P1.ITDC* |
| 8 | P2: Aha... | *P2.mumble_sth* / *P2.explain_standard of knowledge* | |

Tabelle 7: Die Situation findet mitten in der Sitzung statt. Das Paar konnte bereits Teile von einem großen Refactoing durchführen. Dieses wurde teilweise halbautomatisch mit Hilfe von Refactoring-Werkzeugen erledigt. Der Build-Bericht weist allerdings Fehler auf. So müssen einige Stellen manuell angepasst werden. Eine solche Stelle findet das Paar in dieser Situation vor. (FA02[3:420]; Driver: P1; Aktive Person: P1)

| # | Äußerung | *BKM Code* | *DesignIssue Code* |
|---|---|---|---|
| 1 | P1: `NoTeleporterFound` ? | *P1.ask_knowledge* | *P2.ITDC* |
| 2 | P2: Ja... das ist irgendwie eine andere Bedeutung.... | *P2.explain_knowledge* | |
| 3 | P2: Pass mal auf, wir machen das jetzt `OK` und `Unknown` und fertig! | *P2.propose_design* *P2.propose_step* | *P2.Proposal* |

Tabelle 8: Es wird gerade eine Datenstruktur vom Paar bearbeitet. Diese Datenstruktur wurde zuvor aus einem anderen Codeteil kopiert. Die Entwickler überlegen sich, welche der Datenfelder übernommen werden sollen. Die Namen `NoTeleporterFound`, `OK` und `Unknown` sind Namen der Datenfelder der vorhandenen Datenstruktur. (FA03[2:408]; Driver: P2; Aktive Person: P2)

Die Intonation der Äußerung der P1 beim zweiten Vorschlag (Tabelle 12; Äußerung 4) läßt darauf schliessen, dass sie davon ausgeht, dass die P2 ihren Lösungsansatz ergänzen wird. Dies ist wieder an der Intonation der Äußerung der P1 zu merken. Die P1 endet ihre Äußerung mit erhöhter Stimmlage. Die P2 reagiert darauf und ergänzt den Lösungansatz der P1. Die P1 stimmt dieser Erweiterung gleich zu. Dies weist darauf hin, dass die P1 den gleichen Lösungsansatz stellen wollte. Da in dieser Episode die Koordination kaum vorhanden ist, mussten die beiden Personen ihre Lösungsansätze parallel und separat erarbeitet haben.

Das Auftreten der Episoden dieser Episodenklasse ist ein Zeichen für einen vergleichbaren Wissenstand bezüglich des Objekts der Episode. So treten derartige Episoden häufiger bei Paaren auf, die eine ähnliche Berufserfahrung vorweisen. Bei Paaren mit stark unterschiedlichen Kenntnissen / Fähigkeiten treten derartige Episoden häufiger in späteren Teilen der Sitzung auf und sind damit ein Indiz für einen stattgefundenen Lernprozess. Paare, die ähnliche Fähigkeiten aufweisen, können solche Episoden einleiten, um eine implizite Bestätigung des Lösungsansatzes zu bekommen. Die Episode vom Typ *DirectConsensus* tritt vor allem in Situationen auf, in denen die beiden Programmierer das Problem der Episode nicht als schwierig wahrnehmen. Ein Hinweis darauf sind die verhältnismäßig knappen Äußerungen mit Lösungsansätzen und die meist fehlenden Einführungen in den Designkontext.

Die Leistung des Paars in einer Episode vom Typ *DirectConsensus* ist vergleichbar mit der Leistung einer einzelnen Person, da in diesem Episodentyp die gleiche Lösung zu gleicher Zeit von beiden Programmierern erarbeitet wird. Die Leistung wird zusätzlich potenziell um die Koordinationverluste gesenkt. Die Tatsache, dass beide Programmierer bei ihrer Problemlösung zur gleichen Lösung kamen, ist ein Indiz dafür, dass die Lösung hochqualitativ ist. Dies geschieht unter der Annahme, dass beim gleichen Lösungsansatz beider Programmierer eine hochqualitative Lösung wahrscheinlicher ist, als eine niedrigqualitative. Die motivationalen Auswirkungen dieser Episodenklasse sind vor allem in der Paarkonfiguration Anfänger-Experte merkbar. So kann es der

| # | Äußerung | *BKM Code* | *DesignIssue Code* |
|---|---|---|---|
| 1 | P2: Dann wäre die Frage, was er eigentlich braucht, um seine Arbeit zu machen? Also, er hat da irgendwie so 'ne Methode `checkAlles`. Da ist die Rolle vom `rtgMapData` und `ferryPolygon` vertauscht. Genau. | *P2.explain_knowledge* | *P2.ITDC* |
| 2 | P1: Brauchen wir das `MapData`, oder nicht? | *P1.ask_knowledge* | |
| 3 | P2: Also wenn er z.B. eine Fähre checken will... Oder doch, er braucht für dieses `get`... Da gibt es die Methode, die zu einem Teleporter den Von-, Über-, Nach-Knoten und auch hier drüben auch den Von-Knoten... Und dafür braucht er den `rtgMapData` | *P2.explain_knowledge* | |
| 4 | P2: Dann würd' ich sagen, wir kriegen den `rtgMapData` und ein `ferryLayer` rein und gut! | *P2.propose_design* | *P2.Proposal* |
| 5 | P2: Dann hat er die beiden zur Verfügung. So mach' ma das! | *P2.explain_knowledge* | *P2.ITDC* |

Tabelle 9: Es wird gerade der Konstruktor der Klasse `FerryPolygonChecker` bearbeitet. Diese Klasse basiert auf der Klasse `FerryIDChecker`. Die entwickelte Software ist ein Geoinformationssystem, in dem `Ferry` (Fähre), `Polygon` und `MapData` eine Rolle spielen. Die genaue Bedeutung und vor allem die genaue Beschreibung der Funktionalität dieser Begriffe ist aus den Daten nicht nachvollziehbar. (FA03 [1:584]; Driver: P2; Aktive Person: P2)

Anfänger, die gleiche Lösung wie der Experte erarbeitet zu haben, als Erfolg und Anerkennung für sich deuten. Diese beiden Erlebnisse können als motivationsfördernd angesehen werden.

| # | Äußerung | *BKM Code* | *DesignIssue Code* |
|---|---|---|---|
| 1 | P1: `searchTeleporter` ist bei uns inzwischen `findTeleporter` ... | *P1.explain_knowledge* | *P1.ITDC* |
| 2 | P2: Genau, Richtig! | *P2.agree_knowledge* | *P2.Agreement* |
| 3 | P1: ... also [*undeutlich*] rausschmeißen | *P1.propose_design* | *P1.Proposal* |
| 4 | P2: Also weg damit! | *P2.propose_design* | *P1.Proposal* |

Tabelle 10: Das Paar entwickelte in der Sitzung eine Klasse. Diese Klasse wurde durch Anpassung einer vorhandenen entwickelt. Am Anfang der Sitzung wurde ein Teil, der vermutlich unnötige Funktionalitäten enthielt, auskommentiert. Die Situation findet am Ende der Sitzung statt. Das Paar kann jetzt die auskommentierten und unnötigen Codefragmente löschen. (FA03 [3:248]; Driver: P1)

| # | Äußerung | *BKM Code* | *DesignIssue Code* |
|---|---|---|---|
| 1 | P2: Ja, das ist für diese Variable 'ne Erklärung. | *P2.explain_knowledge / P2.amend_activity / P2.show_sth* | |
| 2 | P1: Brauchen wir aber auch nicht!? | *P1.propose_design* | *P1.Proposal* |
| 3 | P2: Nee, das braucht man nicht. | *P2.agree_design* | *P2.Agreement* |

Tabelle 11: Das Paar entwickelte in der Sitzung eine Klasse. Diese Klasse wurde durch Anpassung einer vorhandenen entwickelt. Am Anfang der Sitzung wurde ein Teil, der vermutlich unnötige Funktionalitäten enthielt, auskommentiert. Teile der ursprünglichen Klasse konnten von den Programmierern nicht nachvollzogen werden. Die Situation findet am Ende der Sitzung statt. Das Paar fand einen Codeteil, der im Kommentar eine bisher nicht nachvollzogene Variable erklärt. (FA03 [3:249]; Driver: P1)

### 4.2.4 Episodentyp: Complement

In diesem Episodentyp wird ein Vorschlag gemeinsam erarbeitet. Beide Personen im Paar sind aktiv.

Sie beide beeinflussen die endgültige Lösung des Problems dadurch, dass sie gemeinsam den Problemlösungsweg gestalten. In dieser Episode tragen beide Programmierer

| # | Äußerung | *BKM Code* | *DesignIssue Code* |
|---|---|---|---|
| 1 | P1: OK. Hmm... `if remainingGeometry` Punkt `get` | *P1.propose_design* / *P1.amend_activity* / *P1..write_sth* | *P1.Proposal* |
| 2 | P2: `getNum` ... | *P2.amend_design* | *P2.Amendment* |
| 3 | P1: `getNumGeometry` ist gleich Null. | *P1.amend_design* / *P1.amend_activity* / *P1..write_sth* | *P2.Amendment* |
| 4 | P1: Ja, dann mußt man eigentlich aussteigen und ... | *P1.propose_design* / *P1.amend_activity* / *P1..write_sth* | *P1.Amendment* |
| 5 | P2: ... ein `Message` bringen. | *P2.amend_design* | *P2.Amendment* |
| 6 | P1: Genau. | *P1.agree_design* / *P1.amend_activity* / *P1..write_sth* | *P1.Agreement* |

Tabelle 12: Das Paar bearbeitet eine Methode. Zur Realisierung der nötigen Funktionalität muss ein Algorithmus implementiert werden. (FA04 [2:6]; Driver: P1)

zu der Lösung bei, wobei keine Konfliktanzeichen innerhalb der Zwischenlösungen zu merken sind.

Die Art, wie die Programmierer zu der Lösung beitragen, kann unterschiedlich sein.

So kann es Episoden geben, in denen einer der Programmierer zur Lösung mit zusätzlichen Informationen und der andere mit Vorschlägen beiträgt. Die zusätzlichen Informationen schränken den Problemlösungssuchraum vom Partner ein, so dass er einen Vorschlag erstellen kann. Dieser wird dann vom Paar übernommen. Solch eine Situation ist im Beispiel der Tabelle 13 vorzufinden.

Die Episode wird von P1 initiiert. Die P1 erkennt das Problem und nähert dieses der P2 (Tabelle 13; Äußerungen 1 bis 7). Die Einführung in den Designkontext scheint für die P2 ausreichend zu sein, um mit Hilfe dieser Informationen einen Lösungsvorschlag zu erstellen (Tabelle 13; Äußerung 8). Die P1 scheint bei diesem Vorschlag behilflich zu sein. Sie hat den Problemlösungsprozess der P2 so weit eingeschränkt, dass diese direkt einen Vorschlag stellen kann. Der Vorschlag ist knapp beschrieben, und die Tatsache, dass die P1 diesem zugestimmt hat, läßt darauf schliessen, dass sie der gleichen Meinung bezüglich des Vorschlags ist.

In dieser Beispielepisode wird das Problem gemeinsam gelöst. Die eine Person schränkt den Problemlösungssuchraum vom Partner durch zusätzliche / ergänzende Informationen ein. Ein Hinweis darauf ist die Tatsache, dass die P2 nachfragt, da sie eine der Informationen nicht vollständig nachvollziehen konnte, diese aber als wichtig einschätzt. Die P2 erstellt u.a. dank dieser Informationen einen Lösungsansatz. Der Lösungsansatz

wird eher knapp und fast direkt gestellt.

Es kann jedoch auch *Complement*-Episoden geben, in denen beide Programmierer Lösungsansätze vorschlagen und diese fakultativ um zusätzliche Informationen ergänzen. Innerhalb dieser Vorschläge ist jedoch kein Konflikt zu merken, da ein Vorschlag von beiden Personen gestaltet und erweitert wird.

Ein Beispiel dieser Episodenart ist in der Tabelle 14 zu finden. Die Episode fängt mit einem Vorschlag der P2 an (Tabelle 14; Äußerung 1). Die P2 schlägt vor, eine Methode zu erstellen, in dem sie unsicher ihren Namen benennt. Ihr Vorschlag wird von der P1 ergänzt (Tabelle 14; Äußerungen 2 und 4). Die P1 erweitert den Lösungsansatz um weitere Elemente, die die Schnittstelle dieser Methode direkt beeinflussen. Der Vorschlag wird dann noch ein Mal von der P2 um einen Methodennamen erweitert (Tabelle 14; Äußerung 8). Diesmal ist ihrer Lösungsansatz präziser. Daraufhin erweitert die P1 mit einer zusätzlichen Information den Kontext des Entwurfs (Tabelle 14; Äußerung 9). Sie nennt eine zusätzliche Beziehung, die diese Methode erfüllen muss. Dem folgt ein sicherer Vorschlag der P2 (Tabelle 14; Äußerung 10), in dem sie einen neuen Methodennamen aufschreibt und diesen damit vorschlägt. Diesem Vorschlag stimmt die P1 zu (Tabelle 14; Äußerung 11). Kurz danach fällt ihr auf, dass dieser Lösungsansatz zu einem Fehler geführt hat (Tabelle 14; Äußerung 12). Dieser Fehler wurde dank des automatischen Build-Prozesses der Eclipse-Programmierumgebung entdeckt. Daraufhin ändert die P2 den Namen der Methode noch ein Mal (Tabelle 14; Äußerung 13). Diesem Namen stimmt die P2 zu (Tabelle 14; Äußerung 14).

