Masterarbeit am Institut für Informatik der Freien Universität Berlin,

Arbeitsgruppe Software Engineering

Qualitative Untersuchung von Orientierungsphasen während der Paarprogrammierung

Michael Pöhle
Matrikelnummer: 4492876
michael.poehle@fu-berlin.de

Betreuer: Franz Zieris
Gutachter: Prof. Dr. Lutz Prechelt
Zweitgutachter: Prof. Dr. Elfriede Fehr

Berlin, 29.09.2014

Zusammenfassung

Paarprogrammierung ist eine Softwareentwicklungspraktik bei der zwei Entwickler gleichzeitig an einem Computer arbeiten. Es gibt zum Nutzen der Paarprogrammierung einige quantitative Forschung mit teils widersprüchlichen Resultaten und wenig qualitative Forschung.

In dieser Arbeit wurde ein Teil dieser Praktik qualitativ untersucht, um sie besser zu verstehen und Paarprogrammierern zu helfen, sie effizienter durchzuführen. Dazu wurden Aufzeichnungen von Paarprogrammierungssitzungen ausgerichtet nach der *Grounded Theory Methodology* detailliert analysiert.

Der Fokus liegt auf Phasen während der Paarprogrammierung, in denen das Paar sich orientiert, in denen ihm noch Informationen fehlen oder Entscheidungen getroffen werden müssen, bevor es anfangen kann Softwareartefakte zu produzieren. Dazu werden Rollen vorgestellt, die Entwickler während der Phasen einnehmen, typische Aktivitäten beschrieben und aufgezeigt, welche Konsequenzen verschiedene Handlungsmuster in bestimmten Kontexten haben.

Zusammenfassend lässt sich sagen, dass es für eine flüssige, effiziente Paarprogrammierungssitzung zuträglich ist, wenn in den Orientierungsphasen strukturiert alle wichtigen Themen besprochen werden.

Eidesstattliche Erklärung

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

29.09.2014

Michael Pöhle

Inhaltsverzeichnis

1 Vorwort

1.1 Paarprogrammierung

Paarprogrammierung ist eine agile Entwicklungspraktik, bei der zwei Entwickler zusammen an einem Problem arbeiten. Dabei muss nicht unbedingt Code entstehen. Häufig wird über verschiedene Strategien zur Lösung eines Problems diskutiert, oder einfach günstige Arbeitsabläufe beredet.

> „Als Paarprogrammierung (PP) bezeichnet man den Prozess, bei dem zwei Personen an einem Computer gemeinsam am selben Design, Algorithmus, Code oder Test arbeiten.“ ([18, S. iv])

Bei der Paarprogrammierung nehmen die Partner unterschiedliche Rollen ein. Häufig wird dabei zwischen dem *driver* - er hat die Maus und Tastatur, und schreibt Code und dem *observer*, *pointer*, oder *navigator* [19] unterschieden - der den Code währenddessen überdenkt oder über Design, Strategie und mögliche Schwachstellen nachdenkt.

1.2 Motivation

Die agile Entwicklungsmethode *Extreme Programming* [3] führt große Teile ihres Erfolgs auf die Nutzung der Paarprogrammierung zurück. Die Berichte über den Erfolg mit dem Einsatz von Paarprogrammierung sind allerdings oft anekdotischer Natur. [20]
In den Berichten wird häufig behauptet, dass Paare Probleme schneller lösen, mit dem Prozess zufriedener sind und der resultierende Code deutlich defektfreier ist. Zwei Entwickler kosten aber auch mehr als einer. Daher stellt sich die Frage der Wirtschaftlichkeit. Dazu wurde seit den 1990er Jahren geforscht, allerdings mit uneindeutigen Ergebnissen. Die Ergebnisse zwischen den Studien variieren stark und die Wirtschaftlichkeit der Paarprogrammierung bleibt unklar - siehe Abschnitt 2.2.
Die meisten der Studien sind quantitativer Natur. Um den Prozess der Paarprogrammierung besser zu verstehen und die beobachteten Unterschiede erklären zu können, bedarf es qualitativer Forschung zur Paarprogrammierung. `Kent Beck` sagt in [3, S. 100] zur Paarprogrammierung „It's a subtle skill“. Forschung zur Paarprogrammierung kann den Forschern helfen diese Fähigkeit besser zu verstehen, und Paarprogrammierern, ihn besser zu gestalten.

1.3 Vorgehensweise, Fokus und Zielsetzung der Arbeit

Das Ziel der Arbeit ist die qualitative Forschung zur Paarprogrammierung voranzuschreiten. Da es hierbei hauptsächlich darum geht, zu untersuchen, wie Menschen miteinander handeln, kommt dabei eine typische informatische Methode nicht in Frage. Deshalb wurde als Forschungsmethode für diese

Arbeit die *Grounded Theory Methodolgy* ausgewählt. Mehr dazu in Sektion 3.1.
Innerhalb einer einzelnen Masterarbeit ist es nicht möglich eine umfassende Theorie für den gesamten Prozess der Paarprogrammierung zu entwickeln. So schlägt Stephan Salinger in [11] vor, mithilfe der in seiner Dissertation beschriebenen Basiskonzepte weitere Analysen als Schichten darauf aufzubauen.
Der Fokus dieser Arbeit gilt Orientierungsphasen während der Paarprogrammierung. Orientierungsphasen sind, grob gesagt, die Phasen, in denen das Paar die nötigen Informationen sammelt, um ein vorliegendes Problem lösen zu können. Näheres wird in Kapitel 4.1 beschrieben.

1.4 Aufbau dieser Arbeit

Einleitung

In der Einleitung wird das Themengebiet der Arbeit vorgestellt und der Stand der Forschung umrissen. Außerdem werden Grundlagen für das Verständnis der Arbeitsergebnisse gelegt und beleuchtet mit welchen Daten und Methoden zu den Ergebnissen gelangt wurde.

Hauptteil

Im Hauptteil werden Orientierungsphasen in der Paarprogrammierung genau beleuchtet und auftretende Phänomene vorgestellt.

Schluss

Schließlich werden die Ergebnisse der Arbeit beurteilt und eingeordnet.

Teil I
Einführung

2 Stand der Forschung

2.1 Vorteile der Paarprogrammierung

Es gibt einige anekdotische Berichte über die Vorteile der Paarprogrammierung, häufig genannt werden unter anderen folgende Phänomene:

1. Qualität: Paarprogrammierer erstellen bessere Designs für ihre Lösungen, besseren Code [7, Kap. 12] und machen weniger Fehler [4].
2. Zufriedenheit: Viele Programmierer arbeiten lieber im Paar als allein. Außerdem sind die Paarprogrammierer zuversichtlicher, dass ihr Code gut funktioniert [21].
3. Wissensvermittlung: Während einer Paarprogrammierungssitzung kommuniziert das Paar stark miteinander. Dadurch wird Wissen über Lösungsansätze, benutzte Tools oder bestehenden Code ausgetauscht [16].

Der Partner beim Paarprogrammieren wirkt wie ständiger Codereviewer, wodurch Fehlerraten gering bleiben [8]. Außerdem denken zwei Entwickler über mehr Lösungsmöglichkeiten nach als ein einzelner, was die Wahrscheinlichkeit erhöht, dass eine gute gefunden wird [20].

2.2 Forschung zur Paarprogrammierung

Nach den Erfolgsbeschreibungen zur Paarprogrammierung, mit dem *Chrysler Comprehensive Compensation System* als einem der bekannteren Projekte, unter anderem geleitet von `Kent Beck` [3], bestand der Wunsch, die Behauptungen fundiert nachweisen zu können.

2.3 Quantitative Forschung in der Paarprogrammierung

Viele der ersten Studien haben versucht verschiedene vermutete Dimensionen von Softwareentwicklungsprozessen genauer zu untersuchen. So gab es einige Studien, die versuchten Aspekte wie Wirtschaftlichkeit, Defektrate und Zufriedenheit bei der Paarprogrammierung zu quantifizieren. [20]

Die Resultate variierten oft stark zwischen den Studien, in einigen Studien lösen Paare ein Problem 40% schneller [9], in anderen sind sie 29% langsamer [17]. Berichte zu Defektraten schwanken zwischen 15% weniger [4] und 200% mehr [17].

In einer Meta-Studie aus dem Jahr 2009 vermuten die Autoren, dass einige der Forschungsergebnisse unter Publikationsbias leiden und weisen darauf hin, dass die Unterschiede in der beobachteten Codequalität und Arbeitszeit abhängig von der Komplexität des zu lösenden Problems sind. [6].

2.4 Qualitative Forschung in der Paarprogrammierung

Es gibt nur wenig qualitative Forschung zur Paarprogrammierung. Viele der qualitativen Ergebnisse wurden in eher quantitativen Studien gefunden.

- Paare haben mehr Vertrauen in ihre Losung und mehr Spaß [21]
- die Paarleistung hängt nicht alleine vom Leistungsniveau der Individuen ab [22]
- Paare erzielen besser Lernerfolge [16]

Als Grundlage für diese Arbeit dienen hauptsächlich [12], [11] und [13]. Insbesondere beschreibt `Stephan Salinger` in seiner Dissertation Konzepte für den gesamten Prozess der Paarprogrammierung und schlägt vor, wie weitere Forschung darauf aufbauen kann.

3 Forschungsmethode

Der Prozess der Paarprogrammierung ist nicht sehr gut verstanden. Daher besteht der Wunsch nach qualitativer Forschung zur Paarprogrammierung. Um eine Theorie über den Prozess zu entwickeln, bedarf es seiner Analyse. Als Basis dafür dienten für diese Arbeit Aufzeichnungen von Paarprogrammierungssitzungen. Näheres zu diesen Aufzeichnungen wird in Abschnitt 3.2 beschrieben.
Als Methode für die Entwicklung einer solchen Theorie ausgehend von Daten bietet sich die *Grounded Theory Methodology* an. Wie diese theoretisch funktioniert und tatsächlich angewandt wurde, wird in Sektion 3.1 näher beleuchtet.
Außerdem wird in Abschnitt 3.3 ein grober Überblick über die Basisschicht geliefert und einige Basiskonzepte vorgestellt.

3.1 Forschungsprozess nach der Grounded Theory Methodology

Die *Grounded Theory Methodology* ist ursprünglich eine Methodologie aus der Sozialforschung. Es ist ein Verfahren, bei dem Daten analysiert und auftretende Phänomene benannt werden. Die Benennungen helfen bei der Bildung theoretischer Konzepte und beim Vergleich zwischen Phänomenen

und ihren Kontexten. Die gebildeten theoretischen Konzepte können dann verglichen, in Beziehung gesetzt und kategorisiert werden [1].
Es gibt zwei Arten der *Grounded Theory Methodology*:

1. `Glaser` bevorzugt ein Verfahren, das sehr stark darauf ausgerichtet ist, Daten unvoreingenommen zu analysieren und Muster zu erkennen; äußere Einflüsse wie zum Beispiel Literatur sollen möglichst ignoriert werden [5]

2. `Strauss & Corbin` befürworten einen Ansatz, der besser für wissenschaftliche Überprüfbarkeit ausgelegt ist und ein systematischeres Verfahren anstrebt. [15]

Für diese Arbeit wurde die *Grounded Theory Methodology* nach `Strauss & Corbin` verwendet. Sie macht keine Annahmen über die Datengrundlage und liefert belastbare, datengestützte Resultate.
Im Folgenden wird auf die von `Strauss & Corbin` beschriebenen Phasen der *Grounded Theory Methodology* genauer eingegangen.

Offenes Kodieren

Beim offenen Kodieren geht es darum, die vorliegenden Daten sehr feingranular zu konzeptualisieren, das heißt auftretende Phänomene zu benennen [15, S. 45]. Anfangs wird alles kodiert, um ein Gefühl dafür zu bekommen, wo die wichtigen Aspekte der Daten liegen. Die gefundenen Konzepte werden mit neuen verglichen, sobald man sie findet, so erkennt man wichtige Gemeinsamkeiten und Unterschiede und kann Konzepte gegebenenfalls zusammenfassen, ändern oder kategorisieren.
Verschiedene Konzepte lassen sich anhand ihrer Eigenschaften unterscheiden, das heißt, sie haben innerhalb verschiedener Dimensionen verschiedene Ausprägungen. Es bietet sich an, gerade Konzepte zu Kategorien zusammenzufassen, die sich nur in wenigen Dimensionen stark unterscheiden. [15, S. 50f]

Axiales Kodieren

Während des axialen Kodierens werden gefundene Konzepte zu einander in Verbindung gebracht. `Corbin & Strauss` schlagen vor, dazu ein paradigmatisches Modell zu nutzen, in dem *Phänomene* aufgrund *ursächlicher Bedingungen* und in einem bestimmten *Kontext* auftreten und *Konsequenzen* haben. Dabei gibt es *intervenierende Bedingungen*, welche die möglichen resultierende Phänomene einschränken [15, S. 78].

Selektives Kodieren

Nachdem die Kernkategorie - diejenige Kategorie, mit deren Konzepten sich das Verhalten der Untersuchten am besten beschreiben lässt - gefunden wur-

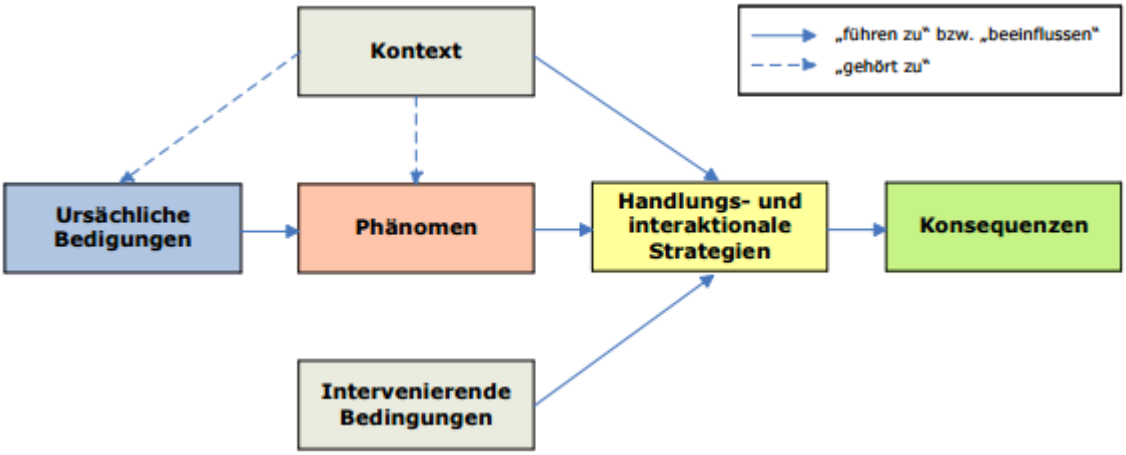

Abbildung 1: Das paradigmatische Modell. Abbildung aus [11, Abb. 3.3].

de, werden die Daten nur noch selektiv kodiert. Phänomene, die hinsichtlich des Kerns unwichtig sind, können dabei ignoriert werden. Das Ziel ist, mit größerer Genauigkeit die wichtigen Zusammenhänge und Phänomene auf einer abstrakteren Ebene in Beziehung zu setzen. Das hilft eine konsistente Theorie über die Daten zu entwickeln und zu den Daten „eine Geschichte zu erzählen". In dieser Geschichte geht es dann hauptsächlich um Konzepte in der Kernkategorie, weitere Kategorien helfen die Konzepte in der Kernkategorie in Kontext zu setzen. [15, S. 94f]

Theoretisches Sampling

Beim theoretischen Sampling geht es darum, gezielt Daten zur Untersuchung zu wählen, die helfen eine bestimmte Kategorie von Konzepten besser zu verstehen. Gerade in der Phase des selektiven Kodierens sollte theoretisches Sampling genutzt werden, um keine unnötigen Daten zu kodieren. Das Ziel ist nicht eine repräsentative Auswahl an Daten zu untersuchen, sondern eine Auswahl, die beim Verständnis der zu untersuchenden Vorgänge hilft. [15, S. 148]

3.2 Datenbasis

Für diese Arbeit dienten als Gegenstand der Untersuchungen umfangreiche Audio- und Videoaufzeichnungen von Paarprogrammierungssitzungen. In den meisten Aufzeichnungen lösen professionelle Entwickler ein konkretes Problem im Paar.

