Karlsruhe Institute of Technology

Empirische Untersuchung der agilen Softwareentwicklung

zur Erlangung des akademischen Grades eines

Doktors der Ingenieurwissenschaften

der Fakultät für Informatik
des Karlsruher Instituts für Technologie (KIT)

genehmigte

Dissertation

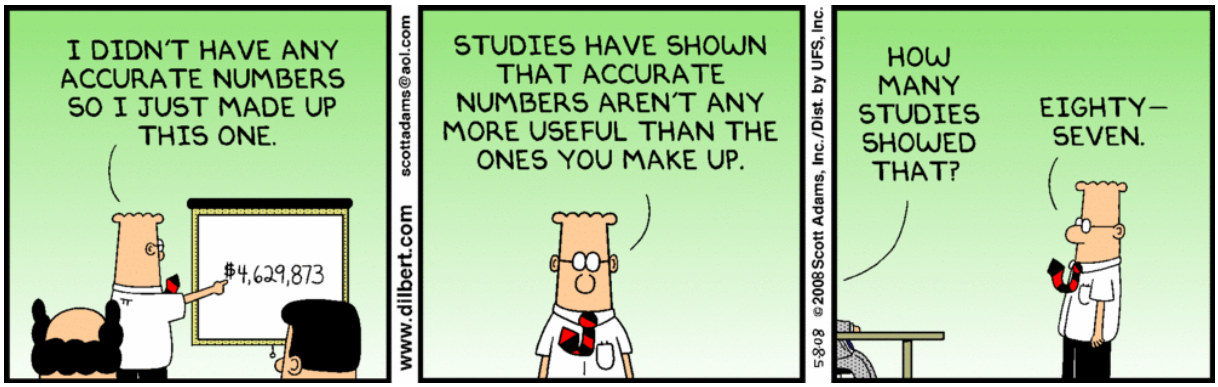
von

Andreas Höfer

aus Bad Säckingen

Tag der mündlichen Prüfung:	18. 10. 2011
Erster Gutachter:	Prof. Dr. Walter F. Tichy
Zweiter Gutachter:	Prof. Dr. Ralf H. Reussner

Inhaltsverzeichnis



Zusammenfassung

Entsprechend ihrer steigenden Popularität seit den 1990er Jahren wurden agile Softwareentwicklungsprozesse in empirischen Studien aufgegriffen. Insbesondere die Entwicklungsmethoden der Paarprogrammierung und der testgetriebenen Entwicklung gerieten früh in den Fokus der Forschung. Dennoch sind zu diesen beiden Entwicklungsmethoden längst nicht alle Fragen beantwortet. In dieser Arbeit werden drei noch zu beantwortende Fragen bezüglich dieser Entwicklungsmethoden identifiziert und es wird mit drei entsprechenden Studien versucht, diese Fragen zu beantworten.

Die erste in dieser Arbeit aufgegriffene Frage ist eine unbeantwortete Forschungsfrage aus einer früheren, an unserem Lehrstuhl durchgeführten Studie, in der wir deutliche Unterschiede zwischen den testgetriebenen Entwicklungsprozessen von erfahrenen und unerfahrenen Einzelprogrammierern nachweisen konnten. Offen war nach dieser Studie, inwieweit die Programmiererfahrung einen Einfluss auf den testgetriebenen Entwicklungsprozess von Programmiererpaaren hat. Um diese Frage zu beantworten, wurden in einem Quasi-Experiment insgesamt drei Gruppen mit jeweils sechs Programmiererpaaren verglichen. Die Gruppe mit viel Programmiererfahrung bestand aus zwölf professionellen Softwareentwicklern. Die anderen beiden Gruppen mit wenig Programmiererfahrung bestanden aus jeweils zwölf Studenten des Extreme Programming-Praktikums der Jahrgänge 2006 und 2008. Alle Programmiererpaare mussten die in Java implementierte Steuerung eines Fahrstuhlsystems ergänzen und sollten dabei die testgetriebene Entwicklung einsetzen. Die Prozesstreue gegenüber der testgetriebenen Entwicklung wurde mit dem an unserem Lehrstuhl entwickelten Werkzeug TestPulse gemessen, welches auch in den folgenden zwei Studien zum Einsatz kam. Bei der Auswertung der Daten zeigte sich, dass die erfahrenen Programmiererpaare die Quelltexte häufiger umstrukturierten und dazu auch häufiger die von der Entwicklungsumgebung angebotenen automatischen Umstrukturierungen verwendeten als die unerfahrenen Programmiererpaare in den anderen beiden Gruppen. Ferner benötigte die Gruppe der erfahrenen Programmiererpaare deutlich mehr Zeit für die Programmieraufgabe als einer der beiden unerfahrenen Gruppen, änderte aber auch deutlich mehr Zeilen im Quelltext.

Die zweite in dieser Arbeit vorgestellte Studie beschäftigt sich mit der Frage, ob Programmiererpaare, in denen die Programmiere unterschiedliche Persönlichkeitsmerkmale haben, effektiver arbeiten als Paare, in denen die Programmierer gleiche Persönlich-

keitsmerkmale haben. Hierzu wurden in einem Experiment mit Studenten der Extreme Programming-Praktika unseres Lehrstuhls und der Universität Bonn zunächst mittels des Keirsey-Temperament-Tests die Persönlichkeitsmerkmale der Teilnehmer ermittelt. Anschließend wurden die Teilnehmer zufällig in Paare mit unterschiedlichen oder gleichen Persönlichkeitsmerkmalen eingeteilt. Insgesamt liegen verwertbare Daten von 15 Programmiererpaaren vor; davon haben die Partner in sieben Paaren jeweils gleiche und in acht Paaren unterschiedliche Persönlichkeitsmerkmale. Die Paare mussten anschließend unter Verwendung der testgetriebenen Entwicklung ein Programm zur Verwaltung von Toto-Fußballwetten erweitern. Bei der Auswertung der Daten zeigte sich, dass die Paare mit unterschiedlichen Persönlichkeitsmerkmalen mehr Anläufe brauchten, um den Akzeptanztest zu bestehen, als die Paare mit gleichen Persönlichkeitsmerkmalen. Gleichzeitig hatten die Paare mit unterschiedlichen Persönlichkeitsmerkmalen die Anweisungsabdeckung der mit dem zu erweiterten Programm ausgelieferten Komponententests stärker erhöht als die Paare mit gleichen Persönlichkeitsmerkmalen.

Die dritte in dieser Arbeit untersuchte Frage ist, ob die testgetriebenen Entwicklung gegenüber der konventionellen Entwicklung nach dem Wasserfallmodell entscheidende Vorteile hat. Dies wurde zwar schon mehrfach untersucht, jedoch sind die bisher in der Literatur veröffentlichten Ergebnisse widersprüchlich. Eine Ursache hierfür könnte sein, dass bisher noch nie die Prozesstreue der Teilnehmer in den Gruppen geprüft wurde. Daher haben wir ein Experiment mit 16 Studenten des Extreme Programming-Praktikums 2010 durchgeführt, in dem die Prozesstreue der Teilnehmer mit TestPulse überprüft wurde. Dadurch ist sichergestellt, dass eventuell beobachtete Unterschiede ihre Ursache wirklich in den unterschiedlichen Prozessen haben. Das Experiment hatte einen gegenbalancierten Entwurf, so dass jeder Teilnehmer zwei Programmieraufgaben zu lösen hatte, eine mit testgetriebener und eine mit konventioneller Entwicklung. Die eine Programmieraufgabe bestand darin, den Soundex-Algorithmus, einen phonetischer Algorithmus zur Indizierung von Wörtern, zu implementieren. In der anderen Aufgabe mussten die Teilnehmer eine 24-Stunden-Videothek um ein neues Preisschema und die Rechnungserstellung erweitern. Ein Unterschied zwischen den Entwicklungsprozessen zeigte sich bei Prozesstreue gegenüber der testgetriebenen Entwicklung. Dies war aber angesichts der geforderten Entwicklungsprozesse eine Voraussetzung für die Validität der Studie. Ebenso gab es Unterschiede bei den Schwankungen des zeitlichen Abstands der Testläufe und beim Anteil der geänderten Zeilen im Testcode, die aber ebenso lediglich bestätigen, dass die Teilnehmer sich im Wesentlichen an die von ihnen geforderten Entwicklungsprozesse gehalten haben. Auswirkungen der unterschiedlichen Entwicklungsprozesse auf Produktivität und Qualität ließen sich dagegen nicht nachweisen.