In dieser Episode findet ein ständiges Ergänzen der Zwischenlösungen zum endgültigen Ergebnis statt. Die Lösung wird von beiden Personen ergänzt. Dies geschieht sowohl direkt als auch indirekt duch das Bereitstellen von Hintergrundinformationen. In den sich ergänzenden Vorschlägen läßt sich kein Konflikt finden. Den Vorschlägen wird oft explizit zugestimmt.

In *Complement*-Episoden wird das Problem gemeinsam gelöst. Beide Personen tragen zu der Lösung bei. Es kann hier aber Episoden geben, in denen eine Person direkt und die andere indirekt durch Erweiterungen des Designkontextes zur Lösung beiträgt. Der Problemlösungsprozess für den endgültigen Lösungsansatz in der Episode ist verteilt, wobei beide Personen in unterschiedlichen Rollen, asymmetrisch, agieren. Eine Person ist damit beschäftigt, den Problemlösungsprozess der anderen durch den den Designkontext erweiternde Informationen zu beeinflussen. Diese führen meist zur Einschränkung des Problemraums, der vom Partner durchsucht wird. Solche Episoden treten vor allem in Situationen auf, in denen sich bei einer der Personen ein größeres Wissen bezüglich des Kontexts des Designs vermuten läßt.

Andersherum kann es auch Episoden geben, in denen beide Programmierer zu der Lösung direkt durch Vorschläge oder Erweiterungen beitragen. So findet bei beiden der gesamte Problemlösungsprozess statt, der mit einem (Zwischen-)Lösungsansatz abgeschlossen wird. Die Problemlösungsprozesse für die einzelnen Zwischenlösungen beeinflussen sich gegenseitig auf dem Weg zu der endgültigen und vollständigen Lösung. Solche Episoden treten häufiger in Sitzungen der Paarkombination Experte-Experte auf. Die betrachteten Probleme sind zumeist komplexer, was sich u.a. durch die Erweiterun-

gen des Lösungansatzes ausdrückt.

Die Leistung der *Complement*-Episoden wird von beiden Personen positiv beeinflusst. Die auftretende Koordination führt meist näher zum endgültigen Lösungsansatz. Zusätzlich können Motivationsgewinne auftreten. Diese könen u.a. dank der beidseitigen Teilnahme an der Leistung und am Erfolg (Lösung eines Designproblems) des Paars auftreten.

Das Voneinanderlernen ist vor allem in den *Complement*-Episoden vorhanden, in denen eine asymmetrische Aufteilung der Rollen auftritt. Hier hat die vorschlagende Person die Möglichkeit, einige Informationen bezüglich des Designkontexts zu erfahren und u.a. mit deren Hilfe einen Lösungsansatz zu erstellen. Der Lernaspekt der *Complement*-Episoden mit symmetrischer Rollenaufteilung ist begrenzt, da diese Episoden vor allem in Situationen auftreten, in denen die Kenntnisse des Designkontextes bei beiden Personen vergleichbar sind.

Die Qualität der erreichten Lösung ist vermutlich besser (verglichen mit Einzellösungen). Der Grund dafür ist die Art der Verteilung des Problemlösens auf zwei Personen. Jede Person trägt zur Lösung das bei, dessen Korrektheit sie annimmt. So wird die Menge der Kontextinformationen, anhand derer der Entwurf entsteht, aus den Mengen der Kontextinformationen jeder Person summiert, was ein Indiz für einen besseren Entwurf ist. Dazu werden die Schlussfolgerungen, die u.a. dank der Kontextinformationen zustande kommen und sich u.a. in den vorgeschlagenen Lösungsansätzen äußern, von einem wahrscheinlich motivierten, da einbezogenem, Partner kontinuierlich beurteilt. Sollten die Wahrscheinlichkeiten der Defektentdeckung und des Erstellens eines korrekten Entwurfs bei beiden Personen höher als die Hälfte sein, so ist in dieser Episode die Wahrscheinlichkeit für einen besseren Entwurf höher (verglichen mit Einzellösungen). Da sich in den *Complement*-Episoden kein Widerspuch findet, kann man es als einen Hinweis für ein Levek an Qualität ansehen, welches von beiden Programmierern als ausreichend eingeschätzt wird.

### 4.2.5 Episodentyp: Conflict

Im Episodentyp *Conflict* versuchen beide Personen, den Entwurf durch ihre Vorschläge direkt zu gestalten. Die von beiden Programmierern gestellten Vorschläge sind nicht deckungsgleich. Wenn man davon ausgeht, dass beide Personen erwarten, dass ihr Ansatz korrekt ist und diesem daher zugestimmt wird, entsteht eine Konfliktsituation. So werden in jeder Konflikt-Episode von beiden Entwicklern Lösungsansätze vorgestellt. Diese Ansätze lassen sich nicht zur selben Zeit anwenden und sind somit in ihrer ursprünglichen Form unvereinbar.

In diesem Episodentyp können beide Personen als aktiv bezeichnet werden. Bei dieser Abstraktion kann man von einer symmetrischen Paarbeziehung sprechen.

Eine Beispielepisode gibt die Tabelle 15 wieder. Dieses Beipiel bildet eine stark reduzierte und einfache Konfliktsituation. Zunächst schlägt P1 eine Lösung vor (Tabelle 15; Äußerung 1). Diesem Vorschlag folgt ein Gegenvorschlag von P2 (Tabelle 15; Äußerung 2). Dem stimmt P1 gleich zu (Tabelle 15; Äußerung 3). In diesem Beispiel sind beide

| # | Äußerung | *BKM Code* | *DesignIssue Code* |
|---|---|---|---|
| 1 | P1: Das ist auch wieder die Änderung von `VirtualColumn` auf `VirtualAttributes`. | *P1.explain_knowledge* | *P1.ITDC* |
| 2 | P2: Hmmm... Moment. | *P2.explain_standard of knowledge* | |
| 3 | P1: Das ist eine andere Implementierung, die diese `Abstracts` nicht benutzt. | *P1.explain_knowledge* | |
| 4 | P2: 'Ne andere Implementierung von der `FeatureAttributConfiguration`, .... | *P2.explain_knowledge / P2.explain_standard of knowledge* | |
| 5 | P1: Ja, ja... | *P1.agree_knowledge* | |
| 6 | P2: ... die was nicht benutzt? | *P2.ask_knowledge* | |
| 7 | P1: Die abstrakte Klasse, die wir eben angepasst haben, nicht benutzt. | *P1.explain_knowledge* | |
| 8 | P2: Eh... Das müssen wir hier noch anpassen. | *P2.propose_design* | *P2.Propose* |

Tabelle 13: Das Paar hat in dieser Sitzung u.a. Anpassungen der Klasse `VirtualColumn` an `VirtualAttributes` vorgenommen. Dies geschah mit Hilfe des Eclipse-Refactoring-Werkzeugs. Nicht alles konnte automatisch angepasst werden; im Buildbericht sind noch Fehler zu finden. In dieser Episode passt das Paar diese an. (FA02 [3:276-3:287]; Driver: P1)

| # | Äußerung | *BKM Code* | *DesignIssue Code* |
|---|---|---|---|
| 1 | P2:Irgendwie würde ich 's mir wünschen, man könnte hier irgendwie so 'was machen, wie, keine Ahnung, `start`, hmm... [*nuschelt kurz*] `editMode` with keine Ahnung... | *P2.propose_design* | *P2.Proposal* |
| 2 | P1: Mit den Sachen, die wir halt brauchen, z.B dem `EditGeometryType`... | *P1.explain_knowledge* / *P1.amend_design* | *P1.ITDC* / *P1.Amendment* |
| 3 | P2: Eben! und... | *P2.agree_knowledge* / *P2.agree_design* / *P2.amend_activity* / *P2.write_sth* | *P2.Agreement* |
| 4 | P1: ... der `EditDialogConfiguration` | *P1.explain_knowledge* / *P1.amend_design* | *P1.ITDC* / *P1.Amendment* |
| 5 | | *P2.write_sth* | *P2.Agreement* |
| 6 | P2: So 'was... | | |
| 7 | P1: ... und dem `snapTheme`. Ah so !? Ok, den nicht. | *P1.explain_knowledge* | *P1.ITDC* |
| 8 | P2: So 'nen, so 'nen, so 'nen... Oder...`editStarter.start` oder so... | *P2.amend_design* / *P2.amend_activity* / *P2.write_sth* | *P2.Amendment* |
| 9 | P1: Das ist dann eigentlich auch so 'n Algorithmus, wie die ganzen `Performer`, oder? | *P1.explain_knowledge* | *P1.ITDC* |
| 10 | | *P2.write_sth* | *P2.Amendment* |
| 11 | P1: Uhum... | *P1.agree_design* | *P1.Agreement* |
| 12 | P1: Yh... Schlüsselwort! | *P1.explain_finding* | *P1.ITDC* |
| 13 | | *P2.write_sth* | *P2.Amendment* |
| 14 | P1: `perform`. Genau. | *P1.agree_design* | *P1.Agreement* |

Tabelle 14: Das Paar implementiert eine Methode. P2 schlägt vor, im Algorithmus der bearbeiteten Methode einen Methodenaufruf unterzubringen. Die aufzurufene Methode exisitiert noch nicht. (FA04 [1:33]; Driver: P2)

Vorschläge präzise formuliert. So ist es relativ eindeutig, dass die beiden Lösungsansätze sich nicht direkt vereinbaren lassen. Zusätzlich widerspricht die P2 explizit dem Lösungsansatz von P1 (Tabelle 15; Äußerung 2), was darauf hindeutet, dass P2 ihren Lösungsansatz und den von P1 zumindest teilweise als widersprüchlich wahrnimmt. Daher kann man in dieser Situation von einem Konflikt sprechen, der jedoch schnell ausgetragen wird. Die P1 ändert ihre Einstellung gegenüber der Lösung für das Problem. Sie bevorzugt die Lösung von P2, was man an der expliziten Zustimmung von P1 erkennen kann (Tabelle 15; Äußerung 3).

Die Vorschläge müssen nicht explizit gestellt werden. Die Rolle eines Vorschlages kann auch eine Begründung für diesen übernehmen (siehe auch: Konzept *Proposal*). Dies ist in der Tabelle 16 zu sehen. Die Situation beginnt mit einem Lösungsvorschlag von P2 (Tabelle 16; Äußerung 1). Dieser ist einigermassen gut beschrieben, weshalb man davon ausgehen kann, dass P1 diesen Lösungsvorschlag nachvollziehen kann. Daraufhin erklärt P1 der P2 ihre bisherige Vorgehensweise bezüglich des Objektes der Episode (Tabelle 16; Äußerung 2). In der nächsten Äußerung verweist P1 auf eine möglicherweise vorab getroffene Anforderung (Tabelle 16; Äußerung 4), vermutlich mit dem Hintergrund, die P2 an einen anderen, bereits existierenden Lösungsansatz für dieses Problem zu erinnern. Der im Vorschlag erwähnte Begriff „Fassade“ läßt vermuten, dass dieser Lösungsansatz auf einer höheren Abstraktionsstufe als dieser von P2 einzuordnen ist. Obwohl der von P1 genannte Ansatz vermutlich nicht detailliert ist, kann man jedoch aus dem weiteren Verlauf der Episode (u.a. Tabelle 16; Äußerung 5) darauf schliessen, dass beide Lösungsansätze nicht übereinstimmen. So ist auch in diesem Beispiel eine Konfliktsituation zu bemerken. Beide Programmierer dieses Beispiels stellen ihre Lösungsansätze (Tabelle 16; Äußerungen 1, 4, 7 und 9) in einer Interaktion vor. Es läßt sich aus dem Verlauf nicht ersehen, dass es bei einem der Programmierer zu einer Einstellungsänderung kommt. Das Objekt der Episode wird ohne explizite Einigung verlassen, und damit endet auch die Episode.

Die Auswirkung einer Konfliktepisode für den Verlauf des Entwurfsprozesses in einer Paarprogrammiersitzung hängt von mehreren Faktoren ab.

In einer Episode kann es zu einer expliziten Einigung mit beidseitiger Zustimmung kommen (Tabelle 15; Äußerung 3), dies ist jedoch oft nicht der Fall, so dass es zu keiner Einigung kommt (siehe Tabelle 16). Darüber hinaus wird in einigen Konfliktsituationen eine Einigung gefunden, die jedoch für einen der Programmierer nicht zufriedenstellend oder nachvollziehbar ist.