Art des Materials

Die untersuchten Aufzeichnungen stammen aus der *Arbeitsgruppe Software Engineering* am Fachbereich Informatik der Freien Universität Berlin. Die Aufzeichnungen bestehen aus:

1. **Bildschirmaufnahme**: Der Großteil eines Videos ist eine hochauflösende Bildschirmaufnahme der Entwickler.
2. **Webcamaufnahme**: Zusätzlich ist in der Bildschirmaufnahme ein Webcam-Video der Entwickler eingebettet. Die Webcamaufnahmen haben eine Auflösung von 320x240 Pixeln.
3. **Audio**: Zu den Bildern wurden die Gespräche der Entwickler und wenn möglich andere Töne (z.B. Tastenanschläge) aufgenommen.

Ein Screenshot aus einer Aufnahme kann im Anhang gefunden werden (Abbildung 5).
In den untersuchten Aufzeichnungen lösen professionelle Entwickler, die regelmäßig in Paaren programmieren, konkrete Aufgaben in ihrem gewohnten Umfeld. Genaueres dazu, wie die Aufzeichnungen entstanden sind, beschreibt `Laura Plonka` in [10, Kap. 3].

3.3 Basisschicht

In [13] schlagen `Salinger & Prechelt` einen Ansatz für die Paarprogrammierungsforschung vor, der darauf abzielt, die Forschung in Schichten zu modularisieren. Die verschiedenen Schichten bauen dabei auf einander oder auf die in [11] beschriebenen *Basiskonzepte* auf. Die Konzepte verschiedener Schichten können sich stark in Detailgrad, Tragweite und Perspektive unterscheiden.

Basiskonzepte

Die Basiskonzepte beschreiben die primären Intentionen einzelner Aussagen und Handlungen von Entwicklern. Um darauf aufbauende Forschung zu ermöglichen, sind die Basiskonzepte so flexibel und generisch wie möglich. [13]

Aufbau der Basisschicht

Die Basisschicht unterscheidet auf der höchsten Ebene *HHI-Konzepte* (von englisch: human-human interaction) und *HCI/HEI-Konzepte* (von englisch: human-computer interaction; human-environment interaction).

Konzept	**Äußerung**	**Kommentar**
ask_knowledge	Was passiert, wenn das hier nicht gesetzt ist?	Die Intention der Aussage ist, vom Partner Wissen zu erhalten.
propose_design	Und wie nennen wir das Ding? `id`?	Es wird vorgeschlagen eine bestimmte Komponente `id` zu nennen.
agree_design	Ummmmm.	Ein Vorschlag wird angenommen. Hier sieht man, dass die Intention eine Äußerung stark von der Intonation oder dem Kontext abhängen können.

Tabelle 1: Beispiele von Basiskonzepten, aus [13]

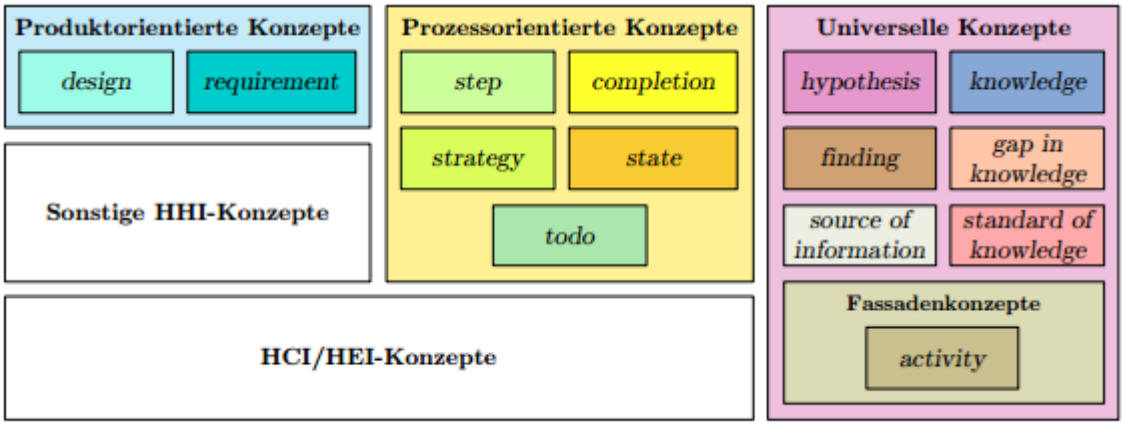

Abbildung 2: Übersicht über die Basiskonzepte. Abbildung aus [23, Abb. 2.3].

HCI-Konzepte

Bei den HCI-Konzepten handelt es sich um Konzepte, die die Interaktion zwischen den Entwicklern beschreiben. Im Folgenden werden die Konzeptkategorien der HCI-Konzepte kurz charakterisiert und Beispiele gezeigt. Die Beschreibungen und Beispiele stammen alle aus [13].

- **Produkt-orientierte Konzepte** handeln von Vorschlägen hinsichtlich Details des Programms. Dabei reichen sie von groben Produktanforderungen zu einzelnen Namensgebungen. Die Produkt-orientierten Konzepte umfassen die Klassen *design* - Gestaltungsmöglichkeiten des Produkts - und *requirement* - Anforderungen an das Produkt.

Konzept	**Äußerung**	**Kommentar**
propose_design	Ja, das muss weg.	Der Sprecher schlägt vor, einen Methodenaufruf zu entfernen.
propose_requirement	Ok, lass uns davon ausgehen, dass sie eine Stunde unterschied haben	Der Sprecher schlägt vor, einen Annahme für das Produkt zu machen, die dabei helfen kann das aktuelle Problem zu lösen.

Tabelle 2: Beispiele von Produkt-orientierten Konzepten

- **Prozess-orientierte Konzepte** beschreiben Aktivitäten und Äußerungen über den Arbeitsprozess des Paares. Dazu gehören Konzepte der Klassen *step*, *strategy*, *completion*, *state* und *todo*.

  Ein *step* beschreibt einen Handlungsschritt, eine *strategy* mehrere Schritte die zu einem Ziel führen. Ein *todo* ist ein Handlungsschritt, der in der Zukunft ausgeführt werden soll.

  Bei *completion* geht es um den Status eines *step*, während *state* vom Stand einer *strategy* handelt.

- **Universelle Konzepte** Bei den universellen Konzepten geht es um Äußerungen, die mit Wissen zu tun haben - Aufforderungen zum Wissenstransfer, zum aktuellen Wissenstand oder das Formulieren neuer Hypothesen und Ideen. Die universellen Konzepte können sowohl in Bezug auf das Produkt als auch den Prozess auftreten.

- **Fassadenkonzepte** Die Fassadenkonzepte sind eine Untermenge der universellen Konzepte. Sie bieten einen Kontext, in dem komplexere Vorgänge aufgeteilt und die Teile durch andere Konzepte beschrieben

Konzept	**Äußerung**	**Kommentar**
propose_strategy	Dann sollten wir das vorher oder nachher machen. Aber nicht gleichzeitig	Von drei groben Handlungsstrategien wird eine als schlecht abgelehnt.
explain_state	Wegen einchecken und so, Ich denke wir sind an einem Punkt, an dem wir eine Pause machen können	Eine Einschätzung zum Stand der Arbeit.

Tabelle 3: Beispiele von Prozess-orientierten Konzepten

Konzept	**Äußerung**	**Kommentar**
explain_standard of knowledge	Da bin ich mir selbst aber auch nicht sicher.	Der Sprecher betont, dass er seinem eigenen Urteil nicht traut.
explain_finding	Die get *(..) (fehlt)*	Dem Sprecher ist gerade aufgefallen, dass eine *getter* - Methode fehlt und macht seinen Partner darauf aufmerksam.

Tabelle 4: Beispiele von universellen Konzepten

werden können. In der Basisschicht wird nur eine Fassadenkonzeptklasse beschrieben, bei der es um die Verbalisierung und Einschätzung von Aktivitäten zwischen dem Paar und dem Computer oder ihrer Umwelt geht. Ein Beispiel für ein Fassadenkonzept ist *think aloud_activity* - z.B.: ein Entwickler schreibt etwas und redet gleichzeitig darüber, möglicherweise um den Partner auf dem Laufenden zu halten.

HCI/HEI Konzepte

Die HCI und HEI Konzepte handeln von Interaktionen der Entwickler mit dem Computer oder ihrer Umgebung. Die in der Basisschicht ([13]) beschriebenen Interaktionen sind:

1. *write*: Programmcode (oder andere Artefakte) schreiben oder ändern
2. *search*: die Suche nach einem bestimmten Zielobjekt
3. *explore*: Erkundung von vorhandenen Artefakten - im Unterschied zur Suche ist noch nicht klar, was genau gefunden werden soll
4. *verify*: Überprüfung von Änderungen an Artefakten
5. *read*: Informationen laut lesen
6. *sketch*: eine graphische Skizze anfertigen
7. *show*: auf etwas zeigen
8. *do*: alles andere

In dieser Arbeit wurden Phasen der Paarprogrammierung untersucht, für die hauptsächlich HHI-Aktivitäten interessant sind.

3.4 Forschungsstrategische Aspekte dieser Arbeit

Verwendung der Basisschicht

Die Basisschicht wurde intensiv während des Einstiegs in die Materie genutzt. Sie half die vorliegenden Daten sinnvoll zu annotieren und zu verstehen und schärfte die theoretische Sensibilität.
Es fällt aber schnell auf, dass der Blickwinkel der Basisschicht ein anderer ist, als sinnvoll für die Betrachtung von Orientierungsphasen in der Paarprogrammierung. Das ist nicht überraschend, die Basiskonzepte dienen dennoch oft als Grundlage für die Beschreibung wichtiger Phänomene während der Orientierungsphasen.

Theoretisches Sampling

Theoretisches Sampling wurde in 3.1 beschrieben. Gewöhnlich werden für das theoretische Sampling neue Daten erhoben. Das war allerdings in diesem Fall wenig praktikabel. Außerdem existieren in der *Arbeitsgruppe Software Engineering* am Fachbereich Informatik der Freien Universität Berlin bereits mehrere Tage Aufzeichnungen von Paarprogrammierungssitzungen, sodass das theoretische Sampling darauf hinauslief, geeignete Aufzeichnungen aus diesen auszuwählen.
Insgesamt wurden für die Arbeit 15 Sitzungen untersucht, die in möglichst unterschiedlichen Kontexten stattfanden.

Teil II
Hauptteil

In diesem Teil werden die Ergebnisse der Arbeit präsentiert. Zunächst wird aber der Fokus der Arbeit genauer beschrieben und der Begriff *Orientierungsphase* besser eingegrenzt. Außerdem fiel während der Untersuchungen auf, dass die Partner während einer Paarprogrammierungssitzung je nach Kontext verschiedenen Rollen einnehmen, sodass ich vorher noch das Konzept der Rolle in der Paarprogrammierung näher beleuchte. Die wichtigsten Unterschiede zwischen den Kontexten der Sitzungen werden auch vorgestellt.

4 Orientierungsphasen

4.1 Fokus der Arbeit

Es war anfangs geplant während der Arbeit die Anfänge von Paarprogrammierungssitzungen zu untersuchen. Ich hatte vermutet, dass die Qualität der Anfänge einen recht großen Einfluss auf die Qualität der restlichen Sitzung hat, oder dass man zumindest schon in den Anfangsphasen Muster erkennen kann, die auf mögliche Probleme hinweisen. Es wäre damit also hilfreich für die Sitzung als Ganzes, Probleme am Anfang zu erkennen und zu vermeiden. Zu Beginn habe ich mir deshalb die ersten 20 Minuten verschiedener Aufzeichnungen angeschaut. Eine Zeitspanne ist allerdings ein eher schlechtes Mittel, um 'Anfang' zu definieren. In vielen Aufzeichnungen passieren sehr unterschiedliche Dinge in den ersten 20 Minuten.

- In der Sitzung KA1 sammelt Informationen zu seinem Problem und dem System, an dem es arbeitet. Es redet dazu teilweise mit einer dritten Person. Nach 20 Minuten ist es damit noch nicht fertig.
- In der Sitzung KA8 ist in den ersten 20 Minuten nur höchstens einer der Partner am Rechner.
- In der Sitzung CA1 orientiert sich das Paar ca. 15 Minuten lang und geht dann langsam dazu über, Programmcode zu schreiben oder zu editieren.
- In der Sitzung LA1 editiert das Paar schon nach 3 Minuten Programmcode.

Ich habe daher begonnen Anfänge zu definieren als: „die Zeit bei dem sich mindestens ein Partner noch nicht klar ist, wie das Sitzungsziel zu erreichen ist“. Das Sitzungsziel kann dabei von *Implementieren des Features X* über *Testen der Funktionalität des Systems* zu *Anforderungen ins Wiki schreiben*

reichen. Die Definition schien mir nützlich, um das, was ich intuitiv als 'Anfang' charakterisiert hatte, abzugrenzen. In diese Definition passen allerdings auch teilweise Episoden einer Sitzung, die gar nicht am Anfang stattfinden. *Orientierungsphase* ist daher ein besserer Name für solche Episoden.
Bei der Abgrenzung ist nicht relevant, ob objektiv alles Nötige zum Erreichen des Sitzungsziels geklärt ist. Sobald den Entwicklern alles klar scheint, beenden sie in der Regel die Orientierungsphase.

4.2 Sinn der Orientierungsphase

In nahezu allen untersuchten Sitzungen treten Orientierungsphasen auf. Während der Orientierungsphasen versucht das Paar ein besseres Verständnis über sein Sitzungsziel, potentiell vorhandene Systeme und Anforderungen aufzubauen.
Wie erwähnt, haben die Partner während der Orientierungsphase subjektiv noch kein ausreichendes Verständnis der Problemdomäne. Sobald die Orientierungsphase beendet wird, glaubt zumindest einer der Partner, alles Nötige verstanden zu haben. Ist das nicht der Fall, fällt das oft später in der Sitzung auf - Missverständnisse werden möglicherweise erst später in der Sitzung erkannt und geklärt; ein nicht ausreichendes Verständnis eines wichtigen Aspekts muss später beseitigt werden.
Je nach Relevanz der Wissenslücken und Missverständnisse, haben sie entweder kaum sichtbaren Einfluss auf den Rest der Sitzung oder es müssen aktuelle Arbeitsflüsse unterbrochen werden, damit das Paar sich erneut orientieren kann.
In vielen Situationen kann man als Paarprogrammierer davon ausgehen, dass der Partner bestimmtes Wissen hat, ohne dies sichergestellt zu haben - d.h entweder man hat das Wissen selbst transferiert, oder gefragt, ob das Wissen vorhanden ist. Dadurch kann man während der Orientierungsphase Zeit sparen. Ist die Annahme allerdings falsch, verliert man möglicherweise später in der Sitzung mehr Zeit, als gespart wurde. Die aktuelle Tätigkeit muss unterbrochen werden, um Teile der Orientierung nachzuholen. Anschließend muss man sich wieder kurz eindenken, bevor die Tätigkeit fortgesetzt wird.