Kapitel 1

Einleitung

Seit den Anfängen der agilen Bewegung in den 1990er Jahren haben agile Softwareentwicklungsprozesse stark an Popularität gewonnen. Sie sind heute soweit akzeptiert, dass sie sogar in großen Softwareunternehmen wie Microsoft eingesetzt werden [BN07].

Aufgrund der ihrer wachsenden Verbreitung wurden die Methoden der agilen Softwareprozesse auch in empirischen Studien aufgegriffen. Dies gilt speziell für die Methoden der Paarprogrammierung und der testgetriebene Entwicklung zu denen es bereits eine große Anzahl an empirischen Studien gibt. Dennoch besteht zu beiden Methoden weiterhin Forschungsbedarf: So beruhen, wie die Betrachtung der verwandten Arbeiten in Abschnitt 3.2 zeigen wird, viele Erkenntnisse über die Effektivität der testgetriebene Entwicklung auf der Beobachtung von Studenten oder in der testgetriebenen Entwicklung unerfahrenen professionellen Entwicklern. Im Bemühen, die bisherigen Ergebnisse besser einordnen zu können, wurde in einer an unserem Lehrstuhl durchgeführten Studie [MH07] untersucht, welche Rolle die Erfahrung bei der testgetriebenen Entwicklung spielt. Da im Extreme Programming die testgetriebene Entwicklung zusammen mit der Paarprogrammierung eingesetzt wird, lag für uns die Frage nahe, wie sich die Erfahrung bei testgetriebener Entwicklung in Kombination mit Paarprogrammierung bemerkbar macht. Diese Frage soll in der ersten Studie dieser Arbeit in Kapitel 4 beantwortet werden.

Im Zusammenhang mit der Paarprogrammierung gibt es noch weitere offene Forschungsfragen. Zwar existieren viele Studien, welche die Paar- mit der Einzelprogrammierung vergleichen, so dass wir über deren Effizienz gegenüber der Einzelprogrammierung bereits recht viel wissen. Über den Einfluss menschlicher Faktoren auf die Paarprogrammierung, wie die Persönlichkeitsmerkmale der Programmierer, ist bis jetzt jedoch recht wenig bekannt. Die wenigen Studien die bisher zu diesem Thema existieren, liefern zum Teil widersprüchliche Ergebnisse. Hier soll die zweite Studie dieser Arbeit in Kapitel 5 helfen zu klären, wie sich die Persönlichkeitsmerkmale der Programmierer in der Paarprogrammierung bemerkbar machen.

Weiter wird die Betrachtung der verwandten Arbeiten zeigen, dass bisherige Experi-

mente, welche die testgetriebene Entwicklung mit anderen Entwicklungsmethoden vergleichen die Prozesstreue der Teilnehmer nicht oder nur unzureichend kontrollieren. Daher besteht für alle bisherigen Studien die Gefahr, dass sich die Teilnehmer nicht an den relativ schwer zu erlernenden testgetriebenen Entwicklungsprozess gehalten haben. Die zuvor angeführte mangelnde Erfahrung der Teilnehmer mit der testgetriebenen Entwicklung verstärkt diese Gefahr noch. Aus diesem Grund haben wir eine Studie durchgeführt, die testgetriebene und konventionelle Entwicklung vergleicht, wobei die Prozesstreue der Teilnehmer kontrolliert wurde. Diese wird als letzte Studie dieser Arbeit in Kapitel 6 vorgestellt.

1.1 Ziel der Dissertation und Hypothesen

Die Dissertation soll helfen, die in der Einleitung aufgezeigten Lücken im empirischen Verständnis der testgetriebenen Entwicklung und Paarprogrammierung zu schließen. Hierzu wurden drei Studien durchgeführt, die durch die folgenden allgemein formulierten Forschungshypothesen motiviert sind:

FH_1 Paare aus erfahrenen Entwicklern unterscheiden sich von Paaren aus unerfahren Entwicklern bezüglich ihres testgetriebenen Entwicklungsprozesses, ihrer Produktivität und der Qualität der Programme.

FH_2 Paare, in denen die Programmierer unterschiedliche Persönlichkeitsmerkmale haben, unterscheiden sich von Paaren, in denen die Programmierer gleiche Persönlichkeitsmerkmale haben, bezüglich ihrer Produktivität, der Qualität der abgelieferten Programme und ihres testgetriebenen Entwicklungsprozesses.

FH_3 Der Einsatz der testgetriebenen Entwicklung führt gegenüber der Verwendung der konventionellen Entwicklung nach dem Wasserfallmodell zu Unterschieden bei der Produktivität der Entwickler und der Qualität der Programme.

Kapitel 2

Grundlagen

Dieses Kapitel widmet sich den Grundlagen, die besonders zum Verständnis der Kapitel 4 bis 6 dieser Arbeit nötig sind. Hierzu werden zunächst die statistischen Grundlagen gelegt, bevor in Abschnitt 2.2 erläutert wird, was wir unter testgetriebener Entwicklung verstehen. In Abschnitt 2.3 wird anschließend dargelegt, wie wir testgetriebene Entwicklung mit unserem eigenen Werkzeug TestPulse erkennen und auswerten. Der letzte Abschnitt dieses Kapitels erklärt die mit diesem Werkzeug und manuell erhobenen Metriken.

2.1 Statistische Grundlagen

Dieser Abschnitt erklärt kurz Hypothesentests, Güteanalyse, nichtparametrische Statistik und Boxplots. Alle Leser, die sich mit diesen Themen bereits auskennen, können direkt zu Abschnitt 2.2 springen. Allen Lesern, denen die in diesem Abschnitt gemachten Ausführungen nicht umfangreich genug sind, seien die folgenden Bücher zur Lektüre empfohlen:

- *Fundamental Statistics for the Behavioral Sciences* von David C. Howell [How99] zu allgemeinen statistischen Grundlagen, zur Datenrepräsentation und zum Hypothesentesten;

- *Nonparametric Statistical Methods* von Myles Hollander und Douglas A.Wolfe [HW99] sowie *Nichtparametrische statistische Methoden* von Herbert Büning und Götz Trenkler [BT94] zum Thema nichtparametrische Testverfahren und

- *Statistical Power Analysis for the Behavioral Sciences* von Jacob Cohen [Coh88] zur Güteanalyse.

2.1.1 Hypothesentests

Will man sicher wissen, ob eine Hypothese über die Population wahr ist, müsste man die gesamte Population untersuchen. Dies ist aber oft nicht praktikabel. Deswegen benötigt man Verfahren, um mithilfe von Ergebnissen, die aus einer zufällig gezogenen Stichprobe gewonnen wurden, über den Wahrheitsgehalt einer Hypothese zu entscheiden. Genau dies leisten Hypothesentests. Mit ihnen hat man eine statistische Methode an der Hand, um auf der Grundlage der aus der Stichprobe erhaltenen Ergebnisse zu entscheiden, ob eine Hypothese über die Population als wahrscheinlich angenommen werden kann oder als unwahrscheinlich abzulehnen ist. Aufgrund des beim zufälligen Ziehen auftretenden Stichprobenfehlers kann man bei Entscheidungen, die mithilfe einer Stichprobe getroffen wurden, nicht garantieren, ob die Hypothese in der Population wirklich wahr oder falsch ist. Hypothesentests erlauben es aber zumindest, die Wahrscheinlichkeit für einen Irrtum zu quantifizieren.