Solch ein Beispiel spiegelt die Tabelle 17 wieder. Diese Situation kann als eine Konfliktsituation betrachtet werden, da auf den Vorschlag von P1 (Tabelle 17; Äußerung 1) die P2 mit einem Widerspruch (siehe Konzept *Disagreement*) antwortet. Im weiteren Verlauf dieser Situation stimmt die P2 dem Vorschlag von P1 nach einer kurzen Erklärung des Kontextes des Vorschlags (Tabelle 17; Äußerung 5) zu, und erwähnt zur gleichen Zeit, dass sie den Lösungsansatz nicht nachvollziehen kann (Tabelle 17; Äußerung 6).

Die Situation der Tabelle 18 ist ebenso ein Beispiel für eine Konfliktsituation. Hier kommt es zu einer Einigung, mit welcher einer der Programmierer nicht zufrieden ist. Nach einer Einführung in den Kontext (Tabelle 18; Äußerungen 1 und 2) schlägt die P1

eine Lösung vor (Tabelle 18; Äußerung 3). Diese wird von P2 abgelehnt und dafür eine andere vorgeschlagen (Tabelle 18; Äußerungen 4 und 6), die im Widerspruch mit der Lösung von der P1 steht („drin lassen“ versus „ausmachen“). Diese Äußerung der P2 (Tabelle 18; Äußerungen 4 und 6) wird von P1 unterbrochen (Tabelle 18; Äußerung 5). P1 äussert in dieser Unterbrechung höchstwahrscheinlich ihre Enttäuschung über den abgelehnten Vorschlag, worauf die fallende Intonation dieses Ausrufes hindeutet .

Es wird angenommen, dass eine Einigung spätestens in dem Moment stattfindet, in dem der Vorschlag umgesetzt wird und bis Ende der Episode kein Widerspruch bzgl. dieses Vorschlags erhoben wird.

Es kann jedoch auch früher durch eine explizite Zustimmung zu einer expliziten Einigung kommen. So eine Einigung ist ein Hinweis dafür, dass die Lösungsqualität von beiden Programmierern akzeptiert wurde und könnte somit auf eine höhere Qualität deuten, wenn man es mit den Konfliktepisoden ohne eine explizite Zustimmung vergleicht. Darüber hinaus kann eine explizite Zustimmung ebenso ein Indiz für einen vergleichbaren Wissenstand über das Objekt der Episode und den Designkontext sein. Zu einer expliziten und überzeugten Zustimmung kommt es vor allem in Situationen, in denen sowohl der Lösungsansatz als auch der Problemlösungsweg zumindest teilweise von dem Zustimmenden nachvollzogen wurde. Sollte es zu einer Einigung kommen, die trotz eines angedeuteten Widerspruchs oder trotz eines angedeuteten Verständnismangels für den Lösungsansatz beziehungsweise Lösungsweg stattfindet, führt dies zu einer Einschränkung der positiven Auswirkungen der Paarprogrammierung. Der Verständismangel weist darauf hin, dass es nicht zu einem ausreichenden Lernprozess kam, der es ermöglicht hätte, den Lösungsansatz nachvollziehen und beurteilen zu können. Ein merkbarer Widerspruch, trotz erreichter Einigung, kann ein Anhaltspunkt dafür sein, dass der Lösungsansatz potenziell qualitative Risikostellen aufweist.

Ein weiterer Faktor, der die Auswirkungen einer Designepisode beeinflussen kann, ist die Form der Interaktion der Programmierer. Die von beiden Programmierern gestellten Vorschläge können verschiedene Formen annehmen. Von Interesse sind hier die Antworten auf folgende Fragen:

- Werden die Vorschläge durch Argumente (Begründungen, zusätzliche Informationen) gestärkt?
- Beziehen sich widersprechende Äußerungen fachlich darauf, was sie widersprechen? Wird der Widerspruchsgrund genannt und erklärt?

Die folgenden Beispiele zeigen die unterschiedlichen Möglichkeiten der Ausprägung der Untermauerung der Lösungsvorschläge mit Hilfe von Argumenten. Diese Ausprägung kann über ein vollständiges Weglassen zusätzlicher Informationen bis hin zu einer einseitigen oder beidseitigen Begründung des Vorschlages reichen.

Die Tabelle 19 stellt ein Beispiel dar, in dem die Vorschläge von keinem der Programmierer durch Argumente gestärkt werden. Die P1 schlägt eine Lösung vor (Tabelle 19; Äußerung 1), ohne in dieser Situation zusätzliche Informationen, die ihren Lösungsansatz näher bringen könnten, zu bieten. Daraufhin antwortet P2 mit einem widersprüchlichen

Lösungsansatz (Tabelle 19; Äußerung 2). Die P2 nennt ebenso keine Gründe ihre Entscheidung.

Im Gegensatz dazu stellt die Tabelle 16 eine Situation dar, in der beide Programmierer ihre Lösungsansätze mit Argumenten stärken.

Der erste hier angegebene Vorschlag wird von P2 nicht weiter beschrieben (Tabelle 16; Äußerung 1). Die P1 nennt jedoch ihren ersten Vorschlag in der Form einer Einführung in den Designkontext (Tabelle 16; Äußerung 4). Sie teilt der P2 die Gründe für ihren vorhandenen Lösungsansatz mit, ohne diesen explizit zu nennen. In ihren weiteren Äußerungen (Tabelle 16; Äußerungen 7 und 9) schlägt die P2 eine Lösung vor. Diese wird durch die Angabe einer Begründung gestärkt (Tabelle 16; Äußerung 9).

In einem Teil der Konflikt-Episoden ist ein asymmetisches Verhältnis bei der Angabe der Argumente für die Vorschläge zu finden. In diesen Situationen gibt eine der Personen Argumente für ihren Lösungsvorschlag an, während die andere Lösungen vorschlägt, ohne fachliche Argumente dafür zu nennen.

Ein Beispiel für diese Situation kann man in der Tabelle 17 finden. Hier schlägt die P1 einen Lösungsansatz vor (Tabelle 17; Äußerung 1), dem die P2 anschliessend widerspricht (Tabelle 17; Äußerung 2). In ihren weiteren Äußerungen versucht die P1 der P2 ihren Ansatz zu begründen (Tabelle 17; Äußerungen 3 und 5). Die P2 gibt für ihren Widerspruch keine Argumente an.

Die Angabe der Argumente für den eigenen Lösungsansatz kann sich stark auf den Verlauf der Konfliktepisoden auswirken.

Sollten die Programmierer in einer Konfliktepisode keine Argumente für ihre Lösungsansätze angeben, so müssen sie beide über einen ähnlichen Wissenstand bezüglich des Objektes der Episode und den dazugehörigen Designkontext verfügen, denn sonst kann der Lösungsansatz vom Partner kaum nachvollzogen werden. So können auch die Lerneffekte einer solchen Episode eingeschränkt sein. Wird der Lösungsansatz nicht nachvollzogen, so ist auch eine kritische Beurteilung des Ansatzes kaum möglich, und damit bleibt die potenzielle Qualitätsverbesserung aus. Bei Programmierern mit vergleichbaren Kenntnissen oder Fähigkeiten kann jedoch das Weglassen zusätzlicher Argumente die Leistung des Paares erhöhen, da der Koordinationsaufwand geringer ist.

Ein weiterer Faktor, der den Verlauf oder die Auswirkungen einer Konfliktepisode beeinflussen kann, ist die Bezugnahme der eigenen Äußerungen auf die Äußerungen des Partners. Hier ist es von Interesse, ob sich die beiden Konfliktparteien in ihren Äußerungen fachlich auf die Äußerungen des Partners beziehen bzw. auf die Argumente des Partners eingehen. Dies ist der Fall, wenn in einer Konfliktsituation ein Vorschlag (siehe Konzept *Proposal*) von einer Äußerung (meistens *ITDC*) des Partners gefolgt wird, die sich auf den Vorschlag bezieht. Diese Äusserung hat die Aufgabe, zusätzliche Informationen über den Kontext des Designs oder den Lösungsansatz selbst mitzuteilen. Diese Informationen sollen dem Partner entweder die Risiken seines Vorschlags signalisieren oder die Vorteile des eigenen Lösungsvorschlags andeuten.

In der Tabelle 20 ist eine Beispielepisode zu finden, in der sich beide Programmierer auf die Äußerungen des Partners beziehen. In dieser Episode macht zuerst die P2 implizit einen Vorschlag (Tabelle 20; Äußerung 2). In der nächsten Äußerungen von P1

ist ein Bezug auf diesen Vorschlag zu merken (Tabelle 20; Äußerungen 3 und 5). Vor allem in der Äußerung 5 geht die P1 auf den Vorschlag von P2 ein und benennt den Nachteil, dass es sich nicht um den kürzesten Weg handeln würde. Danach fängt die P1 an, ihren Lösungsansatz vorzustellen (Tabelle 20; Äußerungen 5, 7 und 8). Nach der ersten Vorstellung des Ansatzes, äussert sich die P2. Sie bezieht sich auf den Vorschlag von P1 und nennt die Problemstelle (Tabelle 20; Äußerung 6), dass eine von P1 an dieser Stelle vermutete Schnittstelle hier nicht zu finden ist.

In der Tabelle 16 ist eine Beispielepisode zu finden, in der sich einer der Programmierer auf die Äußerungen des Partners bezieht. Diese Episode fängt mit einem Lösungsvorschlag der P2 an (Tabelle 16; Äußerung 1). In den folgenden Äußerungen der P1 (Tabelle 16; Äußerungen 2 und 4) schlägt die P1 implizit einen weiteren Lösungsansatz vor. Daraufhin antwortet die P2 mit Äußerungen, die sich auf den Vorschlag der P1 beziehen. Erst weist die P2 auf eine Risikostelle des Vorschlags der P1 hin (Tabelle 16; Äußerung 5), und darüber hinaus begründet die P2 ihren Lösungsansatz u. a. mit der Angabe dieser vermuteten Risikostelle des Lösungsansatzes der P1 (Tabelle 16; Äußerung 5).

Im Gegensatz dazu stellt die Tabelle 15 eine Beispielepisode dar, in der sich die Programmierer nicht auf die Äußerungen des Partners beziehen. Die Vorschläge gehen hier nicht fachlich auf die Äußerungen des Partners ein.

Der gegenseitige fachliche Bezug auf die Äußerungen des Partners ist ein Hinweis für eine potenzielle Qualitätsverbesserung. Das Aufzeigen der Schwächen im Lösungsansatz des Partner ist ein Hinweis darauf, dass die subjektive Einschätzung der Qualität des vorgeschlagenen Entwurfes nicht ausreichend ist. So kann eine bessere Qualität oder diese schneller erzielt werden. Der gegenseitige Bezug kann auch den Effekt haben, den Lernprozess zu gestalten. Ein genaues Verweisen auf die Schwächen der eigenen Lösung erlaubt es dem Vorschlagenden, den eigenen Lösungsweg noch mal, mit zusätzlichen Informationen, zu durchdenken. Der gegenseitige Bezug kann als ein Indiz für eine stattgefundene kritische Beurteilung angesehen werden.

Im Verlauf einer Konfliktepisode wenden die Programmierer je nach Situation unterschiedliche Konfliktstile an. Diese können selbst innerhalb einer Episode verschieden sein. Von Interesse ist der letzte vom Programmierer angewandte Konfliktstil. Dieser kann schlussendlich zu einer impliziten oder expliziten Einigung führen. Aus dem Datenmaterial liessen sich folgende Konfliktstile ermitteln:

- Aufschiebung - die Problemlösung wird verschoben
- Verweis an Externe - der Konflikt wird dank einer externen Instanz gelöst
- Folgschaft - der Konflikt wird durch die Übernahme eines von einem der Programmierer stammenden Ansatzes gelöst
- Lösungsverschmelzung - der Konflikt wird durch die Übernahme eines Vorschlages, der durch die Zusammenführung der Lösungswege beider Programmierer entstand, gelöst

Die Tabelle 21 enthält eine Beispielsituation, in der der Konflikt mit Hilfe des Konfliktstils „Aufschiebung“ gelöst wird. Die Konfliktsituation dieser Episode entsteht bei den Äußerungen 2 bis 5. Auf den Vorschlag der P2 (Tabelle 21; Äußerung 2) antwortet die P1 mit einer Äußerung, welche ihr fehlendes Verständnis für den Vorschlag der P2 ausdrückt (Tabelle 21; Äußerung 3). Zusätzlich könnte man in diese Äußerung einen Widerspruch interpretieren. Nach einem Klärungsversuch des Vorschlags seitens der P2 (Tabelle 21; Äußerungen 4 bis 6) wendet die P1 den Konfliktstil „Aufschiebung“ an. Das Problem wird nicht gelöst, die Zustimmung für den Lösungsansatz bleibt aus. Das Problem wird in die Zukunft verschoben, worauf Äußerungen beider Programmierer hindeuten (Tabelle 21; Äußerungen 7 und 8). Das erfolgreiche Anwenden des „Aufschiebung“-Konfliktstils führt nicht zur Lösung eines Problems. Selbst wenn Lösungsansätze vorgestellt werden, werden diese nicht übernommen. Somit ist die Leistung des Paares, wenn es um die Umsetzung des Entwurfs geht, nicht vorhanden. Dieser Konfliktstil führt zu einer einfachen Verschiebung des Problems, was bei Konflikten mit einer geringen Chance auf eine Lösung vorteilhaft sein kann. So müssen weniger Ressourcen in die Lösung eines solchen Konflikts investiert werden.