4.3 Kontexte

Die untersuchten Paarprogrammierungssitzungen finden in verschiedenen Kontexten statt. Was genau während einer Orientierungsphase besprochen wird und besprochen werden kann, kann stark von diesem Kontext abhängen. Gibt es beispielsweise keine bestehende Software, wird sich das Paar nicht über deren aktuellen Stand unterhalten können.
Im Folgenden wird kurz ein Überblick über die wichtigsten Unterschiede zwischen den untersuchten Kontexten gegeben.

1. **Es gibt kein System, auf das aufgebaut wird:** In den meisten Fällen existiert ein System, das irgendwie verändert oder erweitert werden soll. Ich habe einige Aufzeichnungen untersucht, bei denen ein Stück Software von Grund auf erstellt wurde. Dieses waren Aufzeichnungen von Studenten, die im Paar versuchten ein ihn gestelltes Problem zu lösen.

   Am auffälligsten ist, dass in den anderen Sitzungen das Verständnis der vorhanden Software oft viel Zeit während der Orientierungsphase beansprucht. Das fällt hier im Allgemeinen weg.

2. **Es gibt ein System, beiden Partner fehlen wichtige Informationen:** Wenn schon Software vorhanden ist, ist es gewöhnlich nötig den Teil, der erweitert oder verändert werden soll, gut zu verstehen. Erst dann können sinnvolle Pläne erstellt werden, um das Sitzungsziel zu erreichen. In den untersuchten Sitzungen zu diesem Kontext bestanden große Teile der Orientierungsphasen daraus, vorhandenen Code zu sichten, zu verstehen und Vermutung darüber anzustellen, welche Informationen noch nötig sind, oder wie mit den neu gewonnenen Informationen das Sitzungsziel erreicht werden könnte.

3. **Es gibt ein System, ein Partner hat alle wichtigen Informationen:** Dies ist der Kontext in dem die meisten beobachteten Sitzungen stattfanden. Wie in Abschnitt 2.1 beschrieben, ist Paarprogrammieren eine gute Methode, um Wissen im Team zu verteilen. Hier weiß ein Partner etwas, was der andere nicht weiß und somit entsteht eine Möglichkeit, das Wissen zu teilen.

4. **Es gibt ein System, beide Partner haben die wichtigen Informationen:** Wenn beide Partner alle wichtigen Informationen haben, gibt es theoretisch keinen Grund noch darüber zu reden. Die Partner können sich allerdings in der Regel nicht sicher sein, dass ihr Gegenüber mindestens den gleichen Wissenstand hat oder Informationen so interpretiert wie man selbst. Daher gibt es auch in diesem Kontext Konversationen zum Wissensabgleich, oder zumindest um Gewissheit zu erlangen, dass alle Informationen bei beiden Partnern vorhanden sind und verstanden wurden.

Eine Übersicht über die in dieser Arbeit untersuchten Sitzungen und deren Kontext ist in Tabelle V zu finden.

Sitzung	**Kontext**	**Kommentar**
CA1	Es gibt ein System, ein Partner hat alle wichtigen Informationen	
CA2	Es gibt ein System, ein Partner hat alle wichtigen Informationen	
CA3	Es gibt ein System, beiden Partner fehlen wichtige Informationen	
CA4	Es gibt ein System, ein Partner hat alle wichtigen Informationen	Es ist schon recht viel Zeit vergangen, seitdem der Partner mit den Informationen am Code gearbeitet hat, sodass einige wichtige Informationen bei beiden fehlen
CB1	Es gibt ein System, ein Partner hat alle wichtigen Informationen	
JA1-3	Es gibt ein System, ein Partner hat alle wichtigen Informationen	Und der andere Partner kennt das System, an dem gearbeitet wird gar nicht.
JA4-5	Es gibt ein System, beide Partner haben die wichtigen Informationen	In den JA-Sitzungen versuchen die gleichen Entwickler ein Ziel zu erreichen. In den ersten drei Sitzungen wurden die Grundlagen gelegt, sodass ab der vierten beide Partner eine gute Vorstellung vom System haben
DA2	Es gibt ein System, ein Partner hat alle wichtigen Informationen	Und der andere Partner kennt das System, an dem gearbeitet wird gar nicht.
KA3&7	Es gibt ein System, ein Partner hat alle wichtigen Informationen	
LB1	Es gibt ein System, beiden Partner fehlen wichtige Informationen	
ZB2	Es gibt kein System, auf das aufgebaut wird	Das Paar besteht hier aus zwei Studenten

Tabelle 5: Übersicht über die untersuchten Sitzungen und deren Kontext

5 Phasen und Rollen

5.1 Rollen während der Paarprogrammierung

In der Literatur wird beim Paarprogrammieren häufig zwischen dem *driver* und dem *observer*, oder analogen Bezeichnungen unterschieden. Dabei ist der *driver* derjenige, der die Tastatur und Maus bedient und Code schreibt, während der Observer das Geschriebene versucht nachzuvollziehen und zu verbessern (siehe Kapitel 1.1).
Gerade in Hinsicht auf Orientierungsphasen in der Paarprogrammierung ist diese Unterteilung nicht zutreffend oder zumindest wenig hilfreich. Während der Orientierungsphasen schreibt häufig keiner der Partner Code. Die Orientierungsphasen sind größtenteils von Konversationen zwischen den Partnern geprägt.

In [14] bemerken `Salinger, Zieris & Prechelt`, dass in der Paarprogrammierung als Ganzes die Rollenverteilung in *driver* und *observer* unrealistisch ist und beschreiben, dass ein realistischeres Modell mehr als zwei Rollen beinhaltet, die fließend von Entwicklern eingenommen und abgelegt werden. Zusätzlich können Entwickler mehrere Rollen gleichzeitig ausführen.
Des Weiteren beschreiben sie ein Metamodell für ein neues Rollenkonzept. In diesem Modell bestehen Rollen aus verschiedenen Facetten. Die Facetten helfen anhand des beobachteten Verhaltens eines Entwicklers Rückschlüsse auf dessen Rolle zu ziehen. Dieser Zusammenhang und der Einfluss weiterer Aspekte wird in Abbildung 3 veranschaulicht.
Im untersuchten Material sind verschiedene Verhaltensmuster zu erkennen, die auf ein einheitliches Ziel schließen lassen. Diese Muster weisen auf verschiedene Facetten der von den Entwicklern eingenommenen Rollen hin. Im Folgenden werden die in den Orientierungsphasen auftauchenden Rollen beschrieben. Dabei halten sich die Beschreibungen an ein einheitliches Muster:

- **Kurzbeschreibung**: eine kurze Beschreibung der Rolle und der Umstände, in denen sie auftaucht
- **Aktivitäten**: eine Beschreibung der für die Rolle markanten Aktivitäten
- **Ziel**: Den Aktivitäten kann eine einheitliche Intention zugeschrieben werden. Hier geht es um die Interpretation des beobachteten Verhaltens. Dazu werden verschiedene Facetten der Rolle beschrieben.
- **Beispiele**, die das beschriebene Verhalten unterstreichen

5.1.1 Rolle: Experte

In [14] werden drei Rollen kurz beschrieben. Eine davon ist der *task expert*.

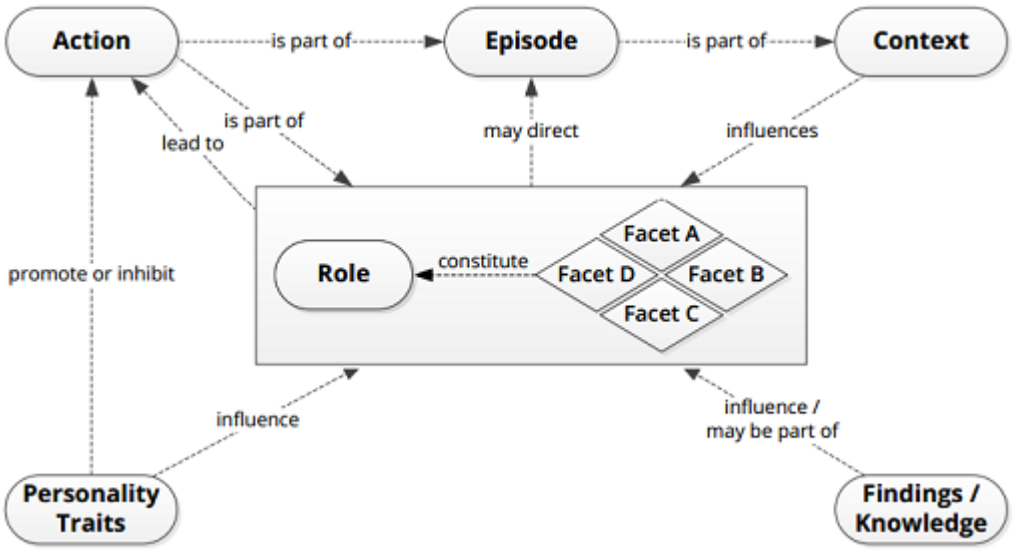

Abbildung 3: Metamodell zu Rollen und dazugehörigen Phänomenen während der Paarprogrammierung. Abbildung aus [14, Abb. 1].

In den meisten untersuchten Sitzungen gibt es ein vorhandenes System, an dem gearbeitet werden soll. Beide Partner kennen das System, aber einer hat kurz zuvor genau an dem Teil des Systems gearbeitet, an dem in der Sitzung gearbeitet werden soll. Er kennt sich daher deutlich besser mit diesem Teil aus als sein Partner und ist daher während der Orientierungsphase der *Experte*. Selbst wenn beide Partner die relevanten Teile des Systems ähnlich gut kennen, kommt es vor, dass einer die Rolle des *Experten* einnimmt.
Der *Experte* muss entscheiden können, was wichtig ist.

Aktivitäten des Experten

Der *Experte* hat mehr relevantes Wissen als sein Partner und versucht dieses Wissen mit ihm zu teilen, er erklärt seinem Partner Neues und beantwortet dessen Fragen. Typisch für den *Experten* sind daher Äußerungen der Art *explain_knowledge*. Häufig wird der Experte vorher versuchen den Wissenstand seines Partners einzuschätzen, um unnötigen Informationsaustausch zu vermeiden, oder um sicherzustellen, dass sein Partner alles nötige weiß (*ask_state_of_knowledge*).
In vielen untersuchten Sitzungen hat sich der Experte außerdem schon darüber Gedanken gemacht, wie man das Sitzungsziel erreichen kann und ist deshalb sehr schnell in der Lage, vollständige Designvorschläge für die Lösung zu präsentieren und zu diskutieren.

Ziel des Experten

Wie in [14] beschrieben hat die Rolle des *Experten* zwei Facetten:

1. **Wissen weitergeben**: Der *Experte* hat Wissen über bestimmte Softwareartefakte, welches sein Partner nicht hat und relevant für das Problem der aktuellen Sitzung ist.

2. **aus Wissen Vorschläge generieren**: In vielen untersuchten Sitzungen hat sich der Experte außerdem schon darüber Gedanken gemacht, wie man Sitzungsziel erreichen kann und ist deshalb sehr schnell in der Lage vollständige Designvorschläge für die Lösung zu präsentieren und zu diskutieren.

Zusammenfassend hat der Experte also zum Ziel effizient sicherzustellen, dass sein Partner alles wichtige weiß um ein gegebenes Problem zu lösen.

Wenn der *Experte* versucht den Wissenstand seines Partner zu erfahren, liegt wie erwähnt nahe, dass er das tut, um den Prozess effizient zu gestalten. Er vermeidet so Dinge zu erklären, die sein Partner schon verstanden hat. Äußerungen der Art *ask_state_of_knowledge* könnten ausschließlich ihren offensichtlichen Zweck zur Intention haben - die Einschätzung des Wissensstands des Partners. Wenn in den untersuchten Sitzungen durch eine solche Äußerung eine Wissenslücke offenbart wurde, folgte aber auch immer Wissenstransfer durch den *Experten*.

Beispiele

- Zu Beginn der Sitzung CA1 erklärt einer der Entwickler - der *Experte* P1, dass er direkt zuvor schon an dem Problem gearbeitet hat und will seinem Partner P2 erklären, was er gerade Neues implementiert hat.

  > *P1*: „Also, wie gesagt, ich habe schon angefangen ungefähr 'ne Stunde programmiert“

  Daraufhin unterbricht ihn P2 mit einem komplett anderen Thema, ungefähr 4 Minuten später greift P1 das ursprüngliche Thema wieder auf und beginnt seinem Partner die neue Implementierung zu erklären.

  > *P1*: „Ähm, also die GUI, die GUI wird erstellt im `FeatureLabelBasicOptionsPanel`“

- In der Sitzung KA3 will das Team vorhandenen Code refaktorisieren. Der *Experte* P1 weist seinen Partner deshalb auf eine kürzliche Änderung am System hin.

> *P1*: „Also einmal, das der Controller, da wo hab ich jetzt das Datumfeld in zwei Datumfelder, son Datum-Zeitfeld gesplittet“

5.1.2 Rolle: Schüler

Der *Schüler* ist meist der Partner des *Experten*. Ihm fehlen noch wichtige Informationen, um ein gegebenes Problem lösen zu können. Es kann sein, dass ihm diese Informationen leicht verständlich präsentiert werden, oder dass er selbst danach suchen und sie verstehen muss. Wenn keiner der Partner das vorliegende System ausreichend kennt, nehmen beide die Rolle des *Schülers* ein, und versuchen die ihnen wichtigen Informationen aus vorhandenem Programmcode, Dokumentation oder Ähnlichem zu erhalten.

Aktivitäten des Schülers

In vielen der untersuchten Sitzungen hört der *Schüler* dem *Experten* aufmerksam zu und fragt gelegentlich nach, wenn er etwas nicht versteht oder weitere Informationen zu einem Thema haben will. In Sitzungen ohne *Experten*, sichten die Entwickler Code und überlegen gemeinsam, welche Informationen sie noch benötigen und gleichen Verständnis ab.
Am markantesten für den *Schüler* sind *ask_knowledge*-Äußerungen. Während sie neues Wissen aufnehmen, kann auch beobachtet werden, dass die *Schüler* Hypothesen über die Funktionsweise ihnen noch unbekannter Dinge aufstellen und versuchen diese zu überprüfen.

Ziel des Schülers

Der *Schüler* verfolgt ein analoges Ziel zum *Experten*. Er will alle Informationen sammeln und verstehen, die nötig sind um das Sitzungsziel zu erreichen. Dazu ist es oft nötig, dass er eigene Wissenslücken erkennen und beseitigen kann.
In einigen Situationen, in denen Entwickler Fragen zum Erklärten stellen könnten die Fragen eher aus Höflichkeit oder als Zeichen der Aufmerksamkeit gestellt werden, ohne dass dem Fragenden die Antwort sonderlich interessiert. Meistens ist es aber naheliegender anzunehmen, dass die Frage zum Zwecke der Wissensgewinnung gestellt wird.
Gerade in Sitzungen, in denen beide Partner wenig vom vorliegenden System wissen, kann man oft beobachten, dass die Entwickler Code studieren und nach bestimmten hilfreichen Methoden und Klassen durchsuchen. Das Codestudium hat entweder einen Wissens- oder Verständnisgewinn zum Ziel oder dient der Überprüfung vorhandener Annahmen und Hypothesen.
Während des Codestudiums verbalisieren die Entwickler häufig neue Hypothesen. Das hat den Effekt, dass dem Partner signalisiert wird, worüber man gerade nachdenkt, er einen Eindruck vom eigenen Wissenstand erhält.