2.1.1.1 Null- und Alternativhypothese

Für den Hypothesentest werden zwei Hypothesen benötigt: die Null- und die Alternativhypothese. Mit der Nullhypothese H_0 wird die Annahme ausgedrückt, dass die mit der Stichprobe bzw. bei mehreren Gruppen zwischen den Stichproben gemachte Beobachtung rein dem Zufall geschuldet ist. Die Alternativhypothese H_A formuliert dagegen das, was die empirische Untersuchung zeigen soll. Sie wird daher aus der Forschungsfrage, also dem Gegenstand der Untersuchung abgeleitet. Nehmen wir zum Beispiel an, wir wollen in einer empirischen Studie den Größenunterschied zwischen Männern und Frauen nachweisen. Die Alternativhypothese drückt dann unsere Annahme aus, dass sich die mittlere Größe von Männern und Frauen unterscheidet. In Kurzschreibweise sieht die Alternativhypothese wie folgt aus:

$$H_A : \mu_{Mann} \neq \mu_{Frau} \Rightarrow \mu_{Mann} - \mu_{Frau} \neq 0 \tag{2.1}$$

Die Nullhypothese ist dann die logische Negation der Alternativhypothese, drückt also die Annahme aus, dass die mittlere Größe von Männern und Frauen gleich ist, was kurz geschrieben wie folgt aussieht:

$$H_0 : \mu_{Mann} = \mu_{Frau} \Rightarrow \mu_{Mann} - \mu_{Frau} = 0 \tag{2.2}$$

2.1.1.2 Ein- und zweiseitige Hypothesentests

Im vorigen Abschnitt haben wir zunächst ungerichtete Hypothesen formuliert, d. h. wir erwarten nur einen Unterschied zwischen der mittleren Größe von Männern und Frauen, haben aber keine Vermutung, ob Männer oder Frauen nun größer sind. Eine solche

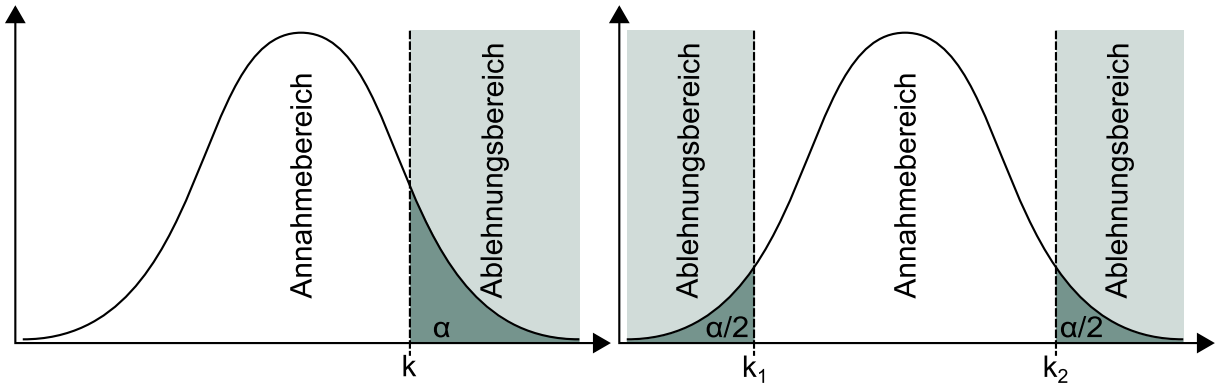


Abbildung 2.1: Gegenüberstellung von ein- und zweiseitigem Hypothesentest.

ungerichtete Hypothese führt zu einem zweiseitigen Hypothesentest, wie er in der rechten Hälfte von Abbildung 2.1 dargestellt ist. Die Kurve stellt dabei eine hypothetische Wahrscheinlichkeitsverteilung für einen Wert unter der Annahme der Nullhypothese dar. In unserem Beispiel wäre es die Wahrscheinlichkeitsverteilung der Differenz der mittleren Größen zwischen Männern und Frauen. An der Stelle der x-Achse, über der sich der Scheitelpunkt der Kurve befindet, müsste folglich die 0 aufgetragen sein. Der Hypothesentest wird als zweiseitig bezeichnet, weil extreme Unterschiede bei den mittleren Größen, egal ob nach links oder rechts, zur Ablehnung der Nullhypothese führen würden.

Um entscheiden zu können, wann man eine Nullhypothese ablehnt und wann nicht, muss man die Ablehnungsbereiche und den Annahmebereich für die unter der Nullhypothese geltende Verteilung definieren. Dies geschieht durch das Festlegen des Signifikanzniveaus α für den Hypothesentest. Dabei gilt, wie man in Abbildung 2.1 rechts sieht, dass die Ablehnungsbereiche des zweiseitigen Hypothesentests an den Stellen beginnen, ab denen die Fläche unter der Verteilungsfunktion in Richtung plus bzw. minus Unendlich jeweils der Größe des halben Signifikanzniveaus entspricht. Diese Stellen werden als kritische Werte bezeichnet und gehören selbst noch zum Ablehnungsbereich. Durch die Wahl des Signifikanzniveaus legt man per Definition auch die obere Schranke dafür fest, dass man die Nullhypothese aufgrund der Werte in der Stichprobe ablehnt, obwohl sie in der unbekannten Wirklichkeit zutrifft. Mithilfe der Teststatistik eines Testverfahren und mit den Daten aus der Stichprobe bzw. den Stichproben wird der *p*-Wert berechnet. Der *p*-Wert gibt die Wahrscheinlichkeit dafür an, ein Ergebnis, wie es mit der Stichprobe bzw. den Stichproben beobachtet wurde oder ein noch extremeres Ergebnis, bei Gültigkeit der Nullhypothese zu erhalten. Ist diese Wahrscheinlichkeit kleiner oder gleich unserem Signifikanzniveau, lehnen wir die Nullhypothese zugunsten der Alternativhypothese ab und sprechen, im Falle unseres Beispiels, von einem statistisch signifikanten Größenunterschied zwischen Männern und Frauen.

Entscheidung	*Unbekannte Wirklichkeit*	
	H_0 ist richtig	*H_0 ist falsch*
H_0 ablehnen	falsche Entscheidung Fehler 1. Art, $w = \alpha$	richtige Entscheidung Güte, $w = 1 - \beta$
H_0 annehmen	richtige Entscheidung –, $w = 1 - \alpha$	falsche Entscheidung Fehler 2. Art, $w = \beta$

Tabelle 2.1: Mögliche Ergebnisse eines Hypothesentests (nach [How99, S. 133, Tabelle 8.1]).

Das eben gesagte gilt auch für gerichtete Hypothesen. Allerdings ergibt sich bei gerichteten Hypothesen ein einziger zusammenhängender Ablehnungsbereich der in seiner Größe dem Signifikanzniveau entspricht. Folglich existiert in diesem Fall auch nur ein kritischer Wert. Gerichtete Hypothesen bieten sich an, wenn man schon eine Idee davon hat, in welche Richtung der vermutete Unterschied ausfallen wird. In unserem Beispiel liegt die Annahme nahe, dass Männer im Durchschnitt größer sind als Frauen. Die Null- und Alternativhypothese wären in Kurzschreibweise dann wie folgt:

$$H_0 : \mu_{Mann} \leq \mu_{Frau} \text{ und } H_A : \mu_{Mann} > \mu_{Frau} \tag{2.3}$$

Hieraus ergäbe sich ein einseitiger, in diesem Fall rechtsseitiger, Hypothesentest, wie er in der linken Hälfte der Abbildung 2.1 zu sehen ist. Würden wir hingegen vermuten, dass Frauen im Mittel größer sind als Männer, würde daraus ein linksseitiger Hypothesentest folgen.