Ein Beispiel eines Konflikts, der mit Hilfe des Konfliktstils „Verweis an Externe“ gelöst wird, ist in der Tabelle 22 zu finden. In dieser Episode entsteht ein Konflikt, da dem Lösungsvorschlag der P2 (Tabelle 22; Äußerung 3) die P1 mit einer Äußerung antwortet, deren Ziel es vermutlich ist, sich für die bestehende Lösung einzusetzen (Tabelle 22; Äußerung 4). Die P1 bezieht sich in ihrer Begründung auf externe Akteure. Die externen Akteure (hier „irgendwelche Leute“) erstellten in der Vergangenheit den Entwurf dieses Codeteils und deren Entscheidung wird angenommen (Tabelle 22; Äußerung 6).

Die Lösung wird bei Anwendung dieses Konfliktstils schnell erzielt. Das Vertrauen, das man der externen Instanz schenkt, wirkt sich im Vertrauen auf die Lösung aus. Die externe Instanz ist am Konflikt nicht direkt beteiligt, daher ist das Risiko der Befangenheit geringer. So kann ein Konflikt, und damit auch das Problem, dank einer externen Instanz schnell gelöst werden. Die Qualität der Lösung hängt jedoch vom externen Akteur ab und wird in der Episode nicht verbessert. Das Lernprozess ist ebenso begrenzt, da potenzielle Wissensmängel bezüglich der Lösung nicht beseitigt werden können.

Eine Beispielepisode für den Konfliktstil „Folgschaft“ ist in der Tabelle 18 zu finden. Diese Episode beginnt mit einer Einführung in den Kontext des Designs (Tabelle 18; Äußerungen 1 und 2). Danach schlägt die P1 einen Lösungsansatz vor (Tabelle 18; Äußerung 3). Daraufhin entsteht in dieser Episode eine Konfliktsituation - die P2 lehnt den Lösungsansatz der P1 ab und schlägt eine eigene Lösung vor (Tabelle 18; Äußerung 4). Eine explizite Zustimmung für den Vorschlag der P2 bleibt aus, die Lösung wird jedoch anschliessend umgesetzt (implizite Zustimmung). So hat sich die P1 ihren Lösungsansatz verlassen und das Paar folgte dem Lösungsansatz der P2. Der Lösungsansatz der P2 wurde von der P1 unverändert übernommen.

Die Konflikte, die durch Folgschaft gelöst werden, können sich durch eine hohe Leistung auszeichnen, da der Koordinationaufwand im Paar gering gehalten werden kann. Solche Konflikte können aber potenziell in motivationalen Risiken resultieren. So kann

es zu einem Motivationsverlust kommen, sollte die eigene Leistung dauerhaft nicht die Lösung beeinflussen. Bei Anfänger-Experte-Paaren kommt es häufig zu derartigen Austragungen, wobei der Lösungsansatz des Anfängers dem des Experten meistens untergeordnet wird. Das schützt das Paar vor potenziell schlechteren Entwürfen, die aus den Lösungsansätzen des Anfängers folgen könnten.

Eine Problemlösung, die durch das Anwenden des Konfliktstils „Lösungsverschmelzung" erzielt wurde, ist in der Tabelle 23 zu sehen. Hier entwickelt sich ein Konflikt nach dem Vorschlag eines Lösungsansatzes der P1 (Tabelle 23; Äußerung 6). Dem antwortet die P2 mit einem Argument, das gegen die Lösung der P1 spricht (Tabelle 23; Äußerung 7). Hier musste die P2 die Lösung der P1 nachvollziehen können und mit dieser Information den Problemlösungsweg wiederholen. Diese führte sie zu der Erkenntnis, dass der Lösungsansatz der P2 eine Schwachstelle hat, die sie in ihrer Äußerung (Tabelle 23; Äußerung 7) der P1 mitteilt. Die P1 nimmt die Information an und schlägt daraufhin einen neuen Lösungsansatz vor (Tabelle 23; Äußerung 8), welcher aufgund der Zusammenführung der Problemlösungwege beider Personen entstanden ist. Dieser Lösungsansatz wird noch ein Mal von der P2 präzisiert (Tabelle 23; Äußerung 9), und die Episode endet mit einer expliziten Zustimmung beider Programmierer (Tabelle 23; Äußerungen 9 und 10).

Die Austragung eines Konfliktes durch „Lösungsverschmelzung" kann besonders aufwendig sein, da der Koordinationsaufwand hoch ist. Die Partner müssen die Lösungsvorschläge untereinander abgleichen und den Problemlösungsprozess potenziell mehrmals wiederholen. Die vermutete Qualität der Lösung sollte jedoch höher sein, da mehrere Lösungsansätze zu einem werden, der von beiden Programmierern als ausreichend hochqualitativ eingestuft wird.

| # | Äußerung | *BKM Code* | *DesignIssue Code* |
|---|---|---|---|
| 1 | P1: Aber dann umbenennen wir `checkFerryPolygon` in `findFerry` ... `findTeleporter`! | *P1.propose_design* | *P1.Proposal* |
| 2 | P2: Nee, mach noch 'ne zusätzliche Funktion - die obere `checkFerryPolygon` macht `findTeleporter` und ruft dann die untere auf. | *P2.challenge_design* | *P2.Challenge* |
| 3 | P1: Ok. | *P1.agree_design* | *P1.Agreement* |

Tabelle 15: Die Entwickler haben gerade eine Schnittstelle hinzugefügt und möchten diese jetzt bearbeiten. P1 möchte diese umbenennen, P2 schlägt aber vor, eine zusätzliche Funktion einzubauen. Die genaue Bedeutung/Funktionalität der erwähnten Namen ist nicht nachvollziehbar. (FA02 [1:481-1:451]; Driver: P1)

| # | Äußerung | *BKM Code* | *DesignIssue Code* |
|---|---|---|---|
| 1 | P2: Also, ich hätte einfach in der `FeatureAttributConfiguration` z. B. ein Set von `VirtualAttributes` oder so was gemacht, und das andere dann später mißbraucht. | *P2.propose_design / P2.propose_strategy* | *P2.Proposal* |
| 2 | P1: Ich bin da am rumspielen gewesen, in welche Richtung kann man was machen. Ich wollte überhaupt was hinterlegen. | *P1.explain_knowledge* | *P1.ITDC* |
| 3 | P2: Ok... | *P2.explain_standard of knowledge* | |
| 4 | P1: Und die andere Sache, ich hatte es so verstanden, dass wir halt versuchen wollen, das über diese Fassade nach außen für alle anderen Geschichten unsichtbar zu machen. | *P1.explain_knowledge / P1.remember_req* | *P1.Challenge / P1.ITDC* |
| 5 | P2: Wie die Fassade aussieht, weiß ich aber noch nicht. Das ist mir noch nicht klar. | *P2.explain_standard of knowledge* | *P2.Challenge / P2.ITDC* |
| 6 | P1: Ja... Das, das.... | *P1.mumble_sth* | |
| 7 | P2: Außerdem wir brauchen erstens mal die Dinger, so wie die hier sind, und zweitens brauchen wir 'ne Abstraktion. Wir brauchen eigentlich beides, meiner Meinung nach. | *P2.propose_design* | *P2.Challenge* |
| 8 | P1: Die Zeit zum Entwickeln werden wir auch haben. | *P1.explain_knowledge* | |
| 9 | P2: Insofern, bevor ich nicht weiß, was mir da brauchen, um unsere neue zu schaffen... Abstraktion... würde ich gern nur das, was da ist, reintun in die Datenstruktur, weil wir das auf jeden Fall brauchen, allein schon für die Persistenz. | *P2.propose_startegy / P2.propose_design / P2.explain_knowledge* | *P2.Challenge / P2.ITDC* |

Tabelle 16: In dieser Sitzung werden zwei Codefragmente aneinander angepasst. Die beiden Fragmente wurden zuvor von den beiden Entwicklern in anderen Sitzungen separat entwickelt. Die bearbeiteten Fragmente beziehen sich hauptsächlich auf die Anpassung der Schnittstelle zwischen der Steuerungs- und Präsentationsschicht (nach MVC). Die genaue Bedeutung/Funktionalität der erwähnten Namen von Artefakten (z.B. Methodennamen) ist nicht nachvollziehbar. (FA02 [3:46-3:57]; Driver: P1)

| # | Äußerung | *BKM Code* | *DesignIssue Code* |
|---|---|---|---|
| 1 | P1: So wie das momentan ... Das kann man hier vernachlässigen | *P1.propose_design* | *P1.Proposal* |
| 2 | P2: Das war eben nicht richtig. | *P2.disagree_design* | *P2.Disagreement* |
| 3 | P1: Das war nach deiner Änderung nicht richtig. Davor war es richtig. | *P1.explain_knowledge* | *P1.ITDC* |
| 4 | P2: [*naharpe*]? Eigentlich nicht. | *P2.mumble_sth* | |
| 5 | P1: Das hat da ... Also die ... Die Strategie vorhin war die, dass man über `allColumnAttributes` auch die Virtuellen mitkriegt. Deshalb musste ich die zurücksetzen und dann, wenn sich nichts ändert, dann wieder setzen . | *P1.explain_knowledge* | *P1.ITDC / P1.Proposal* |
| 6 | P2: Ok ... verstehe jetzt gar nichts, aber Ok. | *P2.explain_standard of knowledge P2.agree_knowledge / P2.agree_design* | *P2.Agreement* |
| 7 | P1: Das ist ... | *P1.mumble_sth* | |
| 8 | P2: Ich habe es nicht verstanden, aber gut. | *P2.explain_standard of knowledge* | *P2.Agreement* |
| 9 | P1: Also, das ist ... Es ist ... | *P1.mumble_sth* | |

Tabelle 17: Das Paar hat gerade ein Refactoring mit Hilfe eines Refactoring-Werkzeugs durchgeführt. Das Werkzeug konnte aber nicht alle Codestellen anpassen. So bearbeitet das Paar die Fehlermeldungen aus dem Build-Bericht. (FA02 [1:417]; Driver: P1)

| # | Äußerung | *BKM Code* | *DesignIssue Code* |
|---|---|---|---|
| 1 | P1: `getRGBColour` war fürs `VisiNetLayer` ? Oder für was? | *P1.ask_knowledge* | *P2.ITDC* |
| 2 | P2: Ja das war glaube ich eigentlich eh' so ne Art.... Ah nee, nee, nee ... das ist schon eine Funktion, die man leider von aussen aufrufen muss. Die ist für die, die ... ehmm... den `ResultCode` umzuwandeln in 'ne Farbe. | *P2.explain_knowledge* | |
| 3 | P1: Ok, dann lass ma es drin. | *P1.propose_design* | *P1.Proposal* |
| 4 | P2: Nee, ich würde es erstmal ausmachen, ... | *P2.challenge_design* | *P2.Challenge* |
| 5 | P1: Oh, Mann! | *P1.mumble_sth* | |
| 6 | P2: ... weil wir wollen es vielleicht anders machen. | *P2.explain_knowledge* | *P2.ITDC* |

Tabelle 18: Das Paar editiert die Header-Datei der bearbeiteten Klasse. Diese Klasse wurde durch das Kopieren einer vorhandenen erstellt. So wird nach und nach jede vorhandene Methode angesprochen, und das Paar entscheidet über das weitere Vorgehen bezüglich dieses Codestückes. (FA03 [1:595]; Driver: P1)

| # | Äußerung | *BKM Code* | *DesignIssue Code* |
|---|---|---|---|
| 1 | P1: `init` brauch' ma? Nee... | *P1.propose_design* | *P1.Proposal* |
| 2 | P2: Ja. | *P2.challenge_design* | *P2.Challenge* |