Es liegt nahe, dass die Entwickler solche Hypothesen verbalisieren, um sie mit dem Partner zu diskutieren und so das Verständnis zu stärken.

Beispiele

- In der Sitzung JA1 erklärt der *Experte* P1 die Funktionsweise des Systems. Während der Erklärung will sein *Schüler* P2 detailliertere Informationen zu einem bestimmten Aspekt haben. Die vollständige Episode findet sich in Tabelle VI. An diesem Beispiel sieht man, dass P2 tatsächlich an der Information interessiert ist; trotz des Missverständnisses fragt er weiter nach, bis seine Frage geklärt ist.

- In der Sitzung CA3 sitzen die Entwickler an einem System, dass sie grob kennen. Ihnen fehlt aber wichtiges Detailwissen, um die ihnen gegebene Aufgabe zu lösen. Sie nehmen beide eine Rolle als *Schüler* ein, studieren vorhandenen Code und versuchen Informationen zu sammeln. Zu Beginn einer Episode schaut sich das Paar Code an, von dem es vermutet, dass er wichtig ist:

  > *P1*: „Guck mal, wir leiten davon ab, geben nen Layer rein und was immer nen `MapSelectionContext` is“

  P1 nutzt daraufhin eine Funktion des Editors, um einen kurzen Überblick über den `MapSelectionContext` zu erhalten. P2 hat zu dem Code noch eine andere Frage:

  > *P2*: „Vom (∼Eventmodel) kriegen wir dann solche Daten wie ob das nen Multithema is oder so“
  > *P1*: „Hmm, wahrscheinlich“

- In der Sitzung CA4 kann man einen *Schüler* (P2) beobachten, der recht viele Fragen zu den Erklärungen seines Partners stellt, mit dem Ziel, das System besser zu verstehen.

  > *P1*: „Es gab diese Poly-, `PolyMovePointHandler`-Methode, die den `MoveHandler` erzeugt oder `null` zurück gibt. [...]“
  > *P2*: „Ok”
  > *P1*: „Die macht halt hier als erstes“
  > *P2*: „Äh, wann (.) tut die das? (.) Wann wird die aufgerufen? Wann?“
  > *P1*: „Ähmm, pre-mousemove“
  > *P2*: „Ok“
  > *P1*: „Und dann gibt’s entweder ’n `MoveHandler` oder es gibt ihn nicht.“

Konzept	**Äußerung**	**Kommentar**
ask_knowledge	P2: In was für nem Zeitfenster wird da geguckt?	Es geht um eine Methode, die regelmäßig überprüft, ob sich eine Datei verändert hat.
explain_knowledge	P1: Ähm, ich fang an zu gucken um 2 Minuten nach der vollen Stunde, weil da garantiert ist, dass da Nachrichtendateien vorliegen, wenn welche vorliegen	P1 denkt, P2 will wissen, wie oft die Komponente überprüft, ob eine neue Datei vorhanden ist.
agree_knowledge	P2: OK	P2 signalisiert, dass er den Erläuterungen folgt.
explain_knowledge	P1: Und monitor dann eben so lang diese Datei, bis sie fertig ist.	
ask_knowledge	P2: Mmm, genau aber also das Zeitfenster für die Veränderung?	P2 versucht seine eigentlich Frage noch einmal genauer zu formulieren.
explain_knowledge	P1: Ja, genau, das ist, äh, Zeitfenster für die Veränderung das ist variabel [...]	
ask_knowledge	P2: Ja [du wartest also, bis] die Größe aufhört sich zu verändern, ja? Musste ja nen Zeitfenster einplanen, in dem immer noch ne Veränderung stattfinden kann.	
explain_knowledge	P1: Ja gut, bis maximal 5 vor der neuen Stunde, so ich warte wirklich lange	
agree_design	P2: Ne, ich meine [...] die Größe des Zeitfensters, also du wartest 10 Sekunden und <* wenn sich dann nichts ändert, *> dann ist die Datei <* fertig *>	Um seine Intention klarer zu machen, schlägt P2 eine Antwort auf seine Frage vor.
agree_design	P1: Achso, das meinst du, ne 30 Sekunden.	P1 versteht, was P2 erwartet und liefert die Antwort.

Tabelle 6: typische Handlung eines *Schülers*. Aus der Sitzung JA1

> *P2*: „Und der bestimmt auch schon den Mousecursor? Quasi, der angezeigt wird?“

5.1.3 Rolle: Verwalter der Orientierungsphase

In einigen Sitzungen gibt es für einen der Partner sehr viel Unbekanntes oder Ungeklärtes. Ein Vorteil der Paarprogrammierung ist, das Wissenstransfer über viele verschiedene Themen natürlich auftritt. Allerdings kann es während einer Orientierungsphase problematisch sein, alles Mögliche zu besprechen. Dadurch kann das Ziel der Orientierungsphase aus dem Fokus geraten und andere für die Sitzung wichtige Themen werden nicht besprochen. Daher nimmt in einigen Sitzungen ein Partner eine Rolle als *Verwalter der Orientierungsphase* ein.

Aktivitäten des Verwalters

Der *Verwalter* reflektiert darüber, was gerade passiert, und ob es dem Ziel der Orientierungsphase zuträglich ist. Falls nicht, entscheidet er, ob es Sinn macht das Ganze zu verschieben und später zu besprechen.
Typisch für den *Verwalter* sind Metaaussagen wie: „Wir reden jetzt schon viel zu lang über das Thema“ und Vorschläge, bestimmte Themen später oder gar nicht zu besprechen.

Ziel des Verwalters

Der *Verwalter* möchte die Orientierungsphase möglichst schnell und reibungslos zum Ziel führen, ohne dabei Wichtiges zu übersehen.
Markant ist das Vorschlagen von Todos, wenn ein Thema angesprochen wird. Wie in Sektion 6.3 beschrieben, gibt es verschiedene Gründe aus denen es Sinn macht, ein Thema zu verschieben. Das Verschieben oder Nichtbehandeln bestimmter Themen ist ein Weg die Orientierungsphase zu fokussieren. Das ist aber nicht unbedingt das Ziel dessen, der den Vorschlag macht, etwas später zu besprechen. In einigen Situationen macht es den Anschein, dass ein Thema verschoben wird, weil der Sprecher sich damit nicht beschäftigen will. In den meisten Situationen gibt es aber keinen anderen, offensichtlichen Grund für das Verschieben eines Themas. Daher halte ich es für sinnvoll, den Effekt der Handlung ihr auch als Ziel zu unterstellen.

Metagespräche über aktuelle Themen, insbesondere solche die daraufhin weisen, dass das aktuelle Thema schon lange diskutiert wird, oder nicht ganz passt, sind ein Indiz dafür, dass der Sprecher auch eine Vorstellung davon hat, was in der aktuellen Phase besprochen werden sollte, und das lange Unterbrechungen problematisch sein können.

Beispiele

- In der Sitzung JA2 gibt es eine Episode, in der sich das Paar über die Benennung einer Methode unterhält. P1 zeigt seinem Partner P2 die Methode `setInputFile`, nachdem er ihre Funktionalität erklärt hat.

  > *P2*: „Wie macht man das?“
  > *P1*: „Äh, `Job.setInputFile`“
  > *P2*: „Das ist aber sehr mis-, misleading!“

  Daraufhin versucht P1 die Benennung zu rechtfertigen, schließlich einigt sich das Paar, die Funktion umzubenennen.

  > *P1*: „Also, `set` ist vielleicht dann falsch, dann eher `add` vielleicht“
  > *P2*: „Ja“

  P1 macht dann etwas typisches für den *Verwalter der Orientierungsphase*, er weißt seinen Partner daraufhin, dass diese Diskussion recht lange gedauert hat, und nicht zur Orientierungsphase passt und schlägt vor, die Umsetzung (d.h. die Methode umzubenennen) auf Später zu verschieben.

  > *P1*: „Soll ich's jetzt machen? Weil wir verlieren uns schon wieder jetz hier in anderen Dingen“
  > *P2*: „Mmm, du hast Recht (.) ähm, dann lass uns doch vielleicht gerade irgendwie Methode definieren“
  > *P1*: „Ich mach 'nen *TODO* hin”

- Während einer kurzen Episode in der Sitzung CA4 schlägt der Entwickler P1 vor, herauszufinden, wie man das Mausicon ändern kann. Sein Partner P2 weiß, dass das ein schwer zu verstehendes Problem ist und schätzt es als verhältnismäßig unwichtig ein:

  > *P2*: „Ich würde erst abwarten, ob's passiert, das ist dann nicht leicht durchzuschauen, wann steigen wann (.) aber sollten wir im Hinterkopf haben.“

- Der Kontext der Sitzung DA2 ähnelt dem der JA-Sitzungen. Auch hier gibt es einen Entwickler, der das zu bearbeitende System überhaupt nicht kennt und der daher sehr viele Fragen stellt. Im Unterschied zu de JA-Sitzungen nimmt der *Experte* hier aber nicht zusätzlich die Rolle eines *Verwalters* ein, sodass die Orientierungsphase in DA2 sehr schleppend vorangeht und das Paar sich oft in Details verliert.

5.1.4 Rollenanerkennung

Bei meinen Untersuchungen ist aufgefallen, dass sich viele Verhaltensmuster der Entwickler erklären lassen, wenn man in Betracht zieht in wie fern sie die Rolle ihres Partners akzeptieren. Ein *Schüler*, der versteht, dass sein Partner mehr über das aktuelle Problem weiß und dessen Rolle als *Experte* anerkennt, neigt zum Beispiel eher dazu dessen Ausführen bis zum Ende zuzuhören, als einer, der die Rolle aus verschiedenen Gründen nicht anerkennt.

Unter den untersuchten Sitzungen, waren diejenigen, bei denen es viele Indikatoren für eine Nichtakzeptanz einer Rolle gibt, eher „schlecht“. Das heißt das Paar kommt nur langsam voran, es dauert sehr lange, bis die Partner sich einigen oder Aktivitäten müssen häufig unterbrochen werden, was Zeit kostet. In einigen Fällen kann genauer nachvollzogen werden, wie die Nichtanerkennung einer Rolle einem späteren Problems zuträglich ist - weil der *Experte* beispielsweise einfach nicht dazu kommt, wichtiges Wissen zu vermitteln.

Gerade in den Sitzungen, in denen einer der Partner deutlich mehr relevantes Wissen hat als der andere, ist davon auszugehen, dass das beiden Partnern bewusst ist. In den meisten Sitzungen wird das sogar direkt aus den Gesprächen klar.
Zum Beispiel beginnen die Ausführungen eines Entwickler in Sitzung CA1 mit:

> „Also wie gesagt, ich habe schon angefangen, ungefähre 'ne Stunde programmiert“

Es ist klar, dass sein Partner das, was er in dieser Stunde programmiert hat, noch nicht kennen kann.

Dennoch gibt es in einigen dieser Sitzungen Indikatoren dafür, dass die Rolle des *Experten* nicht anerkannt wird. Solche Indizien sind beispielsweise, wenn der *Experte* häufig vom Partner unterbrochen wird, seine Ausführungen angezweifelt werden, oder er nicht dazu kommt sie zu beenden. Weiteres dazu in Abschnitt 6.2.
Es gibt auch Verhaltensmuster, die darauf schließen lassen, dass die *Expertenrolle* anerkannt wird. Zum Beispiel schlägt der *Schüler* in der Sitzung CA4 einen Arbeitsschritt vor. Sein Partner hält diesen aber für aktuell unnötig und zeitraubend. Der *Schüler* vertraut auf diese Einschätzung.

Beispiel

Die Sitzung CA2 ist ein sehr gutes Beispiel für verschiedene hier beschriebene Aspekte. Der *Experte* P1 hat seinem Partner P2 viel zu erklären, kommt

aber aus verschiedenen Gründen nicht dazu, seine Gedanken verständlich zu machen. Ein wichtiger Grund ist, dass P2 sein vorgeschlagenes Design anzweifelt (ohne dass P1 mit der Erklärung dazu fertig ist) und ihn regelmäßig in seinen Ausführungen unterbricht, um eine anderes Design vorzuschlagen und zu rationalisieren.
Die ersten 40 Minuten der Sitzung sind davon geprägt, dass die Partner ihre eigenen Ziele verfolgen - P1 will seinem Partner neuen Code erklären und dessen Design rationalisieren, während P2 dieses Design nicht nachvollziehen kann, ein anderes vorschlägt und beginnt zu implementieren. Dabei ist ihm eine Anforderung, die an das System gestellt ist aber nicht klar. P1 weiß davon, schafft es aber nicht die Anforderung seinem Partner verständlich zu machen. Das wird erschwert, da dieser ihn häufig unterbricht und den Sinn dessen Designs hinterfragt.

5.2 Phasen in der Paarprogrammierung

Während der Untersuchung sind mehrere Phänomene aufgefallen. Dabei fällt klar auf, dass bestimmte Phänomene in verschiedenen Phasen auftreten. Daher macht es Sinn, erst verschiedene mögliche Phasen innerhalb der Orientierungsphase zu identifizieren. Diese dienen dann wiederum als näherer Kontext für die Phänomene, die auftreten.
In der Literatur wird auch von Phasen in der Paarprogrammierung gesprochen. Oft wird der Begriff *Phase* allerdings nicht genauer definiert. Zum Beispiel ist in [2] von *design, debugging, testing and coding* die Rede. Die in der Literatur beschriebenen Phasen gelten meist für den gesamten Softwareentwicklungsprozess. Da diese Arbeit einen Fokus auf einen vergleichsweise kleinen Teil des Paarprogrammierungsprozesses hat, macht diese Einteilung im Rahmen dieser Arbeit keinen Sinn.
Bei der Analyse des vorliegenden Materials fällt aber dennoch auf, dass die Themen, die vom Paar besprochen werden, logisch zusammenhängen und dass viele Phänomene gehäuft in bestimmte Phasen auftreten.
Eine Phase, in der das Paar Code sichtet und versucht schon vorhandenen Code zu verstehen, unterscheidet sich markant von Phasen, in denen das Paar Designvorschläge eines zu entwickelnden Softwareartefakts diskutiert. Phasen können aber durchaus fließend ineinander übergehen und sich häufig abwechseln.

5.2.1 Phase: Stand des Systems

Beschreibung der Phase

In nahezu allen untersuchten Sitzungen war das Ziel ein bestehendes Softwaresystem zu erweitern oder zu verändern. Oft sind nicht alle, oder nur wenige Teile des Systems relevant für das zu erreichende Ziel. Diese müssen aber verstanden werden, bevor die Änderungen und Erweiterungen mit dem

gewünschten Effekt implementiert werden können. Auch wenn es noch kein vorhandenes System gibt, wird bei der Softwareentwicklung regelmäßig auf Bibliotheken oder externe Dienste zugegriffen. Um sie korrekt verwenden zu können, müssen zumindest die angebotenen Schnittstellen verstanden werden.
Es gibt also nahezu immer Wissen, welches bei den Partnern vorhanden sein sollte, bevor sie ein gestelltes Problem lösen können.