Einseitige Hypothesentests haben gegenüber der zweiseitigen Variante den Vorteil, dass bei sonst gleichen Bedingungen weniger Datenpunkte in der Stichprobe benötigt werden, um signifikante Ergebnisse zu liefern, sofern die Richtung korrekt vorhergesagt wurde. Stimmt die vorhergesagte Richtung jedoch nicht, wird ein möglicher Effekt jedoch nicht aufgedeckt. Da sich die Richtung bei den Hypothesen für die in dieser Arbeit präsentierten Studien nicht vorhersagen lässt, werden in dieser Arbeit ausschließlich zweiseitige Hypothesentests verwendet.

2.1.1.3 Fehler 1. und 2. Art

Wie im Abschnitt zuvor bereits gesagt, gibt das Signifikanzniveau die obere Schranke dafür an, dass die Nullhypothese aufgrund der mit der Stichprobe gemachten Beobachtungen abgelehnt wird, obwohl sie eigentlich wahr ist. Eine solche Fehlentscheidung

nennt man auch Fehler 1. Art oder α-Fehler, da die Wahrscheinlichkeit hierfür üblicherweise mit dem griechischen Buchstaben α angegeben wird.

Wie Tabelle 2.1 zeigt, ist der Fehler 1. Art nicht die einzige Fehlentscheidung, zu der es beim Hypothesentest kommen kann. Es kann ebenso vorkommen, dass die Nullhypothese angenommen wird, obwohl sie in Wirklichkeit falsch ist. Dies wird als Fehler 2. Art oder β-Fehler bezeichnet, da die Wahrscheinlichkeit hierfür für gewöhnlich mit β angegeben wird.

2.1.1.4 Güte und Effektgröße

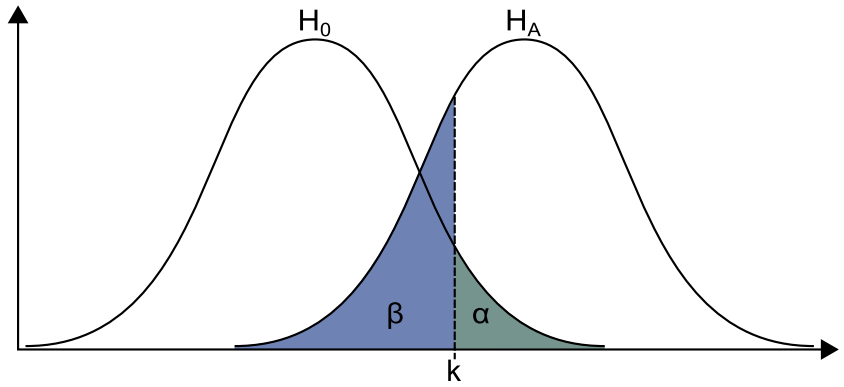


Abbildung 2.2: Darstellung des Zusammenhangs zwischen den Wahrscheinlichkeiten für einen Fehler 1. und 2. Art

Die Fehlerwahrscheinlichkeit für einen Fehler 2. Art wird häufig nicht direkt, sondern indirekt über die Güte angegeben. Die Güte drückt aus, wie groß die Wahrscheinlichkeit ist, in der Realität existierende Unterschiede tatsächlich zu entdecken und gibt somit die Trennschärfe eines Hypothesentests an. Wie aus Tabelle 2.1 ersichtlich, ist die Wahrscheinlichkeit hierfür $1 - \beta$.

Idealerweise möchte man für einen Hypothesentest ein niedriges Signifikanzniveau und eine hohe Güte erreichen. Wie Abbildung 2.2 illustriert, stehen diese beiden Ziele aber in Konkurrenz zueinander. Eine Verkleinerung des Signifikanzniveaus führt zu einer Vergrößerung der Wahrscheinlichkeit für einen Fehler 2. Art und damit zu einer Verschlechterung der Güte. Die Balance zwischen einem geeigneten Signifikanzniveau und niedrige Wahrscheinlichkeit für einen Fehler 2. Art ist daher manchmal schwer zu erreichen.

Außer durch die Wahl des Signifikanzniveaus wird die Güte noch durch zwei weitere

Faktoren beeinflusst: die Effektgröße und die Anzahl der Datenpunkte. Die Effektgröße drückt aus, wie groß der Unterschied zwischen zwei Stichproben ist. Ein gebräuchliches Maß hierfür ist das auf *Cohens d* [Coh88, S. 40 ff.]. Es ist definiert als der Absolutbetrag der Differenz der Mittelwerte der beiden Gruppen geteilt durch die gemeinsame Standardabweichung s der Stichproben. Für die Stichproben gilt demnach die folgende Formel zur Berechnung von Cohens d[1]:

$$d = \frac{|\bar{x}_1 - \bar{x}_2|}{s} \quad \text{mit} \quad s = \sqrt{\frac{(n_1-1)s_1^2 + (n_2-1)s_2^2}{n_1+n_2-2}} \tag{2.4}$$

Cohen [Coh88, S. 24–27] liefert auch eine Orientierung für die Interpretation von Effektgrößen. Nach seinen Ausführungen gelten Werte von 0,2 als kleine, Werte von 0,5 als mittlere und Werte von 0,8 als große Effekte. Die Effektgröße kann man sich auch als Abstand zwischen den Kurven in Abbildung 2.2 vorstellen. Ein größerer Abstand bedeutet eine größere Güte, da die überlappenden Bereiche der Kurven kleiner werden. Für die Auswirkung der Anzahl der Datenpunkte auf die Güte gilt vereinfacht: je mehr Datenpunkte, desto besser wird die Güte. Die Begründung hierzu würde über den Rahmen dieser Arbeit hinausgehen, daher sei an dieser Stelle auf [How99, S. 225 ff.] verweisen, wo der Zusammenhang exakt erläutert wird.

2.1.2 Nichtparametrische Testverfahren

Bisher wurde das Hypothesentesten erklärt, ohne darauf einzugehen, welche konkreten Testverfahren für den Hypothesentest eingesetzt werden sollen bzw. können. Die theoretisch zur Verfügung stehenden Testverfahren lassen sich in zwei große Gruppen einteilen: parametrische und nichtparametrische Verfahren. Parametrische Testverfahren nennen sich so, weil sie bestimmte Annahmen über die in der Population herrschende Verteilung und deren Parameter machen. So verlangt der häufig zum Vergleich zweier Stichproben verwendete t-Test, dass die Stichproben aus einer Population mit Normalverteilung gezogen wurden. Nichtparametrische Verfahren haben dagegen den Vorteil, dass sie keine oder nur schwache Annahmen über die Verteilung der Population machen.

Dies ist aber nicht der einzige Vorteil nichtparametrischer Verfahren im Vergleich mit den parametrischen Varianten. Nichtparametrische Verfahren stellen auch geringere An-

[1] Cohen selbst bezeichnet sein Maß mit d, wenn er von der Effektgröße bei Populationen spricht und mit d_s, wenn er von der Effektgröße bei Stichproben spricht. Folglich müsste es in diesem Fall streng genommen d_s heißen. Sein Maß für Effektgröße ist aber vornehmlich als Cohens d bekannt, weswegen wir diesen Unterschied der Einfachheit halber ausblenden. Cohen gibt auch, je nach Anwendungsfall, mehrere Formeln zur Berechnung der gemeinsame Standardabweichung s der Stichproben vor. Die hier angegebene Formel gilt für zwei unabhängige Stichproben, wobei die Stichproben unterschiedlich groß sein dürfen (vgl. [Coh88, S. 66, 67]).

forderungen an das Messniveau. Während parametrische Verfahren nur mit Intervall- und Verhältnisskalen funktionieren, reichen nichtparametrischen Verfahren Ordinalskalen oder in manchen Fällen sogar Nominalskalen. Zudem sind nichtparametrische Verfahren Ausreißern gegenüber recht robust.