Tabelle 19: Das Paar editiert die Header-Datei der bearbeiteten Klasse. Diese Klasse wurde durch das Kopieren einer vorhandenen erstellt. So wird nach und nach jede vorhandene Methode angesprochen, und das Paar entscheidet über das weitere Vorgehen bezüglich dieses Codestückes. (FA03 [1:591]; Driver: P1)

| # | Äußerung | *BKM Code* | *DesignIssue Code* |
|---|---|---|---|
| 1 | P1: Was machst du? | *P1.ask_step* / *P1.ask_knowledge* | |
| 2 | P2: `SelectionModel` | *P2.explain_knowledge* / *P2.propose_design* | *P2.ITDC* / *P2.Proposal* |
| 3 | P1: Nee, nee! Warte, ich glaub', es ist ... Das brauch ma' nicht mal. | *P1.challenge_design* | *P1.Challenge* |
| 4 | P2: Wieso? Du mußt auch [undeutlich] | *P2.ask_knowledge* *P2.explain_knowledge* | *P2.ITDC* / *P2.Challenge* |
| 5 | P1: Also, das wäre der Standardweg, aber ich glaube, es gibt eine Abkürzung hier. Weil wir ja das sowieso an anderen Stellen schon ... Also ich würd' vermuten, das es 'ne Methode gibt, um das selektierte `FeatureProxy` abzuholen. | *P1.explain_knowledge* | *P1.ITDC* / *P1.Challenge* |
| 6 | P2: Es gibt `FeatureLayerSelection`. Das ist aber alles. Es ist ja mehr. | *P2.explain_knowledge* | *P2.ITDC* / *P2.Challenge* |
| 7 | P1: Es gibt die `FeatureLayerSelection`. Wobei... Was ist ... `FeatureLayer`, `ThemeLayer`, `ID` ... Nein, es ist was anderes. Kann natürlich schon sein, dass wir ... Also, ich würde mal nachgucken, wie das in dem Fall vorgegangen wird, wo wir zum Beispiel Objekte bearbeiten. Also wir haben schon wieder eigentlich den Fall, so ähnlich wie: „du selektierst ein Objekt und dann bearbeitest du seine Geometrie". So ähnlich haben wir es jetzt. Wir haben ein selektiertes Objekt. | *P1.explain_knowledge* | *P1.ITDC* |
| 8 | P1: Dieses selektierte Objekt, das Vorgehen würd' ich gern gleich machen, wie bei anderen. | *P1.propose_design* | *P1.Proposal* |

Tabelle 20: Das Paar fügt eine neue Funktionalität hinzu. Die generelle Vorgehensweise (Strategie) wurde bereits vom Paar festgelegt. (FA04 [1:46]; Driver: P2)

| # | Äußerung | *BKM Code* | *DesignIssue Code* |
|---|---|---|---|
| 1 | P2: Wie ist denn die vorhandenen Datenstrukturen, die für die GUI verwendet werden? Ist da ein `CollumnAttribute` da? | *P2.ask_knowledge* | |
| 2 | P2: Ansonsten würd' ich einfach die verwenden. | *P2.propose_design* | *P2.Proposal* |
| 3 | P1: Ich weiss nicht, wovon du gerade redest. | *P1.explain_standard of knowledge* | *P1.Disagree* |
| 4 | P2: Von dem, was ich gemacht habe. | *P2.explain_knowledge* | *P2.ITDC* |
| 5 | P1: Du hattest nur ein virtuelles Attribut. | *P1.explain_knowledge* | *P1.ITDC* |
| 6 | P2: Ich hatte eigentlich gesagt, dass wir das für die Datenstruktur verwenden wollten, was ich gemacht habe. Aber OK. | *P2.explain_knowledge* | *P2.ITDC* |
| 7 | P1: Wir können erstmal weitergehen. Es ist ja noch nicht festgebunden, was ich gemacht habe. | *P1.propose_step* | |
| 8 | P2: Irgendwann sollen wir es zusammen bringen. | *P1.amend_step* | |

Tabelle 21: Die Situation findet am Anfang der Episode statt. Die P1 erklärt der P2 ihre bisherigen Arbeiten bezüglich des bearbeiteten Codestückes. In dieser speziellen Situation versucht jedoch die P2, ihre bereits geleistete Arbeit vorzustellen. (FA02 [3:409]; Driver: P1)

| # | Äußerung | *BKM Code* | *DesignIssue Code* |
|---|---|---|---|
| 1 | P1: Das ist ein Test. | *P1.explain_knowledge* | *P1.ITDC* |
| 2 | P1: Das müssen wir erstmal überbügeln. | *P1.propose_design* | *P1.Proposal* |
| 3 | P2: Hätte nicht Mock mehr Sinn gemacht? | *P2.propose_design* | *P2.Proposal* |
| 4 | P1: Das habe ich nicht geschrieben ... | *P1.explain_knowlede* | *P1.ITDC* / *P1.Disagreement* |
| 5 | P2: Ah! Ok. | *P2.explain_standard of knowledge* | |
| 6 | P1: ... und ich pass' das jetzt nicht gross irgendwelche Tests von irgendwelchen Leuten an. | *P1.disagree_design* | |

Tabelle 22: Das Paar hat gerade ein Refactoring mit Hilfe eines Refactoring-Werkzeugs durchgeführt. Das Werkzeug konnte aber nicht alle Codestellen anpassen. So bearbeitet das Paar die Fehlermeldungen aus dem Build-Bericht.
(FA02 [1:419]; Driver: P1)

| # | Äußerung | *BKM Code* | *DesignIssue Code* |
|---|---|---|---|
| 1 | P1: Wir könnten, wir könnten ... | *P1.explain_knowledge* | |
| 2 | P2: Es ist ja doof ... | *P2.explain_standard of knowledge* | |
| 3 | P1: `Transform` ... Moment, was macht denn der? Der erzeugt ... | *P1.explain_standard of knowledge* | *P2.ITDC* |
| 4 | P2: ... ein Array, das unter Umständen ... | *P2.explain_knowledge* | |
| 5 | P1: ... `GroupMenuItems`, `ActionGroupId` ... | *P1.explain_knowledge / P1.amend_activity / P1.read_sth* | |
| 6 | P1: Wollen wir eine Liste daraus machen? | *P1.propose_design* | *P1.Proposal* |
| 7 | P2: Dann geht das `Transform` wohl nich mehr. | *P2.explain_knowledge* | *P2.ITDC / P2.Disagreement* |
| 8 | P1: Was wir auch machen können, ist `ArrayUtilities`. | *P1.propose_design* | *P1.Proposal* |
| 9 | P2: Genau, `concat` oder so. | *P2.admend_design* | *P2.Amendment* |
| 10 | P1: Genau. | *P1.agree_design* | *P1.Agreement* |

Tabelle 23: Das Paar fügt neue Funktionalität hinzu. Um diese Funktionalität an den bestehenden Code anzupassen, braucht das Paar eine geeignete Datenstruktur, die an die Schnittstellen passt. (FA04 [1:50]; Driver: P2)

# 5 Zusammenfassung und Diskussion der Ergebnisse

Die vorgestellten Konzepte geben einen detaillierteren Einblick in die Erstellung des Entwurfs während einer PP-Sitzung. Sie erlauben es, eine Beschreibung vom Entwurfsprozess während der PP zu erstellen. Diese Beschreibung wurde dazu verwendet, um wiederkehrende, klassifizierbare Vorgehensweisen aufzuspüren. Diese Vorgehensweisen wurden unter dem Begriff *DesignEpisode* zusammengefasst.

Die Episoden wurden auf Faktoren analysiert, die das Auftreten einer bestimmten Episode beeinflussen.

Es wurden ebenso potenzielle Auswirkungen der jeweiligen Episoden betrachtet. Diese wurden entlang einigen in [9] präsentierten Faktoren untersucht. In dieser Diplomarbeit wurde so die Gruppenleistung als Teil der Wirtschaftlichkeit untersucht, darüber hinaus wurde der Problemlösungsprozess, der Lernprozess, so wie die Qualität des Entwurfs berücksichtigt.

Die gefundenen Episoden können in zwei disjunkte Untermengen, die die Verteilung des Entwurfsprozesses unter den beiden Programmierern wiedergeben, aufgeteilt werden: die einseitigen Episoden (auch „aktiv-passiv“) und die beidseitigen Episoden (auch „aktiv-aktiv“). Die einseitigen Episoden zeichnet aus, dass der ganze Entwurf von einer Person geschaffen wird und die andere an diesem Prozess nur passiv teilnimmt. Zu dieser Menge gehören die Episoden *Single-Handed* und *Guidance*. In den beidseitigen Episoden nehmen beide Programmierer an dem Entwurfsprozess aktiv teil. Die Art ihres Einflusses auf den Entwurf hängt im Detail vom konkreten Episodentyp ab.

Anhand der Auswertung der Ergebnisse liess sich die Aufteilung des Entwurfsprozesses im Paar in den unterschiedlichen Episoden wie folgt darstellen.

| *Episodenname* | Aufteilung des Entwurfsprozesses im Paar |
|---|---|
| *Single-Handed* | einseitig |
| *Guidance* | einseitig |
| *DirectConsensus* | beidseitig |
| *Complement* | beidseitig |
| *Conflict* | beidseitig |

Tabelle 24: Episoden - Zusammenfassung: Einseitig - Beidseitig

Jede Episode bezieht sich auf einen bestimmten Teil des Produkts, der mit dem Konzept *DesignElement* beschrieben wird. Mit dem *DesignElement* ist in jeder Episode ein Problemlösungsprozess verbunden, so muss vom Paar ein Entwurf bzgl. des *DesignElement* geschaffen werden. Dieser Prozess wurde in dieser Diplomarbeit als ein verteilter Kognitionsprozess betrachtet, so wie es von Flor und Hutchins beschrieben wurde [16]. Der Verlauf dieses Prozesses sowie seine Verteilung gehörte zu den untersuchten Aspekten der PP.

In den einseitigen Episoden ist der Prozess nur bei einer Person eindeutig feststellbar. Wegen der mangelnden Aktivität der passiven Person konnte der Verlauf des Problemlösungsprozesses bei ihr nicht betrachtet werden.

In den beiseitigen Episoden konnte das Vorhandensein eines Problemlösungsprozesses bei beiden Personen im Paar festgestellt werden. Die zeitliche Verteilung von diesem Prozess so wie der gegenseitige Einfluss der Prozessteile der beiden Personen spielt so in den beidseitigen Episodentypen eine Rolle. Den Episodentyp *DirectConsensus* zeichnet ein parallel verlaufender Problemlösungsprozess ohne gegenseitigen Einfluss aus. Andere beidseitige Episoden sind durch einen gegenseitigen Einfluss mit zeitlicher Parallelität charakterisiert. Die Ausnahme bildet hier der Episodentyp *Complement* mit asymmetrischer Rollenverteilung. Hier konnte die zeitliche Parallelität nicht eindeutig festgestellt werden.

| *Episodenname* | Problemlösungsprozess | | |
|---|---|---|---|
| | Personen | Zeitliche Parallelität | Gegenseitiger Einfluss |
| *Single-Handed* | Aktive Person | ? | Nein |
| *Guidance* | Aktive Person | ? | Nein |
| *DirectConsensus* | Beide Personen | Ja | Ja |
| *Complement (asymm.)* | Beide Personen | ? | Ja |
| *Complement (symm.)* | Beide Personen | Ja | Ja |
| *Conflict* | Beide Personen | Ja | Ja |

Tabelle 25: Episoden - Zusammenfassung: Verlauf des Problemlösungsprozesses

Die Beschreibung einer Episode mit den *DesignIssue* liefert ebenso Hinweise auf den Episodentyp. Das Vorhandensein von bestimmten *DesignIssue*-Codes determiniert in manchen Fällen den Typ der Episode.

Die Häufigkeit vom Auftreten eines bestimmten Episodentypes ist von bestimmten Faktoren beeinflusst. Während der Untersuchung im Rahmen dieser Arbeit wurden zwei Faktoren gefunden. Der erste ist die Art der Zusammensetzung des Paars. Während dieser Arbeit wurden nur Paare berücksichtigt, in denen mindestens ein erfahrener Entwickler („Experte") einen Teil des Paars bildet. So wurden die Paare „Experte-Anfänger" und „Experte-Experte" untersucht. Der Grund dafür ist die angenommene größte Relevanz einer solchen Paarung im professionellem Umfeld. Dieser Faktor kann als Teil des Kontexts einer *DesignEpisode* betrachtet werden. Im untersuchten Datenmaterial fiel auf, dass bei der Kombination "Experte-Anfänger" verstärkt die Epsiode *Guidance* oder *Conflict* (Folgschaft) auftrat. Dies läßt sich vermutlich dadurch begründen, dass der Anfänger nicht über das nötige Wissen verfügt, um im Falle eines Konflikts seinen Einspruch fundiert begründen und damit seine Position festigen zu können. Im Fall *Guidance* kann man davon ausgehen, dass der erfahrenere Entwickler sich der geringeren

| *Episodenname* | *DesignIssue* Code vorhanden | | |
|---|---|---|---|
| | *ITDC* | *Amendment* | *Disagreement / Challenge* |
| *Single-Handed* | Nein | Nein | Nein |
| *Guidance* | Ja | Nein | Nein |
| *DirectConsensus* | Optional | Nein | Nein |
| *Complement (asymm.)* | Ja | Ja | Nein |
| *Complement (symm.)* | Optional | Ja | Nein |
| *Conflict* | Optional | Optional | Ja |

Tabelle 26: Episoden - Zusammenfassung: Inhalt der Episode

Berufserfahrung des Partners bewusst ist und damit die Nachvollziehbarkeit seines Lösungsansatzes verdeutlichen möchte. Bei der Konstellation "Experte-Experte" traten aufgrund der vergleichbaren Kenntnisse und Fähigkeiten vor allem die Episoden vom Typ *DirectConsensus*, *Complement (symm.)* und *Conflict (Lösungsverschmelzung)* auf. Welcher dieser Typen gewählt wurde, hing u.a. von der Art des Problems ab.

| *Episodenname* | Tritt häufiger bei Paarkonfiguration auf |
|---|---|
| *Guidance* | Experte-Anfänger |
| *DirectConsensus* | Experte-Experte |
| *Complement (symm.)* | Experte-Experte |
| *Conflict (Folgschaft)* | Experte-Anfänger |
| *Conflict (Lösungsverschmelzung)* | Experte-Experte |

Tabelle 27: Episoden - Zusammenfassung: Kontext - Zusammensetzung des Paars

Der zweite Faktor ist die subjektive Einschätzung der Schwierigkeit des zu lösenden Problems durch den aktiven Entwickler. Aus Gründen der Einfachheit wurde bei den Problemen nach Hinweisen gesucht, wo mindestens ein Entwickler das zu lösende Problem als "Einfach" oder "Komplex" einschätzt. Dies geschah in dem untersuchten Material vor allem aufgrund der Analyse der verbalen Äußerungen sowie der eigenen Einschätzung des Schwierigkeitsgrades des Problems.