Diese Phase ist ein guter Ansatzpunkt für den Wissensaustausch im Paar - wie in 2.1 beschrieben, ist es ein Vorteil der Paarprogrammierung, dass solch ein Informationsaustausch natürlich stattfindet. Der Wissensaustausch kann aber auch unnötig sein und Zeit rauben, sollten beide Partner schon über das Wissen verfügen. Deshalb kann man einige Äußerungen beobachten, die darauf abzielen den Wissenstand des Partners einzuschätzen.
Außerdem gibt es viel Wissen, das für das Sitzungsziel irrelevant ist. Auf einer Seite kann es nützlich sein, schnell zu erkennen, sobald die Konversation von etwas handelt, das nicht zum Ziel führt, auf der anderen Seite kann es auch sehr wertvoll sein über Dinge zu reden, die nicht wichtig für das aktuelle Ziel sind, aber vielleicht bei einer späteren Aufgabe helfen.
Am Ende dieser Phase haben die beiden Partner im Allgemeinen ein konsistentes mentales Modell von den wichtigen Teilen vorhandener Systeme, um einen Plan erstellen zu können, das Sitzungsziel zu erreichen.

Beispiele

- In der Sitzung CA2 versucht einer der Entwickler alles, was er für wichtig hält zu erklären. Sein Partner hat dabei eine wichtige Anforderung des Systems nicht vor Augen und kann deshalb viele Begründungen nur schlecht nachvollziehen. Er fängt später an, das System nach seinen Vorstellungen zu modifizieren, dabei fällt ihm auf, dass seine Methode wegen der vergessenen Anforderung nicht zielführend ist.

- In der Sitzung CA3 versuchen die Entwickler ein System zu erweitern, dass sie nur grob kennen. Zu Beginn der Sitzung durchsuchen sie vorhandenen Code nach Klassen und Methoden, die bei der Weiterentwicklung genutzt werden könnten und versuchen deren Implementierung zu verstehen.

5.2.2 Phase: Weg zum Ziel der Sitzung

Beschreibung der Phase

In dieser Phase generiert das Paar aus dem gemeinsamen Wissen Vorschläge, um ein vorliegendes Problem zu lösen. Wenn das Paar ein konsistentes mentales Modell von den wichtigen Teilen des aktuellen Systems und dem

zu lösenden Problem hat, können sich die Entwickler mögliche Lösungswege vorstellen und diese diskutieren. Während dieser Diskussion kann dem Paar auffallen, dass es noch Wissenslücken gibt, dadurch verschwimmt die Grenze dieser Phase zu *Stand des Systems* in einigen Sitzungen.
In einer der untersuchten Sitzungen hat das Paar während dieser Phase mehr oder weniger geraten, wie es dem Sitzungsziel näher kommen kann.
In jedem Fall evaluieren die Partner die Lösungsvorschläge und einigen sich im Idealfall auf einen Weg, dem sie dann den Rest der Sitzung folgen können.
Zusätzlich zum Design der zu entwickelnden Lösung besprechen die Partner während dieser Phase die weitere Strategie.

Die Konzepte Design und Strategie sind in [13] ausführlich beschrieben, ein kurzer Überblick ist in Sektion 3.3 zu finden.
Wie gut das Design durchdacht ist und von beiden Partnern verstanden wurde, hat oft Einfluss auf den weiteren Verlauf der Sitzung.

Beispiele

- In der Sitzung CA1 gibt es erst eine längere Episode, in der das Paar bespricht, wie genau ein neues GUI-Element aussehen soll, damit es möglichst benutzerfreundlich ist. Sie einigen sich auf eine Lösung, nachdem sie Vor- und Nachteile verschiedener Alternativen besprochen haben. Sie gehen danach fließend darin über, vorhandenen Code zu modifizieren.

- In der Sitzung KA3 gibt es eine Episode, in der die Partner die Strategie besprechen, nach der sie vorgehen wollen. Die vollständige Episode ist in Tabelle VII zu finden.

- In der Sitzung KA7 bespricht das Paar erst einen zu verrichtenden Arbeitsschritt, der relativ schnell erledigt ist. Die Entwickler kümmern sich dann um etwas Organisatorisches und greifen danach erneut eine Diskussion über die zu verfolgende Strategie auf. In dieser Sitzung gibt es die Phase *Weg zum Ziel* doppelt, in beiden Fällen geht es um ein verschiedene Ziele.

6 weitere Konzepte

Im Folgenden beschreibe ich Phänomene, die bei der Untersuchung aufgefallen sind und deren Auftreten oder deren Behandlung potentiell Einflüsse auf den Rest einer Orientierungsphase oder gesamten Sitzung haben. In vielen Fällen sind die Phänomene schon teilweise in [13] beschrieben. Hier werden sie allerdings aus anderen Blickwinkeln betrachtet oder erweitert. Außerdem beschreiben die Konzepte im [13] einzelne Äußerungen, während die Konzepte hier gewöhnlich von längeren Episoden ähnlicher Äußerungen handeln.

Konzept	**Äußerung**	**Kommentar**
propose_strategy	P1: Fang' ma an, also am besten fang' ma bei der Entität an, ziehen das raus, und gucken was kaputt geht? Oder, wie wollma vorgehen?	Die Partner haben zuvor das Ziel der Sitzung geklärt.
amend_strategy	P2: Joa, ich würd sagen, wir legen erst ma ne Entität an, und dann, an den Stellen wo [...] diese Felder geschrieben werden, da stellen wir das um auf die Entity und dann	
agree_strategy	P1: Hmm (..) ja.	
amend_strategy	P2: Das kann ja eigentlich gar nicht so viel sein, oder?	Die Frage zielt vermutlich darauf ab, nochmal Zustimmung vom Partner zu erhalten. Weil die Aussage zuvor zögerlich schien.
agree_strategy	P1: Ja, denke ich auch.	

Tabelle 7: *Strategie*besprechung während der Sitzung KA3

Die Konzepte werden alle in folgendem einheitlichen Format beschrieben:

1. **Allgemeine Beschreibung**: In dieser Sektion wird das Konzept kurz beschrieben.
2. **Behandeltes Phänomen**: Die meisten beschriebenen Konzepte behandeln ein bestimmtes Phänomen, dieses wird hier kurz erläutert.
3. **Ursächliche Bedingung**: Dem Auftreten des behandelten Phänomens liegen bestimmte Bedingungen zugrunde. Wenn möglich, werden diese hier angemerkt.
4. **Rolle des Kontext und intervenierende Bedingungen**: Hier wird der nähere Kontext beschrieben, in dem das behandelte Phänomen auftritt. Außerdem werden, wenn vorhanden, diejenigen Bedingungen beschrieben, welche die Handlungsstrategien der Entwickler einschränken.
5. **Konsequenzen**: In diesem Abschnitt werden potentielle und beobachtete Konsequenzen aus dem Verhalten beschrieben.
6. **Beispiele**: In dieser Sektion versuche ich das Phänomen mit anschaulichen Beispielen zu untermalen. Insbesondere werden hier Episoden aus Sitzungen vorgestellt, an denen die beschriebenen Konsequenzen nachzuvollziehen sind.

6.1 Kategorie: Wissenstransfer

In dem Fall, dass es ein bestehendes Softwaresystem gibt, an dem das Team arbeitet, nimmt einer der Partner wie erwähnt oft die Rolle eines *Experten* ein, der seinem Partner alles Wichtige erklärt. Wissenstransfer ist also ein wichtiger Prozess in diesem Kontext. Eine Wissenstransfer-Episode besteht dann daraus, dass der *Experte* versucht seinem Partner etwas zu erklären, bis er sich hinreichend sicher ist, dass dieser es verstanden hat.
In der Basisschicht werden einige Konzepte zu Wissen und Wissenstransfer beschrieben - siehe [13, S. 52]. Außerdem gibt es in [24] nähere Erläuterungen zu verschiedenen Aspekten des Wissenstransfers.
Hier wird der Blick allerdings auf längere Episoden während der Paarprogrammierung gerichtet, in denen Wissen ausgetauscht wird und welche Konsequenzen mögliche Unterschiede in diesen Episoden auf das Verständnis des Schülers und somit auf den Rest der Sitzung haben. Insbesondere ist die Abstraktionsebenen des Wissensaustausch für den weiteren Verlauf der Sitzung interessant.

Wissenstransfer auf hoher Abstraktionsebene

Beschreibung

Wissenstransfer während der Orientierungsphase heißt meist, dass ein *Experte* seinem Partner den aktuellen Stand eines vorliegenden Systems erklärt. Zu den Themen auf hohem Abstraktionslevel gehören dabei grobe Zusammenhänge in der Funktionsweise des Systems, oder Anforderungen, die an das System als Ganzes gestellt sind. Eine Episode besteht dann daraus, dass der *Experte* versucht seinem Partner ein solches Konzept zu erklären, bis er sich hinreichend sicher ist, dass dieser es verstanden hat.

behandeltes Phänomen

Im Grunde wird hier adressiert, dass einer der Partner weniger weiß als der andere. Dafür kann es mehrere mehr oder weniger offensichtliche Indikatoren geben. So könnte der *Schüler* beispielsweise direkt fragen, wie denn eine Softwarekomponente aufgebaut ist. In vielen Fällen ist dem Paar von Anfang an bewusst, dass einer der Partner mehr weiß als der andere - zum Beispiel wenn einer direkt zuvor ohne den Partner neuen Code geschrieben hat.

ursächliche Bedingung

Die ursächliche Bedingung dafür dass ein Partner weniger weiß als der andere ist oft im Kontext zu finden, in dem die Sitzung stattfindet.

- Einer der Partner hat kurz zuvor neuen Code geschrieben, den der andere nicht kennen kann.
- Einer der Partner kennt das System oder die Technologien, mit denen gearbeitet wird, nicht. Weil er zum Beispiel neu in dem Unternehmen ist.
- Einer der Partner hat weniger Erfahrung in der Entwicklung von Software. Beispielsweise könnte einer der Partner gerade in seinem Studium sein, während der andere schon seit langem sein Studium abgeschlossen hat und deutlich mehr Zeit hatte, praktische Erfahrung zu sammeln.

Es gibt sicher noch weitere andere Gründe, aus denen ein Partner weniger weiß, als der andere, aber diese Diskussion würde hier zu weit führen.

Kontext

Um das Phänomen *meinem Partner fehlen Informationen* zu behandeln, muss eben das bewusst sein. Das heißt, irgendwie muss mindestens einer der Partner wissen, dass es zwischen ihnen ein Wissensgefälle gibt, und den Wissenstransfer initiieren.

Wie erwähnt, ist es in vielen Fällen offensichtlich, dass einer der Partner weniger oder mehr weiß. In den anderen Fällen kann diese Information durch gezieltes Fragen (z.B.: „Weißt du, wie X funktioniert?“) erhalten werden.

Konsequenzen

Wenn alles wie geplant läuft, hat der Wissenstransfer geklappt und der *Schüler* weiß am Ende der Episode mehr also vorher.
In einigen Situationen ist eine ausschließlich hohe Abstraktionsebene beim Wissenstransfer ungünstig. Einige Menschen verstehen Dinge besser, wenn sie Details und Beispiele kennen. Unter den untersuchten Sitzungen gab es allerdings keine, in denen ausschließlich auf hoher Abstraktionsebene Wissen transferiert wurde.
Sollte dem *Schüler* die Abstraktionsebene zu hoch sein, kommt es oft dazu, dass er seinen Partner unterbricht (siehe 6.2).
Führt das Fehlen von Detailinformationen zu Missverständnissen oder Wissenslücken, kann das später in der Sitzung zum Problem werden. Mehr dazu in Abschnitt 6.4

Beispiele

- In der Sitzung CA1 beginnt einer der Entwickler P1 damit, seinem Partner neuen Code zu zeigen. Er geht dann aber dazu über, die Neuerungen anhand einer Demo zu demonstrieren. Das Verhalten eines Systems an einer Demo nachzuvollziehen, geschieht auf einer deutlich höheren Ebene, als es direkt am Code zu tun. Es geht weniger um Details.

- Der *Experte* beginnt in der Sitzung JA1 seine Erklärungen auf einer hohen Abstraktionsebene. Teilweise liegt das sicher daran, dass er keine andere Wahl hat, es dauert noch eine Weile, bis die Entwickler den aktuellsten Stand des Codes sehen können.

  > „Ok, aber ich kann dir ja solang schon mal sagen, was dieses Plugin im Großen und Ganzen tut.“

  Er erklärt, dass das Plugin stündlich einen Nachrichtenmitschnitt generiert. Er beschreibt dann weiterhin, dass das Plugin wiederum aus anderen Komponenten besteht und erklärt grob deren Aufgabe.

  Etwas später in der Sitzung kann man beobachten, wie der *Schüler* seinen Partner unterbricht, um detailliertere Informationen zu einem Prozess zu erfragen.

Wissenstransfer auf niedriger Abstraktionsebene

Beschreibung

Zu den Episoden des Wissenstransfers auf niedriger Abstraktionsebene gehören solche, bei denen ein Entwickler seinem Partner die detaillierte Funktionsweise oder Designüberlegungen eines spezifisches Softwareartefakts erklärt. Es geht dabei typischerweise um genaue Erklärungen der Funktion bestimmter Codefragmente und der Gründe, warum der Code wie er ist produziert wurde und nicht anders.

behandeltes Phänomen

Wie beim *Wissenstransfer auf hoher Abstraktionsebene* wird hier das Problem adressiert, dass einem Partner wichtiges Wissen fehlt während der andere es besitzt.

ursächliche Bedingung

Bei hinreichend großen Systemen macht es keinen Sinn anzunehmen, dass die Entwickler sich mit allen Details auskennen. Wenn also jemand bestimmte Details nicht kennt, heißt das einfach, dass er sich damit noch nicht beschäftigen musste. Es macht für diese Arbeit wenig Sinn, weitere Gründe dafür zu suchen.

Kontext

Der nähere Kontext unterscheidet sich hier nicht offensichtlich von dem des *Wissenstransfer auf hoher Abstraktionsebene.*

Konsequenzen

Wenn nur über Details geredet wird, kann es passieren, dass dem *Schüler* grobe Zusammenhänge entgehen. Er versteht dann möglicherweise nicht, wie die Einzelteile, die gut verstanden wurden, zusammenhängen.

Beispiele

- In der Sitzung CA2 will einer der Entwickler P1 seinem Partner P2 Änderungen erklären, die er vor Kurzem am vorliegenden System vorgenommen hat. Er beginnt seine Erklärung auf einer niedrigen Abstraktionsebene:

  > *P1*: „Also was ich gemacht hab in Absprache mit <**Name**>, das ist das `IFeatureAttributeConfiguration`, erweitert um nen `IVirtualColumn`.“

Sein Partner will sich daraufhin den Code genauer anschauen. Im weiteren Verlauf der Sitzung will P2 weiter in erster Linie vorhandenen Code studieren, während P1 ihm dazu Fragen beantwortet und versucht, sein geplantes Design zu erklären. Das Paar bleibt für den Großteil der Diskussion bleibt auf einer relative niedrigen Abstraktionsebene.

> *P1*: „Also hier habe ich momentan noch nicht mehr gemacht, als dass ich aus dem Dialog die entsprechenden Werte abgeholt hab, gesetzt hab, ich habse vorher aus der Konfiguration rausgeholt, und wieder zurückgesetzt, damit die, also und also und gelöscht, damit wenn das Dialog, der Dialog aufgerufen wird, die nicht zusätzlich mit angezeigt werden, wenn die über das `AllColumnAttributes` zu, aufzuholen wären."

P2 kann das von P1 vorgeschlagene Design nicht nachvollziehen, schlägt eine deutlich einfachere Lösung vor und beginnt diese umzusetzen. Erst ungefähr 30 Minuten später erkennt P2, warum sein Vorschlag nicht funktioniert. P1 hatte es nicht geschafft, eine Anforderung mitzuteilen, die das Design einhalten musste.