Natürlich gibt es auch Nachteile nichtparametrischer gegenüber den parametrischen Verfahren. Wenn alle Bedingungen für den Einsatz parametrischer Verfahren erfüllt sind, haben diese eine bessere Güte im Vergleich zum nichtparametrischen Äquivalent. Ein weiterer Nachteil ist, dass sich die Güte nichtparametrischer Verfahren zumeist schwer bestimmen lässt. Häufig kann sie sogar nur geschätzt werden.

Bei Abwägung der Vor- und Nachteile nichtparametrischer Verfahren spricht in den in dieser Arbeit vorgestellten Studien alles für den Einsatz nichtparametrischer Verfahren: Die Studien haben alle recht kleine Gruppen und wenige Datenpunkte, weswegen sich keine belastbaren Aussagen über die Verteilungen der Populationen machen lassen. So liefern beispielsweise Tests auf Normalverteilung wie der Shapiro-Wilk-Test bei kleinen Stichproben keine zuverlässigen Ergebnisse. Auch aus anderen vergleichbaren Studien lassen sich die Verteilungen der Populationen nicht ableiten. Daher bietet es sich an, mit nichtparametrischen Verfahren zu arbeiten, da sie keine bestimmte Verteilung voraussetzen.

Zudem ist aufgrund der kleinen Gruppen zu befürchten, dass sich ein einzelner Ausreißer überproportional stark bemerkbar macht, wenn man nicht auf die dagegen relativ robusten nichtparametrischen Testverfahren setzt.

2.1.2.1 Wilcoxon-Test

In allen Studien dieser Arbeit werden Hypothesentests für den Vergleich zweier Gruppen bzw. zweier Stichproben gebraucht. Unter den parametrischen Verfahren wird für diese Aufgabe üblicherweise der t-Test eingesetzt. Das gebräuchliche nichtparametrische Äquivalent hierzu, das in dieser Arbeit zum Vergleich zweier Gruppen zum Einsatz kommt, ist der Wilcoxon-Rangsummentest. Dieser Test ist auch als Mann-Whitney-U-Test[2] bekannt, wird von nun an jedoch nur noch kurz als Wilcoxon-Test bezeichnet.

Der Wilcoxon-Test prüft die Nullhypothese, dass die Stichproben der gleichen Population entstammen gegenüber der Alternativhypothese, dass die Stichproben aus zwei Populationen stammen, deren Verteilungen zwar die gleiche Form haben, jedoch in der Lage auf der x-Achse zueinander verschoben sind. Für die mathematischen Hintergründe und die Teststatistik sei an dieser Stelle auf [BT94, S. 131 ff.] verwiesen. Für das

[2] Wie Büning und Trenkler [BT94, S. 135–136] zu entnehmen ist, handelt es sich bei Wilcoxon-Rangsummentest und Mann-Whitney-U-Test streng genommen nicht um zwei Namen für das gleiche Testverfahren, sondern tatsächlich um zwei verschiedene Testverfahren, die aber nachweislich äquivalent sind.

Verständnis ist nur wichtig, dass das Testverfahren des Wilcoxon-Tests auf Rängen beruht. Daher ist für den Test auch nur ein ordinales Messniveau vorausgesetzt. Zur Durchführung des Wilcoxon-Tests werden die Datenpunkte beider Stichproben zusammengelegt, nach Größe geordnet und dann der Reihenfolge nach Ränge vergeben. In einer der Stichproben wird dann die Summe der Ränge gebildet. Mittels kombinatorischer Überlegungen lässt sich anschließend ermitteln, wie wahrscheinlich es ist, genau diese oder eine noch extremere Rangsumme zu erhalten, wenn die Nullhypothese wahr ist. Diese Wahrscheinlichkeit ergibt dann den p-Wert für den Wilcoxon-Test.

2.1.2.2 Bonferroni-Holm-Korrektur

Bisher sind wir davon ausgegangen, dass nur zwei Gruppen zu vergleichen sind. Die erste Studie dieser Arbeit in Kapitel 4 hat aber drei Gruppen. Damit müssen drei Stichproben miteinander verglichen werden. Für mehr als zwei Stichproben kann über sogenannte globale Tests wie den parametrischen ANOVA-Test oder den nichtparametrischen Kruskall-Wallis-Test geprüft werden, ob alle Stichproben einer Population entstammen oder mindestens eine aus einer anderen Population stammt. Damit weiß man aber noch nicht, welche das ist und wie viele der Stichproben genau aus verschiedenen Populationen stammen. Deswegen führen wir diesen globalen Test nicht durch und vergleichen vielmehr alle möglichen Paarungen[3] der Stichproben. Damit führen wir allerdings zwangsläufig mehrere einzelne Hypothesentests durch, um die globale Hypothese, dass eine der Stichproben zu einer anderen Population gehört, zu prüfen. Nun drückt das Signifikanzniveau definitionsgemäß die Wahrscheinlichkeit aus, die Nullhypothese abzulehnen, obwohl sie für die Population zutrifft. Wenn man aber mehrere Hypothesentests durchführen muss, um eine globale Hypothese zu prüfen, geht man dabei auch mehrfach das Risiko ein, die Nullhypothese fälschlicherweise abzulehnen. Es kommt folglich zur Kumulierung des Fehlers 1. Art. Daher braucht man ein Verfahren, um diesem Phänomen entgegenzuwirken. Ein solches Verfahren ist die Bonferroni-Holm-Korrektur [HDC88]. Hierbei werden die p-Werte aller paarweisen Hypothesentests aufsteigend geordnet. Anschließend werden sie, angefangen beim kleinsten p-Wert, mit einer der beiden folgenden Ungleichungen mit dem Signifikanzniveau verglichen:

$$p_i \leq \frac{\alpha}{n-i+1} \Rightarrow p_i(n-i+1) \leq \alpha \quad \text{mit} \quad i = 1,\ldots,n \quad \text{und} \quad p_1 \leq \cdots \leq p_n \tag{2.5}$$

Bei der ersten Variante, der Bonferroni-Holm-Korrektur des Signifikanzniveaus, wird das Signifikanzniveau α in Abhängigkeit des Ranges i des jeweiligen p-Werts durch Division verringert. Bei der zweiten Variante, der Bonferroni-Holm-Korrektur der p-Werte,

[3] Die Anzahl der möglichen Paarungen bei n Gruppen ist $\frac{n(n-1)}{2}$.

wird der p-Wert seinem Rang entsprechend durch Multiplikation erhöht. Für beide Varianten gilt: Erfüllt ein p-Wert die Ungleichung, ist das Ergebnis des dazugehörigen Hypothesentests signifikant, erfüllt ein p-Wert die Ungleichung nicht, gilt der dazugehörige Hypothesentest und alle eventuell noch folgenden, zu p-Werten mit höheren Rängen gehörigen, Hypothesentests als nicht signifikant. Wir werden in Kapitel 4 unserer Arbeit die Bonferroni-Holm-Korrektur der p-Werte verwenden, da wir ansonsten zu jedem p-Wert das entsprechend korrigierte Signifikanzniveau angeben müssten, was den Text und Tabellen schlechter lesbar machen würde.

2.1.2.3 Güteanalyse und Asymptotische Relative Effizienz

Wie bereits zuvor gesagt, ist die Bestimmung der Güte, die sogenannte Güteanalyse, bei vielen nichtparametrischen Verfahren schwierig. Dies ist auch beim Wilcoxon-Test nicht anders. Eine Möglichkeit die Güte des Wilcoxon-Tests nach unten abzuschätzen, eröffnet die sogenannte Asymptotische Relative Effizienz, kurz ARE [BT94, S. 279 ff.]. Die ARE gibt für zwei zu vergleichende Testverfahren und eine bestimmte für die Population angenommene Verteilung das Verhältnis der Gruppengrößen an, welche die gleiche Güte liefern, wenn die anderen Parameter zur Berechnung der Güte, also das Signifikanzniveau und die Effektgröße, für beide Testverfahren gleich sind.