Die antizipierten Auswirkungen der unterschiedlichen Episoden auf die Leistung des Paars, so wie auf die Qualität des Entwurfs und auf den Lernprozess wurden analysiert.

Die Leistung des Paars beschreibt, wie schnell das Paar einen Entwurf erstellen und umsetzen kann. Die Paarleistung konnte nicht absolut untersucht werden, da es sich hier um ein sogenanntes *wicked problem* handelt [1]. Es wurde bei der Analyse

| *Episodenname* | Tritt häufiger auf bei Problem des Schwierigkeitsgrades |
|---|---|
| *Single-Handed* | Einfach |
| *Guidance* | Einfach |
| *DirectConsensus* | Einfach |
| *Complement (symm.)* | Komplex |
| *Conflict (Verweis an Externe)* | Komplex |
| *Conflict (Lösungsverschmelzung)* | Komplex |

Tabelle 28: Episoden - Zusammenfassung: Kontext - Einschätzung der Schwierigkeit des zulösenden Problems

nach Hinweisen gesucht, die helfen könnten, die Leistung während der Entwicklung in Paaren in Relation zur separaten Entwicklung durch eine Person einzuschätzen. Die Hinweise wurden hauptsächlich nach dem Modell der Gruppenleistung von Hackmann und Morris [43] analysiert. Sie charakterisierten die Gruppenproduktivität als die Summe der potenziellen Produktivität und der Prozessgewinne abzüglich der Prozessverluste. Bei dieser Analyse wurden Indizien für die potenziellen Prozessgewinne und -verluste gesucht, die aufgrund von Motivations- und Koordinationsänderungen zustande kommen.

| *Episodenname* | Antizipierte Paarleistung |
|---|---|
| *Single-Handed* | Wie separat |
| *Guidance* | Schlechter |
| *DirectConsensus* | Wie separat |
| *Complement* | Besser |
| *Conflict (Aufschiebung)* | Schlechter |
| *Conflict (Verweis an Externe)* | Besser |
| *Conflict (Folgschaft)* | Besser |
| *Conflict (Lösungsverschmelzung)* | Schlechter |

Tabelle 29: Episoden - Zusammenfassung: Auswirkungen - Antizipierte Paarleistung

In den Episoden vom Typ *SingleHanded* ist die Leistung des Paars vergleichbar mit der, die ein einzelner Programmierer erbringt. Ein Grund dafür ist die mangelnde Aktivität der passiven Person bei dem Problemlösungsprozess. Es ist denkbar, dass bei der passiven Person der kognitive Prozess des Problemlösens abläuft. Dieser ist aber

nicht verteilt, wenn man die Bedingungen der verteilten Kognition berücksichtigt (siehe Grundlagen - Verteilte Kognition). Der Koordinationsverlust tritt in diesem Episodentyp nicht auf.

Eine ähnliche Situation kommt bei den Episoden vom Typ *Guidance* vor. Die erbrachte Leistung stammt ebenso hauptsächlich von einer Person. In dieser Episode ist aber ein Koordinationsbedarf merkbar. Die aktive Person benötigt Zeit, um der passiven Person den Designkontext näherzubringen. Dies kann als ein Prozessverlust aufgrund der Koordination aufgefasst werden. Deswegen wird die Paarleistung vom Episodentyp *Guidance* schlechter im Vergleich zur separaten Entwicklung eingeschätzt.

Bei diesen beiden einseitigen Episoden besteht das Risiko des Prozessverlustes aufgrund der mangelnden Arbeitsmotivation der passiven Person. Diese waren in der Datenbasis während der Untersuchung nicht merkbar, können aber langfristig auftreten, sollten die einseitigen Episoden einen grossen Teil der Designepisoden ausmachen. Der Grund dafür ist, dass die Verantwortung für den Erfolg (einen Entwurf zu erstellen und umzusetzen) hauptsächlich bei der aktiven Person liegt und damit das Empfinden des Erfolgs bei der passiven Person vermindert auftreten kann [20]. Beim Episodentyp *Guidance* kann zwar kurzfristig die Motivation des passiven Entwicklers durch Erfolg beim Nachvollziehen des Problems als motivationsfördernder Faktor nach dem Zwei-Faktoren-Modell von Herzberg [20] gesteigert werden. Langfristig kann jedoch dieser Effekt nicht gegeben sein, da der Lernfortschritt aufgrund der begrenzten Menge der Informationen innerhalb der Organisation seine Grenzen hat.

Die Episoden vom Typ *DirectConsensus* sind durch eine vergleichbare Leistung in Bezug auf einen einzelnen Programmierer gekennzeichnet. Der Problemlösungsprozess tritt bei beiden Programmierern zeitlich parallel auf, und beide Programmierer finden die gleiche Lösung zeitnah. Die Koordination, die bei dieser Episodenart auftritt, ist gering. Es ist naheliegend, dass die langfristige motivationale Wirkung dieser Episode positiv ist, da die Verantwortung für den Erfolg auf beiden Seiten liegt. Darüber hinaus kann die schnelle Rückmeldung der Zustimmung des Partners, verbalisiert durch die Angabe der gleichen Lösung, ebenso als motivationsfördernd betrachtet werden.

Die *Complement*-Episoden zeichnet eine bessere Leistung aus als die, die ein einzelner Programmierer separat erbringt. Der hier vorgefundene Koordinationsaufwand ist meistens zielgerichtet (u.a. Erweiterung der Lösung) und sein negativer Einfluss auf die gemeinsame Leistung ist gering. Positiv wirkt sich dafür die Art der Verteilung des kognitiven Prozesses auf die Leistung des Paars aus. Diese ist innerhalb beider Programmierer verteilt, und die Teilergebnisse der Problemlösungsprozesse der beiden Programmierer erweitern die Menge des verfügbaren Wissens (nach der Definition des Problemraumes von March und Simon) des Partners. Dies kann über die Eingrenzung des zu durchsuchenden Problemraums zu einem schnelleren Auffinden des Zielzustandes führen.

Der Episodentyp *Conflict* kann durch unterschiedliche antizipierte Paarleistungen charakterisiert werden, da der Verlauf eines Konfliktes sehr unterschiedlich sein kann. Im Rahmen dieser Diplomarbeit wurden daher vor allem Konfliktlösungsstile berücksichtigt. Diese wurden von den Konfliktstilen von Schwarz inspiriert [7]. Bei der Anal-

yse der verschiedenen Konflikte stellte sich die Frage, ob es Stile gibt, die besonders effektiv zur Lösung eines Konfliktes und zum Erarbeiten eines Entwurfes taugen. Da keine Auffälligkeiten bei der Untersuchung des ganzen Konfliktes auftraten, wurde nur der letzte im Konflikt angewendete Stil berücksichtigt, da dieser zur Lösung des Konfliktes und meistens auch zur Erarbeitung eines Entwurfes führte. Während einer *Conflict*-Episode ist der kognitive Problemlösungsprozess ähnlich verteilt wie im Fall der *Complement*-Episode. Die *Conflict*-Episoden weisen jedoch meistens einen höheren Koordinationsaufwand auf, da dieser häufig nicht zielgerichtet ist, sondern durch die Notwendigkeit der Überwindung der Meinungsunterschiede zwischen den beiden Partnern zusätzlich beeinflusst wird.

Die Paarleistung der Konflikte, die durch Aufschiebung gelöst wurden, ist nicht vorhanden. Das Paar kann zwar einen fortgeschrittenen Problemlösungsprozess zustandebringen, der Zielzustand wird aber nicht erreicht. Somit kann hier maximal von der Erweiterung des verfügbaren Wissens bei den Beteiligten gesprochen werden, nicht aber von einer erbrachten Leistung im Sinne des erstellten und umgesetzten Entwurfes. Konflikte, die durch das Anwenden des Stils Verweis an Externe gelöst wurden, führten in der vorhandenen Datenbasis meistens direkt zur Lösung. Das hatte den Grund, dass die benannte Externe Instanz, meistens als Autorität angesehen werden konnte. So verwiesen die Programmierer vor allem auf Absprachen oder Anforderungen.

Das Lösen eines Konflikts durch den Stil Folgschaft führte meistens auch zu einer vergleichbar schnellen Lösung. Diese Episode trat vor allem bei der Paarzusammensetzung Experte-Anfänger auf, wobei der Experte den Entwurf erarbeitet hat. In diesem Fall ist der Koordinationsaufwand bei dem Autor eines Lösungsansatzes darauf gerichtet, die eigene Lösung zu verteidigen. Dies geschah meistens innerhalb kurzer Zeit. Der Grund dafür ist vermutlich die hohe fachliche Glaubwürdigkeit, die einem Experten zugewiesen wird.

Die *Conflict*-Episoden, in denen der Konfliktstil Lösungsverschmelzung zur Lösung führte, zeichneten eine geringe Leistung aus. Beide Seiten des Konfliktes mussten ihre ursprünglichen Lösungen korrigieren, d.h. im gemeinsamen Problemlösungsprozess haben die Teilprozesse bei jedem der Programmierer zu einer Teillösung geführt, die jedoch einen Irrtum aufwies.

Während dieser Arbeit wurde ebenso die antizipierte Qualität des erstellten Entwurfes analysiert. Diese wurde relativ zu der Qualität gesehen, die von einem Programmierer erzeugt werden könnte.

Bei der Analyse der Qualität wurde angenommen, dass das Durchsuchen eines grösseren Problemraums mit einer höheren Wahrscheinlichkeit zu einem besseren Entwurf führen wird.

Somit spielt die Art der Verteilung des kognitiven Problemlösungsprozesses eine entscheidende Rolle. Die einseitigen Episoden resultieren in der gleichen antizipierten Qualität des Entwurfs wie die, die durch einen Programmierer separat erschaffen wurde. Der Problemlösungsprozesses ist hier nur bei einem der Programmierer direkt sichtbar.

Die Episoden vom Typ *DirectConsensus* resultieren in einer höheren erwarteten Qualität des Entwurfes.

| *Episodenname* | Antizipierte Qualität des Entwurfs |
|---|---|
| *Single-Handed* | Wie separat |
| *Guidance* | Wie separat |
| *DirectConsensus* | Besser |
| *Complement* | Besser |
| *Conflict (Verweis an Externe)* | Besser |
| *Conflict (Lösungsverschmelzung)* | Besser |

Tabelle 30: Episoden - Zusammenfassung: Auswirkungen - Antizipierte Qualität des Entwurfs

Der Punkt, dass beide Entwickler zur gleichen Zeit auf die gleiche Lösung kommen, kann ein starker Hinweis für die Korrektheit dieser Annahme sein. Unter der Annahme, dass die Wahrscheinlichkeit zur Erstellung einer hochqualitativen Lösung bei beiden Programmierern $> 1/2$ ist, deutet die unabhängige Erstellung der gleichen Lösung von beiden Programmierern auf eine noch höhere Wahrscheinlichkeit, dass diese Lösung korrekt ist (siehe Bayessches Theorem z.B. in [23]).

Die *Complement*-Episoden führen vermutlich ebenso zu einer höheren Qualität des Entwurfes. Auch hier ist der Grund in der Verteilung des Problemlösungsprozesses zu finden. Dieser ist auf die zwei Entwickler verteilt, deren Zwischenergebnisse so immer weitergeführt werden.

In den *Conflict (Verweis an Externe)*-Episoden wird auch eine höhere Qualität des Entwurfs erwartet. Dies basiert auf der Annahme, dass die Externe Instanz, auf die sich der Vorschlag bezieht, vertrauenswürdig ist, und dass diese Instanz sehr wahrscheinlich mit ihren Äußerungen im Recht ist. Diese Annahme beruht auf den untersuchten Daten. In jeder dieser Episoden vom Typ *Conflict (Verweis an Externe)* führte das Anwenden dieses Konfliktstils zur einer zeitnahen Beendigung der Episode. Auf die bessere Qualität des Entwurfs deutet die in diesen Konflikten vorhandene Verteilung des Problemlösungsprozesses. Diese resultiert in einer höheren Gesamtzahl der Wissenszustände, die vom Paar analysiert werden, so wie in einer grösseren Wissensmenge, die bei dieser Problemlösung zugänglich ist. So steigt die Wahrscheinlichkeit, dass diese Lösung, die aus einer glaubwürdigen Quelle stammt, erreicht wird.