- In der Sitzung CA4 hat einer der Partner - P1 einen Monat zuvor an dem Code gearbeitet, den das Paar in dieser Sitzung erweitern möchte. Er beginnt seinem Partner P2 auf recht niedriger Abstraktionsebene den Code zu erklären.

  > *P1*: „Ich habe für den letzten eine, eine Factory eingeführt, also für die Teilgeometriemethoden, ähm, genau (..) die (...) wir haben ja diese, diese, (..)"
  > *P2*: „Ja, is richtig, das ist die `create`-Methode"

  P1 will dann direkt am Code Zusammenhänge erklären.

  > *P1*: „Es gab diese Poly-, `PolyMovePointHandler`-Methode, die den `MoveHandler` erzeugt oder `null` zurückgibt."

  P2 unterbricht ihn daraufhin, ihm fehlen noch etwas abstraktere Informationen

  > *P2*: „Äh, wann (.) tut die das? (.) Wann wird die aufgerufen? Wann?"
  > *P1*: „Ähmm, pre-mousemove"

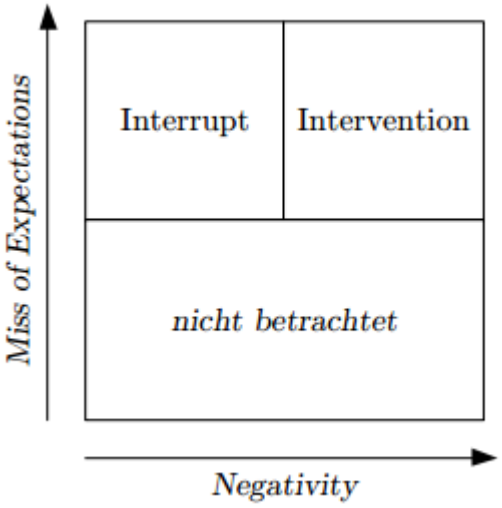

Abbildung 4: Spektrum der Dimensionen Negativity und Miss of Expectations. Abbildung aus [23, Abb. 3.1].

6.2 Kategorie: Unterbrechungen

Während der Erklärungen des *Experten* haben die *Schüler* ihn in jeder untersuchten Sitzung mindestens ein Mal unterbrochen. Die Unterbrechungen hatten verschiedene Motivationen.
Unterbrechungen sind im Kontext der Orientierungsphasen interessant, weil die Art und der Grund der Unterbrechungen ein guter Indikator für beispielsweise die *Rollenanerkennung* sind, besonders wenn ähnliche Arten von Unterbrechung gehäuft auftreten.
In [23] beschreibt `Franz Zieris` Konzepte zur Beeinflussung der Aktivität des Partners bei der Paarprogrammierung. Insbesondere beschreibt er, dass Unterbrechungen sich in zwei Dimensionen unterscheiden: *miss of expectation* und *negativity*. Er beschreibt in [23] nur solche Phänomene mit hohem Grad an *miss of expectation*. Einen Überblick liefert die Abbildung 4.
Der Blickwinkel auf Beeinflussungen ist in dieser Arbeit ein wenig anders - interessant für den Verlauf der Orientierungsphase ist, in wie weit die Motivation der Unterbrechungen mit der Rolle des Unterbrechenden übereinstimmen. Dennoch sind in den beobachteten Szenarien Korrelationen zwischen der Motivation und dem Grad an *miss of expectation* aufgefallen.

zur Rolle passende Unterbrechungen

Beschreibung

In vielen Situationen wurde der *Experte* unterbrochen, weil sein Partner etwas nicht verstanden hat, zu einem bestimmten Thema mehr Details er-

fahren wollte oder ihm ein grober Überblick über das Besprochene fehlte. In diesen Fällen haben die Unterbrechungen einen geringen bis mittleren Grad an *miss of expectation* und geringen Grad an *negativity*.
Im späteren Verlauf der Orientierungsphase, wenn sich das Paar über das Design einer neuen Funktionalität oder einer Änderung unterhält, kommen noch Unterbrechungen mit höherem Grad an *negativity* hinzu - wenn einer der Entwickler einen Vorschlag schlecht findet und ablehnt.
Diese Unterbrechungen haben aber alle, aus Sicht des *Schülers*, das Ziel mehr zu erfahren oder zu verstehen, beziehungsweise einem sinnvollen Design näher zu kommen.

behandeltes Phänomen

Während der Ausführungen seines Partners fällt einem Entwickler auf, dass er etwas nicht verstanden hat, oder ihm zu dem Gesagten noch weitere Dinge interessieren.

Kontext

Während der Ausführungen eines Entwicklers fällt seinem Partner ein Problem auf, dass er geklärt sehen möchte. Es ist aus der Sicht des Partners aber nicht absehbar, dass die Ausführungen ein zeitnahes Ende haben werden, sodass es seiner Meinung nach nötig ist, die Ausführungen zu unterbrechen, um das Problem zu klären.

intervenierende Bedingungen

Dem Zuhörer muss auffallen, dass es in den Erklärungen seines Partners ein Problem gibt - etwas was nicht ausreichend erklärt wurde, falsch war oder nicht verstanden wurde. Außerdem muss das Problem als wichtig genug eingeschätzt werden, um den Partner zu unterbrechen.

Konsequenzen

Wenn viele Unterbrechungen dieser Art auftreten und der Unterbrochene versuchtm die auftretenden Fragen zu beantworten - was normales Verhalten ist, kann es sein, dass das Paar sich immer weiter in Details verliert oder vom Einstiegsthema abschweift und das ursprüngliche Ziel vergisst.
Bei einigen Paaren ist ein klarer Mechanismus zu beobachten, mit dem sie nach Unterbrechungen zum Ursprungsthema zurückkehren, sodass sie weniger dazu neigen weit abzuschweifen.

Beispiele

- In den Sitzungen JA1-9 arbeitet ein Entwickler P1, der den zu bearbeitenden Code selber geschrieben hat mit einem externen Entwickler P2, der die Rolle des *Schülers* einnimmt, zusammen. Da P2 das System

absolut nicht kennt, gibt es in den Sitzungen recht häufig Unterbrechungen von P2, der meist etwas von P1 erwähntes genauer besprechen möchte. Häufig bittet P2, nachdem seine Frage geklärt wurde, seinen Partner explizit mit dem ursprünglichen Thema fortzufahren. In der Sitzung JA1 erklärt P1 die Funktonalität eines Newsplugins. P2 unterbricht ihn dabei, weil er zu einem Detail gern genauere Informationen hätte. Sobald er diese hat, erinnert er seinen Partner an das ursprüngliche Thema:

> *P2*: „Und das Newsplugin macht jetz von dieser ganzen Sache was davon?“

- In der Sitzung DA2 gibt es eine ähnliche Situation. Der Entwickler P1 kennt das System grob, während P2 es gar nicht kennt. Auch hier kommt es zu häufigen Unterbrechungen durch P2. Sein Partner versucht, seine Fragen sofort zu beantworten, das Paar hat aber keinen klaren Rückkehrmechanismus, wie man ihn in den JA-Sitzungen beobachtet. Das heißt, häufig während P1 etwas erklärt, fällt P2 etwas Neues auf, er fragt nach und P1 beginnt nun nach eine Erklärung dafür zu suchen. Dadurch ist die Orientierungsphase in dieser Sitzung sehr lang. Es werden viele Fragen beantwortet, allerdings kommt das Paar nur schleppend voran.

nicht zur Rolle passende Unterbrechungen

Beschreibung

In den untersuchten Sitzungen waren Unterbrechungen zu beobachten, die im Unterschied zu denen in Sektion 6.2 zum Ziel hatten, ein komplett anderes Thema zu besprechen. Die Unterbrechungen in Sektion 6.2 sind Reaktionen auf gerade Gesagtes und zielen darauf ab, zu dem Gesagten passende Subthemen zu besprechen, um beispielsweise Details oder Missverständnisse zu klären. Bei den hier beschriebenen Unterbrechungen will der Unterbrecher das aktuelle Thema nicht weiterführen, weil er es beispielsweise für unwichtig hält, und durch ein komplett anderes Thema ersetzen.
Unterbrechungen dieser Art haben einen vergleichsweise hohen Grad an *negativity* und sind zumindest unerwarteter als *zur Rolle passende Unterbrechungen.*
Des Weiteren sind Unterbrechungen dieser Art meist ein guter Indikator dafür, dass die Rollenverteilung im Paar entweder nicht klar ist, oder wie in Abschnitt 5.1.4 beschrieben, nicht anerkannt wird.
Die beobachteten Instanzen dieses Phänomens handeln alle von einem *Schüler*, der den *Experten* unterbricht, aber nicht mit dem primären Ziel, neue Informationen zu sammeln oder etwas Missverständliches zu klären.

behandeltes Phänomen

Einer der Entwickler will mit dieser Art Unterbrechung ein aktuell besprochenes Thema beenden und ein neues einleiten. Der Grund dafür ist, dass der Entwickler das aktuelle Thema für unwichtig hält oder ein, seiner Meinung nach besseres alternatives Thema besprechen will. In vielen Fällen treten beide Situationen gleichzeitig auf.

ursächliche Bedingung

Ein Grund dafür, dass ein Thema besprochen wird, welches ein Partner für unwichtig hält, kann sein, dass das Paar falsche Vorstellungen vom Wissenstand des jeweiligen Partners hat. Erklärungen von Konzepten, die ein Entwickler gut kennt und versteht, sind für ihn uninteressant; die Zeit könnte gut für andere Themen genutzt werden.
Es kommt allerdings auch vor, dass einer der Partner sein eigenes Thema besprechen möchte, unabhängig davon ob das aktuelle Thema wichtig ist oder nicht.

Konsequenzen

Im besten Fall kann durch das Abbrechen eines Thema Zeit gespart werden - gerade wenn das Thema von beiden Partnern gut verstanden ist. Das konnte in den untersuchten Sitzungen aber nicht beobachtet werden.
In den untersuchten Sitzungen, ist diese Art Unterbrechung unabhängig davon, ob das Thema schon bekannt ist - es soll einfach ein anderes Thema besprochen werden. Dieses Verhalten ist in den meisten Sitzungen ein guter Indikator dafür dass der Unterbrecher die Rolle des *Experten* nicht anerkennt, was mit insgesamt eher durchzogenen Sitzungen korreliert.

Beispiel

Der Entwickler P1 in der Sitzung CA2 hat vor der Sitzung Code geschrieben. Er nimmt die Rolle des *Experten* ein und beginnt dem *Schüler* P2 den neuen Code zu erklären. Während der Erklärungen zweifelt P2 mehrfach an der Sinnhaftigkeit der vorgestellten Implementierung. Er würde das Problem gern anders lösen und unterbricht die Ausführungen seines Partners. Das Ziel der Unterbrechung ist in dem Fall nicht, wie vom *Schüler* zu erwarten wäre, Neues zu lernen, sondern den eigenen Designvorschlag zu besprechen. Gerade in dieser Sitzung ist dieses Verhalten ein Indikator für ein Nichtanerkennen P1's Expertenrolle. Mehr zu diesem Thema in Abschnitt 5.1.4.

sonstige Unterbrechungen

Es gibt Unterbrechungen, zu denen keine der anderen Beschreibungen in Abschnitt 6.2 passen. Das sind externe Unterbrechungen und solche, bei

denen das Thema komplett gewechselt wird, aber mit der Intention zum ursprünglichen Thema zurückzukehren. Insbesondere fallen in diese Kategorie Unterbrechungen aufgrund verschiedener organisatorischer Probleme.
Für den weiteren Verlauf der Sitzung sind diese Unterbrechungen in den untersuchten Sitzungen größtenteils uninteressant und werden hier deshalb nur kurz erwähnt.

Beispiel

Zu Beginn der Sitzung CA1 fragt einer der Partner, ob das Feature, welches das Paar implementieren soll, destabilisierend wirkt. Im weiteren Verlauf der Unterhaltung versucht das Paar Wege zu finden, diesen Effekt möglichst zu vermeiden.
Dieses Thema musste geklärt werden, da es Einfluss darauf hat, ob und wie das Paar den produzierten Code in die Versionsverwaltung einchecken kann. In diesem Fall resultieren daraus sogar neue Anforderungen für den zu produzierenden Code. Die Unterbrechung an sich hat aber wenig mit der Paarprogrammierungssitzung zu tun. Es ging nur darum ein organisatorisches Detail zu klären.

6.3 Todos

Beschreibung

Wenn einer der Partner versucht, die Unterhaltung zu einem neuen Thema zu bringen, gibt es mehrere Möglichkeiten, damit umzugehen. Entweder wird das neue Thema akzeptiert und darüber geredet, es wird abgelehnt, oder die Unterhaltung zu dem neuen Thema wird auf später verschoben. Das Team einigt sich auf ein *Todo* - das Thema später zu besprechen.
Das Konzept *Todo* wird in [13] und [11] ausführlich beschrieben. Während der Untersuchung von Orientierungsphasen in dieser Arbeit ist aufgefallen, dass nur wenige Teams in der Orientierungsphase potentielle Themen verschieben, die meisten vorgeschlagenen Themen wurden angenommen und sofort besprochen.
Die Entscheidung, dass ein Thema für die aktuelle Orientierungsphase aus irgendeinem Grund unpassend ist, ist eine charakteristische Aktivität des *Verwalters* (siehe 5.1.3).

behandeltes Phänomen

Einer der Entwickler will das Gespräch auf ein neues Thema leiten, welches sein Partner für unpassend hält. Ein Thema könnte unpassend wegen einer Kombination folgender Gründe sein:

1. Das Thema ist weit entfernt vom aktuellen Thema, welches wichtig ist und erst abgeschlossen werden sollte.

2. Das Thema passt nicht zur Orientierungsphase.
3. Es dauert sehr lang das Thema zu besprechen.

In einigen Situationen beginnt das Paar über ein Thema zu sprechen und ihm fällt währenddessen auf, dass es besser verschoben werden sollte - typischerweise weil die Entwickler schon recht viel Zeit darauf verwendet haben und immer weiter von ihrem ursprünglichen Ziel abkommen.

ursächliche Bedingung

Die Frage nach den ursächlichen Bedingungen ist hier die Frage nach dem Grund für das Aufwerfen von Themen, die im Nachhinein als unpassend eingeschätzt werden. Wenn ein Thema vorgeschlagen wird, sieht einer der Partner Bedarf es zu besprechen. Gewöhnlicherweise ist der Grund dafür ein Wissensgefälle im Paar.

Kontext und intervenierende Bedingungen

In den beobachteten Sitzungen kam es erst dazu, dass einige Themen verschoben wurden, wenn es insgesamt recht viele zu besprechende Themen gab. Das geht Hand in Hand mit den beobachteten Situationen, in denen ein Partner die Rolle des *Verwalters* einnimmt (siehe 5.1.3). Wenn nur wenige unpassende Themen vorgeschlagen werden, neigen die beobachteten Paare dazu sie direkt zu adressieren. Erst bei vielen oder zeitraubenden Unterbrechungen der Orientierungsphase haben einige Paare das Verschieben einiger Themen in Betracht gezogen.
Außerdem muss mindestens einer der Partner in der Lage sein einzuschätzen, dass ein bestimmtes Thema aus irgendeinem Grund unpassend ist und seinen Partner davon überzeugen können.

Konsequenzen

Durch das Verschieben bestimmter Themen wird die Orientierungsphase schneller und reibungsloser beendet. Sollten einige Themen falsch als unpassend eingeschätzt werden, kann das allerdings zu Problemen führen. Des Weiteren waren einige Situationen zu beobachten, in denen das Team sich entschieden hat ein Thema zu verschieben, im weiteren Verlauf der Sitzung aber nicht mehr darauf zurück kam.