Im ungünstigsten Fall hat der Wilcoxon-Test gegenüber dem t-Test eine ARE von 0,864. Das bedeutet, dass man für den Wilcoxon-Test maximal $\frac{1}{0{,}864}$ Mal mehr Datenpunkte pro Gruppe als für den t-Test benötigt, um mit beiden Testverfahren die gleiche Güte zu erhalten. Bei gleicher Gruppengröße erhält man wiederum mit dem Wilcoxon-Test mindestens eine Güte, als hätte man einen t-Test mit 0,864 Mal weniger Datenpunkten eingesetzt.

Wie bereits erwähnt, haben wir uns unter anderem gegen parametrische Testverfahren entschieden, weil wir uns nicht sicher sein können, ob die Normalverteilung die korrekte Verteilungsfunktion für die Population ist. Diese Entscheidung wurde auch durch die Tatsache erleichtert, dass die ARE des Wilcoxon-Tests gegenüber dem t-Test bei einer normalverteilten Population 0,955 beträgt, d. h. der Wilcoxon-Test ist in diesem Fall nur geringfügig weniger effizient. Weiterhin gibt es Verteilungen der Population, für welche die ARE größer 1 ist, was bedeutet, dass der Wilcoxon-Test in diesen Fällen effizienter arbeitet als der t-Test [BT94, S. 285].

2.1.3 Grafische Darstellungen von statistischen Daten

Neben intuitiv verständlichen Darstellungsformen, wie Histogrammen und Balkendiagrammen werden in dieser Arbeit häufig *Boxplots* (vgl. [How99, S. 74]) zu Visualisie-

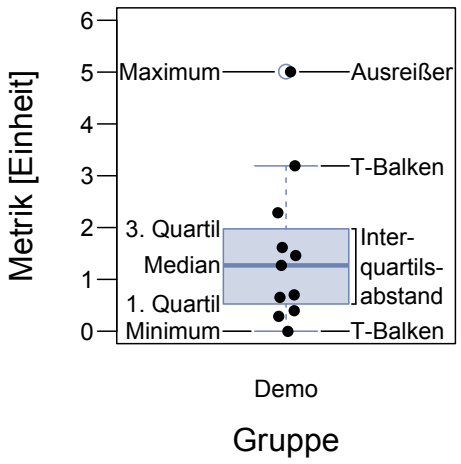


Abbildung 2.3: Beispiel für einen Boxplot.

rung von Daten verwendet. Da diese Form der Visualisierung nicht selbsterklärend ist, soll sie an dieser Stelle erläutert werden.

Boxplots haben ihren Namen von der Box, die sich, wie es in Abbildung 2.3 ersichtlich ist, vom ersten bis zum dritten Quartil erstreckt. Das erste und das dritte Quartil trennen die zentralen 50 Prozent der Datenpunkte von den unteren bzw. oberen 25 Prozent der Datenpunkte. Der Abstand zwischen dem ersten und dem dritten Quartil wird auch als Interquartilsabstand bezeichnet. Der Balken innerhalb der Box kennzeichnet den Median, also den Wert, der die Menge der Datenpunkte in zwei gleich große Hälften teilt. Manchmal werden Boxplots auch als *Box-and-Whisker-Plots* bezeichnet. Mit *whisker*, dem englischen Begriff für Schnurr- oder Barthaar, sind dabei die beiden T-Balken gemeint. Sie verbinden die untere bzw. obere Grenze der Box mit dem jeweils extremsten Datenpunkt in gleicher Richtung, der nicht weiter als eineinhalbfachen Interquartilsabstand von der Box entfernt ist[4]. Liegen Datenpunkte weiter als eineinhalb Mal den Interquartilsabstand von der Box entfernt, wie es in Abbildung 2.3 für das Maximum der Fall ist, dann werden diese so genannten Ausreißer durch einen Ring in der Farbe der Box gekennzeichnet.

Boxplots wurden ursprünglich dazu entworfen, viele Datenpunkte vereinfacht darzustellen und gleichzeitig eine Vorstellung von deren Verteilung zu geben. Bei wenigen Datenpunkten, wie in unseren Studien der Fall, bietet es sich jedoch an, die Lage der Datenpunkte auch direkt anzugeben. Deswegen werden in dieser Arbeit die Datenpunkte immer als schwarze Kreise über den Boxplot gezeichnet.

[4]Diese Definition für die maximale Länge der T-Balken ist üblich und entspricht der Standardeinstellung für Boxplots in der Statistiksoftware R [R]. Allerdings gibt es keine feste Vorschrift für die Bedeutung der T-Balken. Sie können über beliebige Vielfache des Interquartilsabstands definiert sein oder beliebige Quantile, wie das 10 %- und 90 %-Quantil repräsentieren. In seltenen Fällen werden die T-Balken stets bis zum Minimum und Maximum gezeichnet, wodurch die Darstellung von Ausreißern entfällt.

2.2 Testgetriebene Entwicklung

In allen unseren Studien wurde die testgetriebene Entwicklung eingesetzt. Bevor wir erklären können, wie wir diese untersuchen, müssen wir zuerst definieren was wir unter testgetriebener Entwicklung verstehen. Diese Definition ist inhaltlich an den Abschnitt zum gleichen Thema aus der an unserem Lehrstuhl unter Mitwirkung des Autors dieser Arbeit erstellten Studie [MH07] angelehnt.

Testgetriebene Entwicklung oder auch Test-Zuerst-Entwicklung bezeichnet eine agile Entwicklungsmethode, deren zentraler Ansatz die Wiederholung sehr kurzer Entwicklungszyklen ist, in denen die Tests stets vor dem zu testenden Teil der Anwendung geschrieben werden. Die Tests haben bei der testgetriebenen Entwicklung neben der Funktion der Qualitätssicherung auch die Aufgabe, das gewünschte Verhalten der Anwendung zu spezifizieren, stellen also eine Art ausführbare Spezifikation dar. Dieser Ansatz unterscheidet sich von konventionellen, plangetriebenen Prozessen bei denen Tests in der Regel nach der Entwicklung der Anwendung (oder zumindest nach der Fertigstellung eines Teils der Anwendung) in einer eigenen Testphase erstellt werden und nur der Qualitätssicherung dienen.

Testgetriebene Entwicklung kann als eigenständige Entwicklungsmethode eingesetzt werden, wird aber häufig im Kontext von Extreme Programming eingesetzt, bei dem die testgetriebene Entwicklung zu den zentralen Praktiken [Bec05, S. 50] zählt. Der wachsenden Aufmerksamkeit für agile Softwareprozesse im Allgemeinen und speziell für das Extreme Programming hat die testgetriebene Entwicklung auch ihren Durchbruch zu verdanken, obwohl die Idee zu dieser Entwicklungsmethode bereits zuvor existierte.

Wie funktioniert aber nun die testgetriebene Entwicklung? Beck definiert die testgetriebene Entwicklung als die Einhaltung der folgenden zwei Regeln:

- „Write new code only if an automated test has failed.“ [Bec02, S. ix] – Schreibe nur neuen Code, falls ein automatischer Test fehlgeschlagen ist.

- „Eliminate duplication“ [Bec02, S. ix] – Entferne Duplikation.