Die Episoden des Typs *Conflict (Lösungsverschmelzung)* erzeugen eine bessere Qualität des Entwurfs, als es bei einem separaten Entwickler der Fall wäre. In Episoden dieser Art wird das Ergebnis von beiden Personen erarbeitet. Dies geschieht in einem Konflikt, d.h. es musste innerhalb der Episode zumindest zu einem Widerspruch gekommen sein. Dies kennzeichnet den Moment, in dem eine explizite Qualitätsverbesserung stattfindet. So sind die Art der Problemlösungsprozessverteilung und das Vorhandensein einer expliziten Qualitätsverbesserung Indizien dafür, dass die Qualität vom erarbeiteten Entwurf besser ist.

Die letzte Auswirkung, die bei der Untersuchung berücksichtigt wurde, war der Lernprozess bei der PP. Das Lernen wurde bezüglich der Inhalte in zwei Gruppen aufgeteilt. So wurde das Erlernen des Kontexts des Projekts und der aktuellen Lösung samt dazugehöriger Vorgehensweise analysiert. Diese Faktoren sollten Hinweise darauf geben, wie effektiv dieser Prozess verläuft. Bei der Analyse wurde die Annahme gemacht, dass das Erlernen der Lösung ohne die Kenntnis des dazugehörigen Kontexts nicht effektiv sei. Bei dem Lernprozess wurde die Dauer der Betrachtung des Problems, sowohl visuell, als auch auditiv, als ein wichtiger Determinant des Effektes des Lernens betrachtet. Dies beruht auf den Eigenschaften der Lernkurve, die steigt, je länger gelernt wird [12].

| *Episodenname* | Antizipierte Effektivität des Lernprozesses |
|---|---|
| *Single-Handed* | Eingeschränkt |
| *Guidance* | Gut |
| *DirectConsensus* | Kaum |
| *Complement (asymm.)* | Gut |
| *Complement (symm.)* | Eingeschränkt |
| *Conflict (Aufschiebung)* | Eingeschränkt |
| *Conflict (Verweis an Externe)* | Gut |
| *Conflict (Folgschaft)* | Potenziell gut |
| *Conflict (Lösungsverschmelzung)* | Potenziell gut |

Tabelle 31: Episoden - Zusammenfassung: Auswirkungen - Antizipierte Effektivität des Lernprozesses

Der Lerneffekt während der Episoden vom Typ *Single-Handed* ist stark eingeschränkt. Die aktive Person erzeugt in diesem Episodentyp den Entwurf eigenständig. Dies geschieht meist ohne jegliche zusätzliche Erklärung. Die passive Person verfolgt ihr Vorgehen. Sollte die passive Person bereits den Kontext dieser Lösung ausreichend kennen, kann bei ihr ein eingeschränkter Lernprozess auftreten. Sie sieht, wie innerhalb dieses gegebenen Kontexts die aktive Person vorgeht und welche Lösung in solch einer Situation angewendet wird. Sollte die passive Person jedoch den Kontext nicht ausreichend kennen, kann sie die gefundene Lösung nicht nachvollziehen. Zusätzlich sind die Episoden dieses Typs meist von kurzer Dauer. Daher ist die Möglichkeit des Lernens für die passive Person nur eingeschränkt vorhanden.

Bei den Episoden des Types *Guidance* ist ein guter Lerneffekt zu erwarten. In diesen Episoden orientieren sich die Aktivitäten der aktiven Person in zwei Richtungen: der passiven Person den Kontext dieser Episode näher zu bringen und einen Lösungsansatz zu erstellen. Die passive Person begrenzt sich hier auf Wiederholungen und Nachfra-

gen, nimmt jedoch nicht direkt an der Erstellung des Entwurfs teil. Die passive Person kann somit sowohl die gefundene Lösung als auch den Weg dahin besser nachvollziehen. Sollte die passive Person die Lösung dennoch nicht nachvollzogen haben, kann sie dies in Fragen äußern, was den Lerneffekt zusätzlich verstärkt. In einigen *Guidance*-Episoden wiederholt die passive Person Teile der Informationen, die sie von der aktiven erfahren hat. Das kann als das Hervorrufen des Rezenzeffektes interpretiert werden [12].

Die *Complement*-Episoden mit asymmetrischer Rollenverteilung zeichnet ebenso ein guter antizipierter Lerneffekt aus. In diesem Episodentyp erweitert eine Person die gemeinsame Wissensbasis, und die andere erarbeitet u.a. anhand dieser Wissensbasis einen Lösungsansatz. Dank der Einführung in den Designkontext bekommt die Person, die den Lösungansatz erstellt, die Möglichkeit, diesen nachzuvollziehen, was zusätzlich durch das Vorhandensein von interaktiven Elementen, wie im Falle des Episodentyps *Guidance*, verstärkt werden kann. Zusätzlich wird der Lerneffekt bestätigt, d.h. es wird u.a. anhand der Einführung in den Designkontext ein Lösungsansatz erarbeitet. Dies kann für die Person, die den Lösungsansatz erarbeitete, als positiver Verstärker im Sinne der operanten Konditionierung gedeutet werden [24].

Bei den *Complement*-Episoden mit symmetrischer Rollenverteilung kann der Lerneffekt eingeschränkt sein. Die Ursache dafür ist, dass dieser Episodentyp das Auftreten ebenbürtiger Partner bzgl. der Fähigkeiten und der Kenntnisse des Kontexts benötigt.

In den *Conflict (Aufschiebung)*-Episoden ist der Lerneffekt ebenso eingeschränkt. Das kommt daher, da sie nicht zu einer Lösung führen, womit keiner eine erfolgreiche Problemlösung vorgeführt bekommt.

*Conflict (Verweis an Externe)*-Episoden charakterisiert eine Lösung, die u.a. anhand eines Bezugs auf eine externe Instanz vom Paar gefunden wurde. Bei dieser Instanz handelt es sich meistens um eine glaubwürdige Quelle. Der antizipierte Lerneffekt in dieser Episode wurde als gut eingeschätzt, da die beiden Personen die Möglichkeit bekommen, sowohl die Lösung als auch ihre Gründe (u.a. den, der von der externen Instanz stammt) kennenzulernen.

Der Lerneffekt der weiteren *Conflict*-Episoden hängt stark von ihrem Inhalt und Verlauf ab. Das Untermauern der Vorschläge mit Argumenten kann so z.B. als lernfördernd angesehen werden.

All diese Aspekte sollten im Kontext der Ziele der Sitzung betrachtet werden. Je nach Projektart und Verlauf ist es denkbar, dass der beabsichtigte Effekt unterschiedlich ausfällt. Es ist z.B. naheliegend, dass in Projekten mit grosser Mitarbeiterfluktuation u.a. die Lernaspekte der Paarprogrammierung besonders interessant sind. Wiederum kann in Projekten, deren Ziel es ist, Software zu erstellen, die eine besonders hohe Qualität aufweisen soll (z.B. missionskritische Systeme), die erzielte Qualität der wichtigste Effekt der PP sein.

Mit dieser Betrachtung können die Handlungen der beiden Programmierer während eines Konflikts unter Bezug auf die Nutzenorientierungsfunktionen analysiert werden. In dieser Diplomarbeit wurden zwei ausgewählte Nutzenorientierungsfunktionen der Handlungen beider Personen berücksichtigt. Es wurden somit Indizien für koopera-

tives und kompetitives Handeln gesucht.

Ein kooperativer Konflikt strebt die bestmögliche Lösung an. Die Art der bestmöglichen Lösung hängt von den Zielen der Sitzung ab. So kann es Sitzungen geben, bei denen die Leistung beim Erstellen und dem Umsetzen des Entwurfs von besonderer Bedeutung ist. Andersherum kann es Sitzungen geben, wo die Qualität des Entwurfs wichtig ist. Bei jeder dieser Sitzungen würde ein kooperativer Konflikt anders aussehen. Steht die Leistung vom Paar im Vordergrund ist es naheliegend, dass die Programmierer öfter den Konfliktstil „Folgschaft" oder „Verweis an Externe" anwenden sollen. Liegt die Betonung auf der Qualität der Lösung ist der Konfliktstil „Lösungsverschmelzung" von besonderem Interesse. Aus diesem Grund ist die Kooperativität des Paars von den Zielen der Sitzung abhängig. Die erwähnten Eigenschaften einer Konfliktepisode - die Existenz einer expliziten Einigung und die Begründung eigener Vorschläge - können als kooperationsfördernd angesehen werden. Die Existenz einer expliziten Einigung kann beiden Programmierern mehr Vertrauen in die Korrektheit der Lösung geben. Dieses Vertrauen erlaubt es, sich vollständig auf die folgenden Probleme zu konzentrieren, anstatt an der Korrektheit der Lösung zu zweifeln. Die Begründung eigener Vorschläge gibt dem Partner mehr Chancen auf die Nachvollziehbarkeit der Lösungen und Lösungswege. So kann seine Leistung einfacher bei der Lösung des Problems eingesetzt werden.

Bei der Analyse des kooperativen Konflikts wurden vor allem Koordinationsverluste berücksichtigt. Die Fertigkeits- und Motivationsverluste werden hier kaum beachtet. Das ist mit den Schwierigkeiten solcher Analysen aus dem vorhandenen Datenmaterials zu begründen.

In den kompetitiven Konflikten weist mindestens einer der Programmierer ein kompetitives Verhalten auf. So versucht er, die Episode zu einer Lösung zu bringen, in der seine Leistung und seine Fähigkeiten im Vordergrund stehen, wobei die vom Partner minimiert oder nicht berücksichtigt werden. So kann das Ausbleiben einer expliziten Einigung ein Hinweis auf einen kompetitiven Konflikt sein, da eine Zustimmung die Leistung des Partners anerkennen und damit der Leistungsunterschied nivelliert werden würde. Ebenso kann ein kritischer Bezug auf die Aussagen vom Partner, ohne diese mit fachlichen Argumenten zu stärken, ein Indiz auf einen kompetitiven Konflikt sein. Auf diese Art und Weise kann ein kompetitiv handelnder Programmierer versuchen, die Leistungen oder Fähigkeiten vom Partner zu minimieren. In einem kompetitiven Konflikt wird der Konfliktstil „Lösungsverschmelzung" nicht angewendet. Um die eigene Leistung und Fähigkeiten zu betonen, sollte der Konfliktstil „Folgschaft" angewendet werden. Sollte dieser zu keiner Einigung führen, sollte die Leistung vom Partner minimiert werden. Dies lässt sich entweder durch den Konfliktstil „Verweis an Externe" oder „Aufschiebung" erreichen.

Demzufolge können je nach Ziel der Sitzung und Einschätzung der Schwierigkeit der Aufgabe die Paare zusammengestellt werden. Zusätzlich können die Entwickler über die Ziele der Sitzung und den Wissenstand vom Partner aufgeklärt werden. In dieser Situation könnte die bewußte Auswahl bestimmter Episodentypen helfen, diese Ziele zu erreichen.

# 6 Grenzen der Arbeit und Ausblick

Diese Diplomarbeit befasste sich mit der Charakterisierung des Entwurfsprozesses bei der Paarprogrammierung. Das Thema ist sehr komplex und wurde bei weitem nicht in Rahmen dieser Arbeit erschöpft. Das erste Verbesserungspotenzial bietet die verwendete Methodik, da die im Rahmen dieser Diplomarbeit betriebene Forschung vor allem die Prozeduren des offenen und teilweise axialen Kodierens nutzte. Es gibt jedoch noch die Möglichkeit, auf eine systematische Weise durch das Anwenden der Prozedur des selektiven Kodierens eine wissenschaftliche Theorie zu erstellen.

Eine weitere Begrenzung bildet die Menge und Art der verwendeten Daten. Diese konnte zwar für diese explorative Erstellung der Beschreibung des Entwurfsprozesses ausreichend sein, müsste aber für eine fundiertere Beschreibung und optionale Theorie stark erweitert werden.

Die in dieser Arbeit betrachteten Auswirkungen der Episoden betrafen nur die direkten und auch direkt sichtbaren Auswirkungen, was ebenso zu einer starken Einschränkung der Arbeit führt. Es ist denkbar, dass die mittel- oder langfristigen Effekte der Kognition (u.a. Emotion, Aufmerksamkeit, Problemlösen) nach bestimmten Episoden auftreten, und diese dann auf weiteren Verlauf der Sitzung oder sogar auf die gesamte Zusammenarbeit Einfluss nehmen.