Beispiele

- In der Sitzung JA2 gibt es ein Szenario, in dem sich das Paar über den Namen einer Funktion unterhält. Ein genauerer Überblick über die Episode ist in Abschnitt 5.1.3 zu finden. Das diskutierte Problem ist recht unwichtig und für das Verständnis dessen, was P1 versucht zu erklären komplett unwichtig. Außerdem passt die Aktivität „Methoden

umbenennen“ nicht in die Orientierungsphase. Das Paar entscheidet sich, sie später durchzuführen.

- In der Sitzung JA3 erklärt der *Experte* P1 einen Teil der Funktionalität des vorliegenden Systems. Er sieht ein, dass dieser Teil etwas umständlich ist, beharrt aber darauf, dass es keinen anderen Weg gibt. Sein Partner bezweifelt das. P1 versucht seinen Partner mehrere Minuten lang zu überzeugen. Er bricht die Erklärung schließlich ab, und schlägt vor das Thema zu verschieben:

  > *P1*: „Hm, warte, lass uns das wann anders klären, das gehört glaub ich jetzt nicht hier hin.“
  > *P2*: „Ja, hmm, wo wollen wirs aufnehmen?“
  > *P1*: „Emm, das (.) dass ich dir das erkläre? Das, häh, ich weiß nicht, wo da der richtige Platz für ist. [...] Das könnte ich mal ins Wiki mit aufnehmen“

  Das Thema passte in die Orientierungsphase und war relativ wichtig, die Diskussion dauerte aber schon recht lange und es machte den Eindruck, als könnte sie auch noch mehrere Minuten lang weiter gehen. Der Zeitaufwand scheint hier der ausschlaggebende Grund für das Verschieben des Themas.

  Im weiteren Verlauf dieser und der nächsten Sitzungen kann nicht beobachtet werden, dass P1 das Problem tatsächlich in der Wiki erklärt.

- Später in der Sitzung gibt es eine Situation, in der das Paar schon längere Zeit über ein Thema spricht, P2 weißt explizit darauf hin, dass das Thema gleich besprochen werden sollte und erklärt warum.

  > *P2*: „Das, das wäre schon ne Sache über die wir uns jetzt unterhalten können, weil das wird ja, sich eventuell darauf auswirken wie wir das jetzt implementieren. “

6.4 Kategorie: Designdiskussion

Die Phase *Weg zum Ziel* (näheres: Sektion 5.2.2) ist hauptsächlich von Diskussionen zum Design der zu erstellen Softwareartefakte geprägt. Partner bringen Designvorschläge, nehmen sie an oder lehnen sie ab und begründen möglicherweise ihren Standpunkt und ihre Vorschläge. In die Kategorie *Designdiskussion* fallen Episoden, in denen das Paar ein Design vorschlägt oder ablehnt und den Vorschlag oder die Ablehnung möglicherweise begründet.

Hin und wieder ist der Ablauf der Designdiskussionen oder die Art der Designvorschläge ein guter Indikator für Probleme früher in der Orientierungsphase. Beispielsweise neigen Partner, welche die vorliegende Problemdomäne sehr gut verstanden haben, dazu sinnvolle Designvorschläge zu bringen.
In [13, S. 59ff] beschreiben `Salinger & Prechelt` das Konzept *Design*, eine kurze Zusammenfassung ist in Sektion 3.3 zu finden. In diesem Abschnitt beschreibe ich weitere Dimensionen von Designdiskussionen - jene, die für die Orientierungsphase von Interesse sind:

1. **Sinnhaftigkeit**: Aus der Sicht eines allwissenden Beobachters, ist ein Vorschlag oder dessen Ablehnung sinnvoll?
2. **Begründung**: Wird ein Vorschlag, bzw. dessen Ablehnung begründet?

Hierbei ist zu beachten, dass ein Beobachter, der Sitzungen untersucht, selten ein allwissender Beobachter ist. Viele Informationen fehlen oder sind schwierig zu interpretieren, sodass es in einigen Situationen schwierig einzuschätzen ist, wie sinnvoll ein Designvorschlag ist.
Es wurden keine Instanzen von begründeten unsinnigen Designvorschlägen beobachtet.

sinnvolle, unbegründete Designvorschläge und Ablehnungen

Beschreibung

Die meisten beobachteten Instanzen von *Designdiskussionen* waren sinnvolle Vorschläge ohne Begründung oder nähere Erklärung. In vielen Fällen scheint der Partner das Design nachvollziehen zu können, sodass es keiner weiteren Erklärung bedarf. Sollte der Partner das Design nicht nachvollziehen können, wird eine Erklärung gewöhnlich nachgeliefert - näheres dazu im nächsten Abschnitt 6.4.
Einer unbegründeten Ablehnung folgt oft ein Gegenvorschlag, mit dem eine neue *Designdiskussions*-Episode beginnt. Die Begründung für die Ablehnung wird dadurch oft implizit geliefert.

ursächliche Bedingung

Einer der Partner hat entweder einen Einfall hinsichtlich des Produktdesigns oder hat sich schon vorher Gedanken über das Design gemacht, und möchte diese seinem Partner mitteilen. Mit Einfall ist hier das gemeint, was in [13] mit dem Konzept *finding* beschrieben ist:

„Ein Einfall ist das Resultat eines Gedankengangs aus einer neuen Erkenntnis. *Neu* heißt dabei während der aktuellen Sitzung und häufig innerhalb der vorangegangenen Minute. Eine Erkenntnis beschreibt neues Wissen - im Unterschied zur Erinnerung.“ [13, S. 113]

intervenierende Bedingungen und Kontext

Wenn der Sprecher keine Erklärung zu seinem Vorschlag bereit hat, wird er dazu neigen, gar nicht erst zu versuchen, seinen Vorschlag zu erklären.

Konsequenzen

Im Idealfall verstehen beide Partner das besprochene Design und können sich schnell darauf einigen.
Im schlechtesten Fall kann der Mangel an Begründung und Erklärung zu Missverständnissen führen.

Beispiele

- Die Partner in der Sitzung KA7 sollen ein neues Feature implementieren. Einer der beiden hat sich zuvor kurz angeschaut wo im Code das am bestem zu einzufügen ist. Sein Partner macht recht schnell einen Vorschlag:

  > *P2*: „Ich hätt jetz hier nen `FeatureSwitch` eingebaut, dass man einfach zwischen Neuem und Altem (∼Formular) per `FeatureSwitch` umschaltet“

  Sein Partner ist mit dem Vorschlag einverstanden, es gibt also keinen Grund für weitere Ausführungen.

- In der Sitzung CA4 entschließt sich das Paar für vorhandenen Code einen Test zu schreiben. Einer der Entwickler schlägt vor einen `JUnit4`-Test zu schreiben. Dem Partner sind die Vorteile dessen vermutlich schon klar, sodass der Vorschlag keiner Begründung bedarf.

sinnvolle, begründete Designvorschläge und Ablehnungen

Beschreibung

In einigen Situationen liefern die Entwickler weitere Erklärungen oder Begründungen für ihre Designvorschläge oder - Ablehnungen. Häufig nachdem es irgendeinen Indikator gab, der vermuten lässt, dass ihr Partner dem Design nicht folgen kann oder anderer Meinung ist.

behandeltes Phänomen

Einer der Entwickler hat einen Einfall zum Design und er vermutet, dass sein Partner seinen Einfall nicht ohne weitere Erklärung nachvollziehen kann. Die Erklärung wird in einigen Situationen eher als Nebenprodukt eines „lauten Denkens“ geliefert.

intervenierende Bedingungen und Kontext

In den meisten beobachteten Fällen gab es kurz vorher in der Sitzung irgendein Zeichen, das darauf hinweist, dass weitere Erklärungen zum Design nötig sind.

Konsequenzen

Das Vorhandensein einer Erklärung und Begründung hilft bei der Einigung auf ein Design - solang die Begründungen nachvollziehbar sind.

Beispiele

- In der Sitzung CA1 gibt es eine längere Episode, in der das Paar diskutiert, ob ein Bedienelement als Button oder Checkbox dargestellt werden sollte. Schließlich wirft einer der Entwickler ein:

  > „Dann wär doch ne Möglichkeit, das wir das <* die Checkbox *> hinterher auch hier einbauen“

  Sein Partner ist nicht überzeugt, daher begründet er den Vorschlag später mit:

  > „Wieso an einer Stelle so, und an der anderen anders? Wär jetz meine Frage. (.) Also entweder ist es sinnvoller mit Button, dann an beiden Stellen mit Button, oder es ist sinnvoller mit Checkbox, dann an beiden Stellen die Checkbox.“

  Sein Partner kann das nachvollziehen und sie einigen sich darauf, später auch an dieser Stelle eine Checkbox einzubauen.

- In der Sitzung ZB2 weist der Entwickler P1 seinen Partner auf ein Problem im Code hin. Er schlägt vor, an einer Stelle im Code ein `Objekt` anstelle eines festen `String` zu verwenden und gibt unaufgefordert eine Begründung dazu.

  > *P1*: „Jetz is aber das Ding, dass wir das vielleicht doch so machen sollten, dass das als Objekt, also diese Queue übergeben wird und nicht fest eingetragen, weil ich mein' wir müssen ja Queues und Topics verwenden.“

  Sein Partner kann die Begründung nachvollziehen, weist ihn aber darauf hin, dass die Annahme, sie müssten Queues und Topics verwenden falsch ist.

unsinnige, unbegründete Designvorschläge und Ablehnungen

Beschreibung

Unsinnige Designvorschläge sind solche, die aus der Sicht eines allwissenden Beobachters nicht sinnvoll sind. Der Entwickler, der den Vorschlag liefert wird ihn sicherlich für sinnvoll halten. Wenn ihm wichtige Informationen fehlen oder er Informationen misinterpretiert, kann sein Vorschlag aus einem anderen Blickwinkel allerdings sinnlos scheinen.
Wie angedeutet, sind solche weniger sinnvollen Vorschläge und Ablehnung ein Indikator für ein Fehlen von Wissen oder Missverständnissen und damit oft ein Zeichen, dass in der Phase *Stand des Systems* (siehe 5.2.1) wichtiges Wissen nicht transferiert oder verstanden wurde.

intervenierende Bedingungen und Kontext

Wie erwähnt, ist ein Grund für die Unsinnigkeit eines Vorschlags ein Fehlen von Wissen. Dieses Fehlen fällt im paradigmatischen Modell zu den intervenierenden Bedingungen, da es die Handlungsmöglichkeiten einschränkt.

Konsequenzen

Unsinnige Vorschläge und Ablehnungen bieten die Möglichkeit, Wissenslücken zu erkennen und zu schließen. Zum Problem wird das Ganze nur, wenn beide Partner schlechtes Design für gut erachten und umsetzen wollen. Das kam in den untersuchten Sitzungen aber nie vor.

Beispiel

In der Sitzung CA2 hat einer der Entwickler eine wichtige Anforderung an das System nicht vor Augen. Seine Vorschläge zum Design der Lösung sind daher objektiv inadäquat. Seinem Partner fällt das auf und er versucht ihm die Anforderung klar zu machen, es gelingt ihm aber nicht. Erst deutlich später fällt dem Entwickler selbst auf, dass ihm die Anforderung entgangen ist, und sein Vorschlag daher überdacht werden sollte.

6.5 Ende der Orientierungsphase

Wie eingangs beschrieben, ist eine Orientierungsphase eine Phase, in der das Paar nicht weiß, wie es ein vorliegendes Problem lösen will. Sobald also einer der Partner ein gute Vorstellung davon hat, wird eine Orientierungsphase beendet. Meistens stellt dieser sicher, dass auch sein Partner eine ähnlich gute Vorstellung von der Problemlösung hat.
In einigen der untersuchten Sitzungen fiel aber auf, dass entweder nur einer der Partner eine gute Vorstellung von der Lösung hat, oder dass eine vermeintlich gute Vorstellung viele Lücken besaß. Beide Szenarien führen zu unterschiedlichen Ausprägungen des Endes einer Orientierungsphase.

In einigen Sitzungen ist das Ende zwischen der Orientierungsphase und dem Rest der Sitzung sehr verschwommen, sodass die Einteilung schwierig fällt.

klares Ende

Beschreibung

Im einfachsten Fall haben beide Partner eine gute Vorstellung von den Schritten, die zur Lösung des vorliegenden Problems nötig sind und wollen beide anfangen die Lösung umzusetzen. Das führt dazu, dass es einen eindeutigen Punkt gibt, an dem das Paar aufhört über potentielle Lösungen zu reden, und anfängt eine umzusetzen.

behandeltes Phänomen

Zu einem klaren Ende der Orientierungsphase kommt es, wenn beide Partner einen Plan für den Rest der Sitzung haben und zuversichtlich sind, dass dieser Plan gut ist.

intervenierende Bedingungen und Kontext

Einschränkend wirkt die Situation, wenn der Partner eines Entwicklers eine andere oder keine Vorstellung vom restlichen Verlauf der Sitzung hat. Dieser Zustand muss entweder beseitigt werden, oder es kommt zu einem *einseitigen Ende*, welche im nächsten Abschnitt 6.5 näher besprochen wird.

Konsequenzen

Das Ende der Orientierungsphase kann potentiell voreilig eingeleitet werden. Das passiert, wenn sich ein vermeintlich guter Plan als schlecht erweist. Dieses Szenario wurde in den untersuchen Situationen allerdings nie beobachtet. Ein klares Ende war im untersuchten Material ein Zeichen dafür, dass beide Partner das Design und die weitere Strategie gut verstanden haben und beginnen wollen, eine Lösung zu implementieren. Orientierungsphasen mit klarem Ende führten tendenziell zu guten Sitzungen.

Beispiele

- Kurz vor dem Ende der Orientierungsphase zu Beginn der Sitzung CA4 fällt dem Paar auf, dass der Code, den es modifizieren soll, ungetestet ist. Die Partner entscheiden sich, erst einmal einen Test für den Code zu schreiben und beginnen anschließend eben damit.
- In der Sitzung CA3 verschafft sich das Paar zusammen in den ersten Minuten einen Überblick über ihre Problemdomäne. Sobald es ein solides Verständnis davon hat, was getan werden muss, um das

Sitzungsziel zu erreichen, beginnen die Entwickler Tests für die neue Funktionalität zu schreiben.

einseitiges Ende

Beschreibung

Zu einem *einseitigen Ende* kommt es, wenn nur einer der Partner die Orientierungsphase beenden will. Das erkennt man zum Beispiel daran, dass dieser die Tastatur nimmt und anfängt Code zu schreiben, während sein Partner noch Fragen und Anmerkungen hat, die von der Art in die Orientierungsphase passen.

behandeltes Phänomen

Nur einer der Partner möchte die Orientierungsphase beenden, und dass ohne auf den anderen zu warten.

intervenierende Bedingungen und Kontext

Aus irgendeinem Grund nimmt sich derjenige, der die Orientierungsphase abschließen will nicht die Zeit, seinen Partner auf den gleichen Stand zu holen. Das kann mehrere Gründe haben:

1. Er realisiert nicht, dass sein Partner noch Anmerkungen hat, oder ihm Informationen fehlen
2. Er nimmt an, dass das nicht zu einem Problem führt, und sein Partner vielleicht sogar während des Implementierens auf den gleichen Stand kommt.

Konsequenzen

In den beobachteten Sitzungen führt ein *einseitiges Ende* dazu, dass für längere Zeit beide Partner unterschiedliche Ziele verfolgen. Solang das der Fall ist, helfen sie einander nicht und neigen sogar dazu, sich gegenseitig gehäuft zu unterbrechen.