Aus diesen zwei Regeln leitet Beck die drei Schritte der testgetriebenen Entwicklung ab. Wir übernehmen im Prinzip Becks Definition [Bec02, S. x], präzisieren sie aber ein wenig, indem wir zwischen *Anwendungs-* und *Testcode* unterscheiden. Anwendungscode ist dabei der Code, der an den Kunden ausgeliefert wird und die Funktionalität der Anwendung implementiert. Testcode dient nur zur Überprüfung der Anwendung und wird für gewöhnlich nicht an den Kunden ausgeliefert. Zusätzlich verwenden wir die Begriffe *erfolgreich* und *fehlgeschlagen* um die Ergebnisse eines Testlaufs zu beschreiben. Ein erfolgreicher Testlauf, ist ein Testlauf, bei dem kein Testfall einen Fehler anzeigt. Bei

einem fehlgeschlagenen Testlauf trat dagegen bei mindestens einem Testfall ein Fehler auf. Mit diesen Begriffen lässt sich das Vorgehen bei der testgetriebenen Entwicklung wie folgt beschreiben:

Testcode schreiben Schreibe einen Testfall, der fehlschlägt. Falls nötig, schreibe gerade so viel Anwendungscode, dass sich der Testfall übersetzen lässt[5]. Dieser Schritt beginnt mit einem erfolgreichen und endet mit einem fehlgeschlagenen Testlauf.

Anwendungscode schreiben Schreibe gerade so viel Code, dass alle Testfälle erfolgreich laufen. Dieser Schritt beginnt mit einem fehlgeschlagenen und endet mit einem erfolgreichen Testlauf.

Umstrukturieren Entferne jegliche in den zwei vorausgehenden Schritten eingeführte Duplikation (und andere Verstöße gegen einen guten Programmierstil) aus dem Anwendungs- und Testcode. Dieser Schritt beginnt und endet mit einem erfolgreichen Testlauf.

Für eine komplette Folge dieser Schritte in der Reihenfolge ihrer Aufführung wird in dieser Arbeit der Ausdruck des *TGE-Zyklus* verwendet. Der gesamte testgetriebene Entwicklungsprozess setzt sich im Idealfall aus einer Folge solcher TGE-Zyklen zusammen, wie in Abbildung 2.4 dargestellt. Wie Link [Lin05, S. 12] schreibt, soll dabei kein einzelner TGE-Zyklus länger als fünf Minuten dauern, was den iterativen Charakter der testgetriebenen Entwicklung verdeutlicht. Aus unserer Definition des testgetriebe-

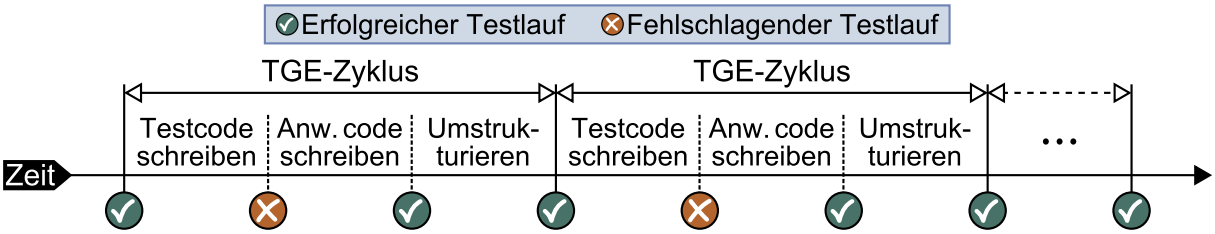


Abbildung 2.4: Der idealisierte TGE-Prozess

nen Entwicklungsprozess leiten wir drei Regeln ab, nach denen wir entscheiden, ob eine durch den Entwickler vorgenommene Änderung im Anwendungscode mit dem Entwicklungsprozess konform ist oder nicht:

[5] Beck erlaubt auch Testfälle, die sich nicht übersetzen lassen (vgl. [Bec02, S. x]). Unser Werkzeug TestPulse ist jedoch nicht in der Lage, solche Fälle automatisch korrekt zu klassifizieren. Abhilfe schafft hier die Möglichkeit von TestPulse zur manuellen Begutachtung der Änderungen (s. auch Abschnitt 2.3.3).

Regel 1 Die Änderung liegt innerhalb einer bereits existierenden Methode, die von einem fehlschlagenden Testfall des letzten Testlaufs vor dem Zeitpunkt der Änderung aufgerufen wurde.

Regel 2 Die Änderung liegt innerhalb einer neu angelegten Methode, die von einem fehlschlagenden Testfall des nächsten Testlaufs nach dem Zeitpunkt der Änderung aufgerufen wurde.

Regel 3 Eine Umstrukturierung ändert die Struktur des Codes, jedoch nicht sein Verhalten (vgl. [Fow99, S. 8]).

Die Regeln 1 und 2 können relativ leicht automatisch überprüft werden, wenn alle Änderungen und die Testläufe protokolliert wurden. Für Regel 3 ist dies theoretisch ebenfalls denkbar, obgleich dies technisch sehr aufwändig ist. Daher implementiert unser Werkzeug, wie der nächste Abschnitt näher erklären wird, nur eine halbautomatische Erkennung von Umstrukturierungen.

2.3 Halbautomatische Erkennung der testgetriebenen Entwicklung

Um bestimmen zu können, ob eine Änderung im Anwendungscode mit den Regeln der testgetriebenen Entwicklung übereinstimmt, muss der Entwicklungsprozess genau protokolliert werden. Zu diesem Zweck wurde an unserem Lehrstuhl TestPulse[6] entwickelt. Die erste Idee und Umsetzung zu diesem Werkzeug stammt noch von Müller und ist in [MH07] erstmals erläutert. In der Zwischenzeit wurden die einzelnen Komponenten von TestPulse mehrfach überarbeitet [Leo07, Sch09]. Die letzte große Weiterentwicklung wurde von Philipp [Phi08] durchgeführt. In seiner Arbeit findet sich auch eine Erklärung der Arbeitsweise von TestPulse, an der wir uns in diesem Anschnitt orientiert haben. Zusätzlich berücksichtigen wir bei unserer Erklärung Verbesserungen an TestPulse, die durch den Autor dieser Arbeit in Zusammenarbeit mit einer am Lehrstuhl angestellten wissenschaftlichen Hilfskraft umgesetzt wurden.

2.3.1 Übersicht über TestPulse

Die Abbildung 2.5 gibt einen Überblick über die Architektur von TestPulse. Wie man dort sieht, besteht TestPulse aus zwei Komponenten: Die erste dient zur Datensammlung, die zweite zur Datenanalyse. Die Unterteilung in zwei Komponenten hat den Vorteil, dass sich die Datensammlungskomponente, die durch vier Eclipse-Plugins realisiert

[6] Der Quellcode von TestPulse ist im Internet frei verfügbar. Näheres dazu in Anhang D.

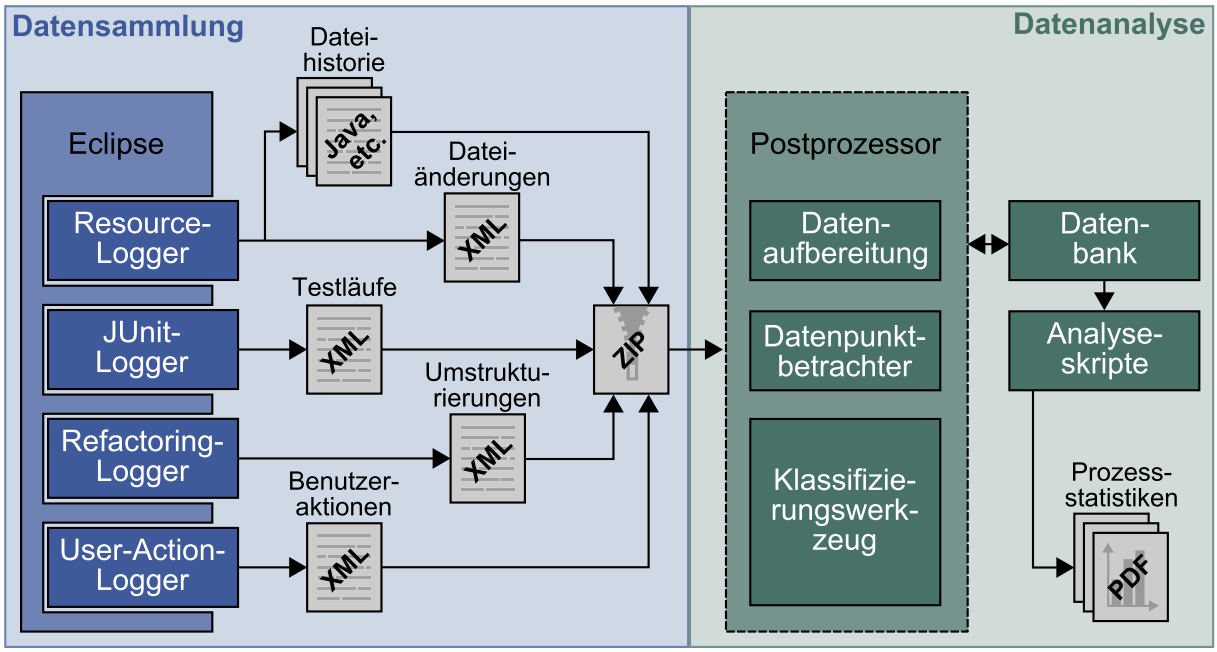


Abbildung 2.5: Die Architektur von TestPulse.