Zusätzlich wurden die persönlichkeitstheoretischen Aspekte der zusammenarbeitenden Personen nicht betrachtet. Der Grund dafür ist die Form der Datenbasis. Diese müsste von Anfang an unter Berücksichtigung der Untersuchung der Persönlichkeiten der Programmierer erfasst werden.

Manche Thesen der von Cockburn beschriebenen Effekte der Paarprogrammierung [9] waren mit der gewählten Forschungsmethodik kaum zu untersuchen. So z.B. benötigt die These „der endgültige Anteil der Defekte im Quellcode ist niedriger“ Datenmaterial, in dem ähnliche Aufgaben in PP-Sitzungen und separat gelöst wären. Dem müsste eine quantitative Untersuchung folgen.

Die Leistung des Paars wurde als Menge des umgesetzten Designs innerhalb einer gewissen Zeit definiert. Dies hat den Grund, dass im vorhandenen Datenmaterial nur Sitzungen zu finden waren, die höchstwahrscheinlich zu Projekten gehören, die mit XP oder XP-ähnlicher Vorgehensweise organisiert waren. Diese Vorgehensweisen charakterisieren sich durch kurze Iterationen der Entwicklung. In dem vorhandenen Datenmaterial kam es in keiner der Sitzungen vor, dass ein Entwurf erstellt, festgehalten und nicht umgesetzt wurde. Die Einschätzung der Leistung vom Paar wurde aus den Indizien für die Leistungsänderung in Relation zu der angenommenen Leistung eines einzelnen Programmierers abgeleitet.

Es wurde ebenso die Qualität des erstellten Entwurfs betrachtet. Hier war es unmöglich, die Qualität objektiv einzuschätzen. Dies liegt u.a. daran, dass der architektonische Entwurf vom System und von der Software sowie die Anforderungen nur in einem sehr begrenzten Ausmass aus den vorhandenen Daten abzulesen war. Es wurde lediglich nach Hinweisen gesucht, die für eine stattgehabte potenzielle Qualitätsverbesserung oder bessere Qualität im Vergleich mit der separaten Erstellung des Entwurfs stehen.

Der Lernprozess gehörte zu den Aspekten der PP, die in dieser Diplomarbeit betrachtet wurden. Hier war das vorhandene Datenmaterial nicht ausreichend, um die eindeutigen Auswirkungen einer Sitzung oder einer Designepisode unter den Aspekten der bekannten Forschunsergebnisse im Bereich der Lernpsychologie zu untersuchen.

Die Untersuchung der Konflikte bei der PP könnte ebenso weiter fortgeführt werden. Diese könnten in einem größeren Rahmen, u.a. mit der Berücksichtigung weiterer Nutzenorientierungsfunktionen bei jedem der Programmierer betrachtet werden. Bei den (Conflict)-Episoden wurden ebenso die motivationalen Aspekte nicht ausreichend betrachtet.

Diese Diplomarbeit kann als Anregung für eine weitere interdisziplinäre Forschung der Paarprogrammierung dienen, wobei die sozialwissenschaftlichen Aspekte einen besonders starken und bis jetzt wenig untersuchten Einfluss zu haben scheinen.

# 7 Zusammenfassung

Diese Arbeit konnte belegen, dass es wiederkehrende, klassifizierbare Vorgehensweisen im Rahmen der Paarprogrammierung gibt. Es liessen sich fünf Episodentypen ableiten, die den Verlauf und Inhalt der Episode beschreiben und sich durch spezifische Eigenschaften auszeichnen. Je nach Art der Episode stehen bestimmte Auswirkungen der Paarprogrammierung im Vordergrund.

# Tabellenverzeichnis


1 Vergleich der Programmiersitzungen . . . 21
2 Single-Handed Beispiel 1 . . . 32
3 Single-Handed Beispiel 2 . . . 32
4 Single-Handed Beispiel 3 . . . 33
5 Single-Handed Beispiel 4 . . . 33
6 Guidance Beispiel 1 . . . 36
7 Guidance Beispiel 2 . . . 37
8 Guidance Beispiel 3 . . . 38
9 Guidance Beispiel 4 . . . 39
10 Consensus Beispiel 1 . . . 40
11 Consensus Beispiel 2 . . . 40
12 Consensus Beispiel 3 . . . 41
13 Complement Beispiel 1 . . . 44
14 Complement Beispiel 2 . . . 45
15 Conflict Beispiel 1 . . . 51
16 Conflict Beispiel 2 . . . 52
17 Conflict Beispiel 3 . . . 53
18 Conflict Beispiel 4 . . . 54
19 Conflict Beispiel 5 . . . 54
20 Conflict Beispiel 6 . . . 55
21 Conflict Beispiel 7 . . . 56
22 Conflict Beispiel 8 . . . 57
23 Conflict Beispiel 9 . . . 58
24 Episoden - Zusammenfassung: Einseitig - Beidseitig . . . 59
25 Episoden - Zusammenfassung: Verlauf des Problemlösungsprozesses . . . 60
26 Episoden - Zusammenfassung: Inhalt der Episode . . . 61
27 Episoden - Zusammenfassung: Kontext - Zusammensetzung des Paars . . . 61
28 Episoden - Zusammenfassung: Kontext - Einschätzung der Schwierigkeit des zulösenden Problems . . . 62
29 Episoden - Zusammenfassung: Auswirkungen - Antizipierte Paarleistung . . . 62
30 Episoden - Zusammenfassung: Auswirkungen - Antizipierte Qualität des Entwurfs . . . 65
31 Episoden - Zusammenfassung: Auswirkungen - Antizipierte Effektivität des Lernprozesses . . . 66


# Literatur


[1] A. Abran, P. Bourque, R. Dupuis, and J.W. Moore. *SWEBOK: Guide to the Software Engineering Body of Knowledge-SWEBOK*. IEEE Computer Society: Los Alamitos, California, 2004.

[2] A. Anderson, R. Beattie, K. Beck, D. Bryant, M. DeArment, M. Fowler, M. Fronczak, R. Garzaniti, D. Gore, B. Hacker, et al. Chrysler Goes to Extremes. *Distributed Computing*, 1(10):25–28, 1998.

[3] B.J. Baars. *The cognitive revolution in psychology*. The Guilford Press, 1986.

[4] K. Beck and C. Andres. *Extreme programming explained: embrace change*. Addison-Wesley Professional, 2004.

[5] K. Berkel. *Konflikttraining: Konflikte verstehen, analysieren, bewältigen*. IH Sauer-Verlag, 1997.

[6] Bertelsmann Wörterbuch der deutschen Sprache. http://www.wissen.de/wde/generator/wissen/ressorts/bildung/woerterbuecher/index.html.

[7] R. Böhm. Konfliktmanagement. *Eine Einführung. VÖGB/AK [Online] URL: www.voegb. at*, 27, 2007.

[8] T. Bonacker and P. Imbusch. Zentrale Begriffe der Friedens-und Konfliktforschung: Konflikt, Gewalt, Krieg, Frieden. *Friedens-und Konfliktforschung. Eine Einführung, Wiesbaden*, 2005.

[9] A. Cockburn and L. Williams. The costs and benefits of pair programming. *Extreme programming examined*, pages 223–248, 2001.

[10] J.M. Corbin and A. Strauss. Grounded theory research: Procedures, canons, and evaluative criteria. *Qualitative sociology*, 13(1):3–21, 1990.

[11] R.M. Cyert and J.G. March. *Behavioral theory of the firm*. Blackwell Publishing, 1992.

[12] H. Ebbinghaus. *Memory: A contribution to experimental psychology*. Teachers College, Columbia University, 1913.

[13] W. Edelmann. *Lernpsychologie*. Beltz, PVU, 2000.

[14] Working Group Software engineering body of knowledge, editor. *Information technology: Software life cycle processes*. Standards Australia, Homebush, Australia, 1997.

[15] M. Ertinger. *Nur das Geld zählt?: Erfassung von Nutzenorientierungen bei der Zuteilung von Geld und materiellen Gütern*. Forschungszentrum, Zentralbibliothek, 2003.

[16] N.V. Flor and E.L. Hutchins. Analyzing distributed cognition in software teams: A case study of team programming during perfective software maintenance. In *Empirical Studies of Programmers: Fourth Workshop*, pages 36–64. Intellect Books, 1991.

[17] B.G. Glaser and A.L. Strauss. *The discovery of grounded theory*. Aldine de Gruyter New York, 1967.

[18] M. Halbwachs. *On collective memory*. University of Chicago Press, 1992.

[19] J. Heckhausen and H. Heckhausen. *Motivation und Handeln*. Springer, 2006.

[20] F. Herzberg, B. Mausner, and B.B. Snyderman. *The motivation to work*. Transaction Publishers, 1993.

[21] J. Hollan, E. Hutchins, and D. Kirsh. Distributed cognition: toward a new foundation for human-computer interaction research. *ACM Transactions on Computer-Human Interaction (TOCHI)*, 7(2):174–196, 2000.

[22] P. Köppel. *Konflikte und Synergien in multikulturellen Teams: Virtuelle und face-to-face-kooperation*. Deutscher Universitätsvlg, 2007.

[23] U. Krengel. *Einführung in die Wahrscheinlichkeitstheorie und Statistik: Für Studium, Berufspraxis und Lehramt*. Vieweg, 2005.

[24] G.R. Lefrancois. *Psychologie des Lernens*. Springer, 1994.

[25] A.H. Maslow. A theory of human motivation. *Twentieth Century Psychology: Recent Developments in Psychology*, page 22, 1946.

[26] M. Minsky. *The society of mind*. Simon & Schuster, 1988.

[27] P. Naur and B. Randell. Software Engineering: Report of a conference sponsored by the NATO Science Committee (7–11 Oct. 1968), Garmisch, Germany. *Brussels, Scientific Affairs Division, NATO*, 1969.

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

[29] I.P. Pavlov and G.V. Anrep. *Conditioned reflexes: An investigation of the physiological activity of the cerebral cortex*. Dover New York, 1960.

[30] F.R. Pfetsch. *Konflikt und Konfliktbewältigung*. Klett, 1994.

[31] J. Radatz, editor. *IEEE standard glossary of software engineering terminology*. The Institute of Electrical and Electronics Ehgineers, NY, USA, 1990.

[32] T. Reenskaug. Models-views-controllers. *Technical note, Xerox PARC, December*, 1979.

[33] L. Rising. *The patterns handbook: Techniques, strategies, and applications*. Cambridge Univ Pr, 1998.

[34] W. Rollett. *Strategieeinsatz, erzeugte Information und Informationsnutzung bei der Exploration und Steuerung komplexer dynamischer Systeme*. LIT Verlag Berlin-Hamburg-Münster, 2008.

[35] S. Salinger, L. Plonka, and L. Prechelt. A coding scheme development methodology using grounded theory for qualitative analysis of pair programming. *THE PROBLEMS OF PROFESSIONALS*, page 9, 2008.

[36] S. Salinger and L. Prechelt. What happens during Pair Programming? In *Proceedings of the 20th Annual Workshop of the Psychology of Programming Interest Group (PPIG 08), Lancaster, England*, 2008.

[37] Stephan Salinger. Aspekte des Pair Programming. Ausgewählte empirische Untersuchungen. http://www.inf.fu-berlin.de/inst/ag-se/, 2004.

[38] W. Schönpflug and U. Schönpflug. *Psychologie: allgemeine Psychologie und ihre Verzweigungen in die Entwicklungs-, Persönlichkeits-und Sozialpsychologie: ein Lehrbuch für das Grundstudium*. Beltz, 1997.

[39] R.L. Solso. *Kognitive Psychologie*. Springer, 2006.

[40] C. Stangor. *Social groups in action and interaction*. Psychology Pr, 2004.

[41] A.L. Strauss and J.M. Corbin. *Basics of qualitative research: Techniques and procedures for developing grounded theory*. Sage Publications Inc, 1998.

[42] W. Stroebe and M. Reiss. *Sozialpsychologie*. Springer, 2003.

[43] F. Tschan. *Produktivität in Kleingruppen*. Verlag Hans Huber, 2000.

[44] L.S. Vygotsky. *Mind in society*. Harvard University Press Cambridge, MA, 1980.

[45] WAHRIG Deutsche Rechtschreibung. http://www.wissen.de/wde/generator/wissen/ressorts/bildung/woerterbuecher/index.html.

[46] J.B. Watson. *Behaviorism*. Transaction Pub, 1998.

[47] B. Weiner, R. Reisenzein, and W. Pranter. *Motivationspsychologie*. Beltz, Psychologie-Verl.-Union, 1994.

[48] Wörterbuch der deutschen Gegenwartssprache. http://www.dwds.de/woerterbuch.

[49] L. Williams, RR Kessler, W. Cunningham, and R. Jeffries. Strengthening the case for pair programming. *IEEE software*, 17(4):19–25, 2000.

[50] L.A. Williams and R.R. Kessler. All I really need to know about pair programming I learned in kindergarten. 2000.

[51] P.G. Zimbardo and R.J. Gerrig. *Psychologie, 18. aktualisierte Auflage*. München: Pearson Education Deutschland, 2008.