Beispiel

In der Sitzung CA2 hat der Entwickler P2 einen Lösungsweg, den er gut versteht und der ihm sinnvoll erscheint vor Augen. Sein Partner hat zuvor schon begonnen, einen anderen Lösungsweg zu implementieren, den P2 für unnötig komplex hält. P2 hat die Tastatur und beginnt seinen Vorschlag umzusetzen, während sein Partner noch zu besprechende Themen hat.

verworrenes Ende

Beschreibung

Das Paar beginnt eine Lösung zu implementieren, dabei fällt allerdings schnell auf, dass wichtige Probleme nicht besprochen wurden und es fällt zurück in eine Orientierungsphase. Dieses Muster wiederholt sich dann potentiell mehrfach.

intervenierende Bedingungen und Kontext

Die Tatsache, dass das Paar versäumt hat, etwas für die Umsetzung ihres Plan wichtiges zu besprechen gehört hier zu den intervenierenden Bedingungen.

Konsequenzen

Das Paar kann auf verschiedene Weisen mit einer solchen Situation umgehen. In den untersuchten Sitzungen haben die Paare versucht eine Lücke im Plan sofort kurz zu besprechen, um möglichst schnell weiter zu implementieren. Das führte häufig dazu, dass diese Lücke unzureichend besprochen wurde und kurz darauf weitere Lücken auftreten, was zu dem typischen Schema des *verworrenen Endes* führt - ein wiederholtes Beenden und Wiederaufnehmen einer Orientierungsphase.

Beispiele

- In der Sitzung CA2 bespricht das Paar verschiedene Abwägungen zum Design ihrer Lösung. P1 hat schon zuvor an dem Problem gearbeitet. Während der Unterhaltung nimmt sich P2 die Tastatur und beginnt die Richtung der Unterhaltung vorzugeben. Wie vom *Schüler* zu erwarten, schaut er sich Code an und stellt seinem Partner dazu Fragen und sammelt so wichtige Informationen. Ohne klare Strategie - er hat eine Vorstellung vom nächsten Arbeitsschritt, aber wenig darüber hinaus - beginnt P2 Code zu modifizieren, dabei fällt dem Paar in mehreren Situationen auf, dass noch Fragen ungeklärt sind. Sodass das Paar abwechselnd Code produziert und neu erkannte Probleme diskutiert.

- Das Ende der ersten Orientierungsphase in KA7 läuft ähnlich ab. Das Paar verschafft sich einen Überblick über die Problemdomäne und einigt sich dann auf einen Arbeitsschritt. Um diesen zu vervollständigen sammelt es noch einige weitere Informationen. Es modifiziert ein wenig Code und checkt die Änderung in die Versionsverwaltung ein. Das Ganze dauert ungefähr 7 Minuten. Die Partner beginnen dann erneut sich zu orientieren und besprechen die weitere Strategie.

Teil III
Schluss

7 Bewertung

Was wurde in dieser Arbeit geleistet?

In dieser Arbeit werden Orientierungsphasen während der Paarprogrammierung eingegrenzt und deren Sinn herausgestellt. Orientierungsphasen werden weiter in Subphasen unterteilt, die sich in der Art der geführten Gespräche unterscheiden.
Des Weiteren wird ein Rollenkonzept vorgestellt und verschiedene Rollen charakterisiert. In der Arbeit wird beschrieben, wie diese Rollen zusammenhängen und welche Konsequenzen aus dem Nichtanerkennen entstehen.
Das Ganze wird weiter unterstützt durch eine umfassende Beschreibung von Verhaltensmustern während der Orientierungsphasen, in denen zusätzlich hypothetische und beobachtete Konsequenzen aus dem Verhalten aufgezeigt werden.
Die beschriebenen Konsequenzen sind meist eindeutig positiv oder negativ einzuordnen, sodass Praktiker der Paarprogrammierung mithilfe dieser Arbeit besser in der Lage sein sollten, problematische Muster zu erkennen und zu vermeiden. Der Blick auf das Wichtige während der Orientierungsphasen sollte gestärkt sein.
Zusammenfassend lässt sich sagen, dass das Paar am Ende eine Orientierungsphase im Idealfall eine gute Vorstellung davon haben sollte, wie es das Sitzungsproblem lösen möchte und welche Schritte für diese Lösung nötig sind. Für effiziente Orientierungsphasen ist es hilfreich, wenn Themen gut strukturiert besprochen werden. Sinnvolle Unterbrechungen sollten behandelt werden ohne sich dabei in Details zu verlieren und zu weit vom ursprünglichen Thema abzukommen.

8 Vergleiche mit vorhandener Literatur

Verhältnis zur Basisschicht

Die beschriebenen Verhaltensmuster sind größtenteils Erweiterungen der Basisschicht. Es werden schon beschriebene Phänomene aus einem anderen Blickwinkel betrachtet und im Rahmen von Episoden analysiert.
In der Basisschicht werden keine Konzepte zu Rollen und Phasen vorgestellt, aber gerade die verschiedenen Rollen und der Umgang damit scheinen sehr nützlich zum Verständnis von Orientierungsphasen.

Rollen während der Paarprogrammierung

Die meisten Quellen zur Paarprogrammierung beschreiben die Rollen *driver* und *observer* oder etwas analoges. Darüber hinaus wird gewöhnlich behauptet, dass die Partner auf verschiedenen Abstraktionsebenen über das aktuelle Geschehen nachdenken.
In [14] argumentieren die Autoren, dass diese Unterteilung unrealistisch ist und schlagen ein anderes Modell für Rollen in der Paarprogrammierung vor. Während meiner Untersuchungen ist auch aufgefallen, dass es äußerst hilfreich für die Interpretation verschiedener Verhaltensmuster ist, den Entwicklern Rollen zuzuweisen. Der klassische Aspekt des Tastatur-Bedienens ist dabei eher uninteressant.

9 Ausblick

Während der Untersuchungen des Materials ist ein Entwickler aufgefallen, der hin und wieder auf einer Metaebene über das aktuelle Geschehen spricht. Er hatte selbst anscheinend eine Vorstellung von einem idealen Ablauf und hat regelmäßig über den aktuellen Stand der Sitzung reflektiert. Außerdem hat er regelmäßig gerade Besprochenes und noch zu Besprechendes zusammengefasst.
All das hatte den Effekt, dass die Orientierungsphasen sehr strukturiert und effizient abliefen. Es scheint also äußerst hilfreich, als Entwickler eine Vorstellung von einem guten Paarprogrammierungsprozess zu haben und regelmäßig über die aktuelle Sitzung zu reflektieren. Als solches ist das aber keine sonderlich greifbare Handlungsempfehlung für Paarprogrammierer.

Teil IV
Anhänge

Transkriptionshinweise

Die Transkriptionen halten sich an das in [11, Anhang III] beschriebene Schema.

Notation	**Beschreibung**	**Kommentar**
(.), (..), (...)	Kurze (bis zu einer Sekunde), mittlere (mehr als eine und bis zu zwei Sekunden), bzw. längere Sprechpausen	Die Zeiten sind grob geschätzt.
<* K *>	Ersetzungen und Kommentare durch den Transkriptor	
<** E **>	Ersetzung durch den Transkriptor zur Anonymisierung	
(∼W)	Wahrscheinlicher Wortlaut eines schwer verständlichen Äußerungsteiles	
[...]	Nicht wiedergegebener Teil einer Äußerung	

Tabelle 8: Transkriptionshinweise. Aus [11]

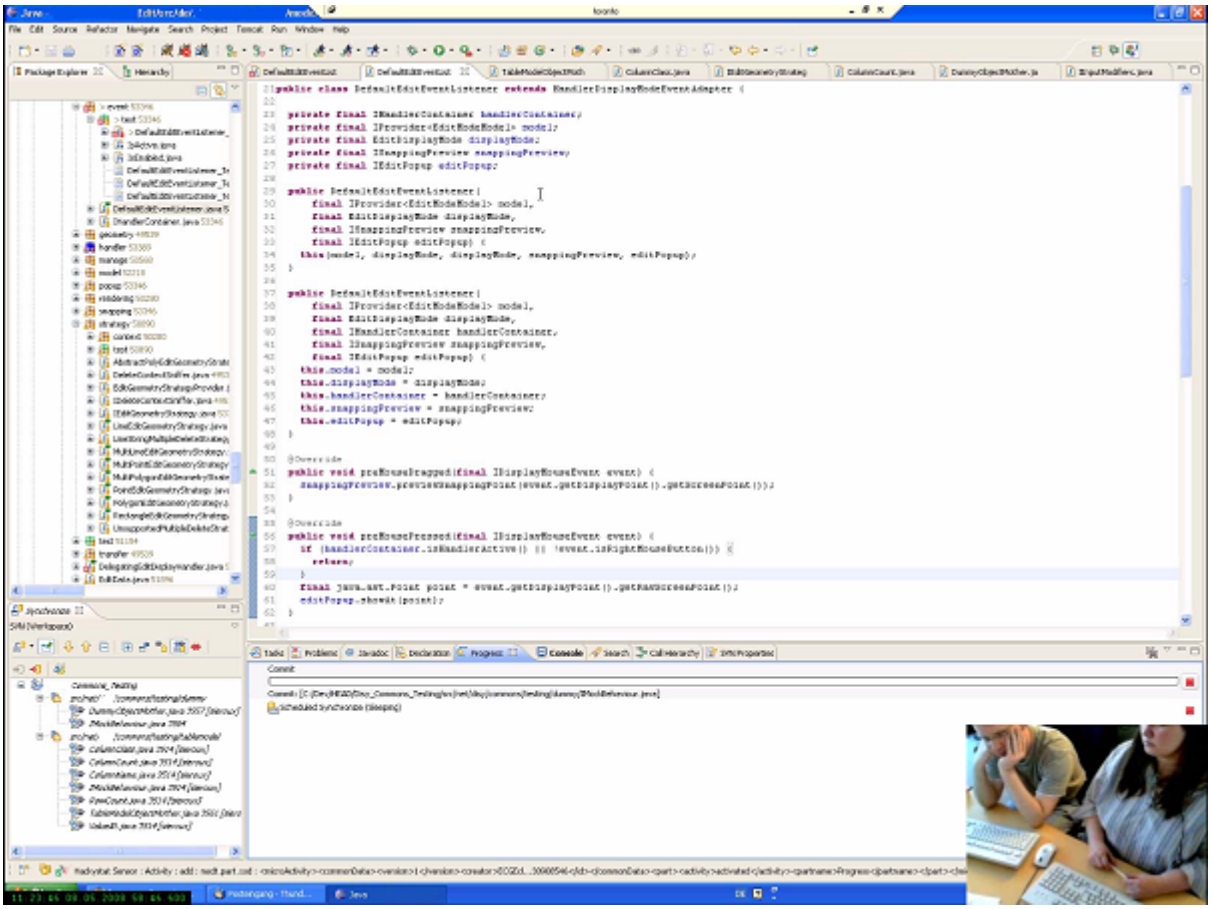

Abbildung 5: Screenshot aus der Sitzung CA4

Literatur

[1] Anselm Strauss im Interview mit Heiner Legewie und Barbara Schervier-Legewie: Forschung ist harte Arbeit, es ist immer ein Stück Leiden damit verbunden. Deshalb muss es auf der anderen Seite Spaß machen. In: *Forum Qualitative Sozialforschung Volume 5, No. 3, Art. 22*, 2004

[2] Baheti, Prashant ; Gehringer, Edward ; Stotts, David: Exploring the Efficacy of Distributed Pair Programming. In: *In: Wells, Don (Hrsg.) ; Williams, Laurie (Hrsg.): Extreme Programming and Agile Methods - XP/Agile Universe 2002 Bd. 2418. Springer Berlin / Heidelberg*, 2002, S. 387–410

[3] Beck, Kent: *Extreme programming explained: embrace change.* Addison-Wesley, 2000

[4] Cockburn, Alistair ; Williams, Laurie A.: The Costs and Benefits of Pair Programming. In: *Proceedings of the First International Conference on Extreme Programming and Flexible Processes in Software Engineering (XP2000)*, 2000

[5] Glaser, Barney M. ; Strauss, Anselm L.: *The Discovery of Grounded Theory: Strategies for Qualitative Research.* Aldine de Gruyter, 1967

[6] Hannay, Jo E. ; Dybå, Tore ; Arisholm, Erik ; Sjøberg, Dag I.: The Effectiveness of Pair Programming: A Meta-Analysis. In: *Information and Software Technology 51 (7)*, 2009, S. 1110 – 1122

[7] Jeffries, Ron E. ; Anderson, Ann ; Hendrickson, Chet: *Extreme Programming Installed.* AddisonWesley Longman Publishing Co., Inc., 2000

[8] McBreen, Pete: *Questioning Extreme Programming.* AddisonWesley Longman Publishing Co., Inc., 2002

[9] Nosek, John T.: The case for collaborative programming. In: *Communications of the ACM 41*, 1998, S. 105–108

[10] Plonka, Laura: A Comparison Between Student and Professional Pair Programmers. In: *Proceedings of the 20th Annual Workshop of Psychology of Programming Interest Group (PPIG '08), Lancaster*, 2008

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

[12] Salinger, Stephan ; Prechelt, Lutz: What happens during Pair Programming. In: *Proceedings of the 20th Annual Workshop of Psychology*

of Programming Interest Group (PPIG '08), Lancaster, September 10-12, 2008., 2008

[13] Salinger, Stephan ; Prechelt, Lutz: *Understanding Pair Programming: The Base Layer*. BoD — Books on Demand, Norderstedt, Germany, 2013

[14] Salinger, Stephan ; Zieris, Franz ; Prechelt, Lutz: Liberating Pair Programming Research from the Oppressive Driver/Observer Regime. In: *Proc. of the 35th International Conference on Software Engineering (ICSE), San Francisco*, 2013, S. 1201–1204

[15] Strauss, Anselm ; Corbin, Juliet: *Grounded Theory: Grundlagen Qualitativer Sozialforschung*. Psychologie Verlags Union, 1996

[16] Upchurch, Richard L. ; Williams, Laurie A.: In Support of Student Pair Programming. In: *Proceedings of the thirty-second SIGCSE technical symposium on Computer Science Education*, 2001, S. 327–331

[17] Vanhanen, Jari: *Effects of Pair Programming at the Development Team Level: An Experiment*, University of Technology, Helsinki, Licentiate Thesis, 2005

[18] Williams, Laurie A.: *The collaborative software process*, The University of Utah, Diss., 2000

[19] Williams, Laurie A.: Integrating Pair Programming into a Software Development Process. In: *14th Conference on Software Engineering Education and Training*, IEEE, 2001, S. 27–36

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

[21] Williams, Laurie A. ; Kessler, Robert R. ; Cunningham, Ward ; Jeffries, Ron: Strengthening the Case for Pair Programming. In: *Software, IEEE*, 2000, S. 19–25

[22] Wilson, Judith D. ; Hoskin, Nathan ; Nosek, John T.: The benefits of collaboration for student programmers. In: *SIGCSE technical symposium on Computer science education*, 1993, S. 160–164

[23] Zieris, Franz: *Qualitative Untersuchung von Beeinflussungen der Aktivität des Partners bei der Paarprogrammierung*, Freie Universität Berlin, Master's Thesis, Februar 2012

[24] Zieris, Franz ; Prechelt, Lutz: On Knowledge Transfer Skill in Pair Programming. In: *Proceedings of the 8th ACM/IEEE International*

Symposium on Empirical Software Engineering and Measurement, Torino, Italy, 2014