ist, sehr einfach auf den Rechnern von Experimentteilnehmern installieren lässt. Die komplexere Datenanalysekomponente muss dann nur einmal auf dem Rechner des Experimentleiters installiert werden. Dies erleichtert die Durchführung von Experimenten und Feldstudien mit unserem Werkzeug. Zudem schreiben die Eclipse-Plugins die gesammelten Ereignisse quasi unverarbeitet in die Protokolldateien, so dass bei der Datensammlung keine von den Teilnehmern wahrnehmbaren Zeitverzögerungen in der Eclipse-Entwicklungsumgebung entstehen. Die Vermeidung von Zeitverzögerungen ist wichtig, da die Datensammlung in Studien sonst von den Teilnehmern als störend empfunden würde und sich somit möglicherweise ihr Programmierverhalten verändert.

2.3.2 Datensammlung

Wie bereits zuvor erwähnt und in Abbildung 2.5 ersichtlich, besteht die Datensammlungskomponente aus vier Eclipse-Plugins. Die vier Eclipse-Plugins erzeugen jeweils eine XML-Protokolldatei. Der JUnit-Logger hält in seiner Protokolldatei die Ausführung von JUnit-Testläufen fest. Er vermerkt dabei jeweils den Start und das Ende des gesamten Testlaufs und aller darin ausgeführten Testfälle. Weiterhin schreibt er die Ergebnisse der Testfälle und des gesamten Testlaufs in die Protokolldatei. Im Falle von fehlschlagenden Testfällen wird zudem die Ursache für den Fehlschlag festgehalten.

Der Resource-Logger speichert neben einem Protokoll aller geänderten Dateien zudem die aktuellen Versionen dieser Dateien mit Zeitstempeln in der Dateihistorie ab.

Die wichtigsten Dateien sind hierbei die Java-Dateien, die später bei der Datenanalyse verwendet werden, um den Zustand des Programms zu jedem Testlauf rekonstruieren zu können. Das Speichern der veränderten Dateien durch den Resource-Logger wird durch Speichervorgänge in der Eclipse-Entwicklungsumgebung ausgelöst und zwar unabhängig davon, ob der Entwickler freiwillig speichert oder ob Eclipse automatisch speichert, wie z. B. vor dem Ausführen von Testläufen oder des Eclipse-Formatierungswerkzeugs.

Eclipse bietet eine stetig wachsende Zahl automatischer Umstrukturierungen an. Um diese Umstrukturierungen automatisch korrekt als solche klassifizieren zu können, protokolliert der Refactoring-Logger deren Ausführung mit Zeitstempel und den Namen der von der jeweiligen Umstrukturierung betroffenen Dateien.

Der UI-Logger registriert Fokuswechsel in der Eclipse-Benutzeroberfläche, so kann man nachvollziehen, wie lange sich der Entwickler in welcher Ansicht der Eclipse-Benutzeroberfläche aufhält. Derzeit werden diese Daten allerdings derzeit nicht ausgewertet.

Am Ende einer mit der Datensammlungskomponente protokollierten Programmiersitzung lassen sich die komplette Dateihistorie und alle XML-Protokolldateien zu einer ZIP-Datei komprimieren und exportieren und sind damit zur Datenanalyse bereit.

2.3.3 Datenanalyse

Zur Datenanalyse einer Programmiersitzung startet man den PostProzessor von TestPulse und importiert die ZIP-Datei mit den gesammelten Protokollen und gibt die Metadaten wie z. B. die Gruppenzugehörigkeit an. Sobald man die Eingaben bestätigt, beginnt der PostProzessor mit dem Einlesen und Entpacken der ZIP-Datei. Danach werden alle Versionen der Java-Dateien mithilfe des Formatierungswerkzeugs Jalopy [Jal] in ein einheitliches Format gebracht. Dabei werden Instanzvariablen, Methoden und Konstruktoren, etc. in alphabetischer Reihenfolge sortiert, Kommentare entfernt und nicht druckbare Zeichen vereinheitlicht. Somit spielen Änderungen, wie beispielsweise das Verschieben von Methoden innerhalb der Klasse oder das Hinzufügen von Leerzeilen bei der späteren Analyse, keine Rolle mehr.

Im Anschluss daran werden die verschiedenen Versionen der Java-Dateien aufgrund ihres Namens jeweils einer Java-Datei zugeordnet. So werden beispielsweise die Java-Dateiversionen *Klasse.123.java* und *Klasse.234.java* der Java-Datei *Klasse* zugeordnet. Sobald alle Java-Dateiversionen einer Java-Datei zugeordnet wurden, wird für jede Dateiversion der abstrakte Syntaxbaum aufgebaut und daraus die Positionen der ausführbaren Element, d. h. Methoden und Konstruktoren, im Quelltext bestimmt. Diese Information wird später für den Vergleich der Änderungen benötigt.

Weiterhin wird festgelegt, welche Java-Dateien zum Test- und welche zum Anwen-

dungscode gehören. Java-Dateien gelten als Testcode, sobald ihre Namen die Zeichenketten *Test*, *Stub*, *Dummy* oder *Mock* enthalten. Alle anderen Java-Dateien werden als Anwendungscode betrachtet. Obgleich diese Regel sehr simpel ist, hat sie für alle in dieser Arbeit ausgewerteten Datensätze zu einer korrekten Klassifikation geführt.

Im nächsten Arbeitsschritt werden alle XML-Protokolldateien (außer der des UI-Loggers) eingelesen, damit die nötigen Informationen über Testläufe und Umstrukturierungen zur Verfügung stehen. Dann werden die Änderungen an den Java-Dateien bestimmt. Hierzu werden zunächst die Parameter von Konstruktoren und Methoden sowie die Rückgabewerte von Methoden auf ihren vollqualifizierenden Namen zurückgeführt. Damit lassen sich Konstruktor- und Methodensignaturen zuverlässig vergleichen. Auf diese Weise wird in Signaturen z. B. erkannt, dass es sich bei *Object* und *java.lang.Object* um den gleichen Typ handelt.

Nachdem die vollqualifizierenden Namen der Parameter und Rückgabewerte gefunden wurden, werden die Änderungen an den ausführbaren Elementen zwischen zwei aufeinander folgenden Versionen einer Java-Datei bestimmt. Wird ein ausführbares Element in beiden aufeinander folgenden Versionen einer Java-Datei gefunden, dann wird mittels des Diff-Algorithmus bestimmt, welche Zeilen sich geändert haben und das Element wird als *geändert* markiert. Findet sich ein ausführbares Element nur in der neueren der beiden Versionen so wird es als *hinzugefügt* markiert. Findet es sich hingegen nur in der älteren der beiden Versionen wird es als *gelöscht* markiert. Im Anschluss an die

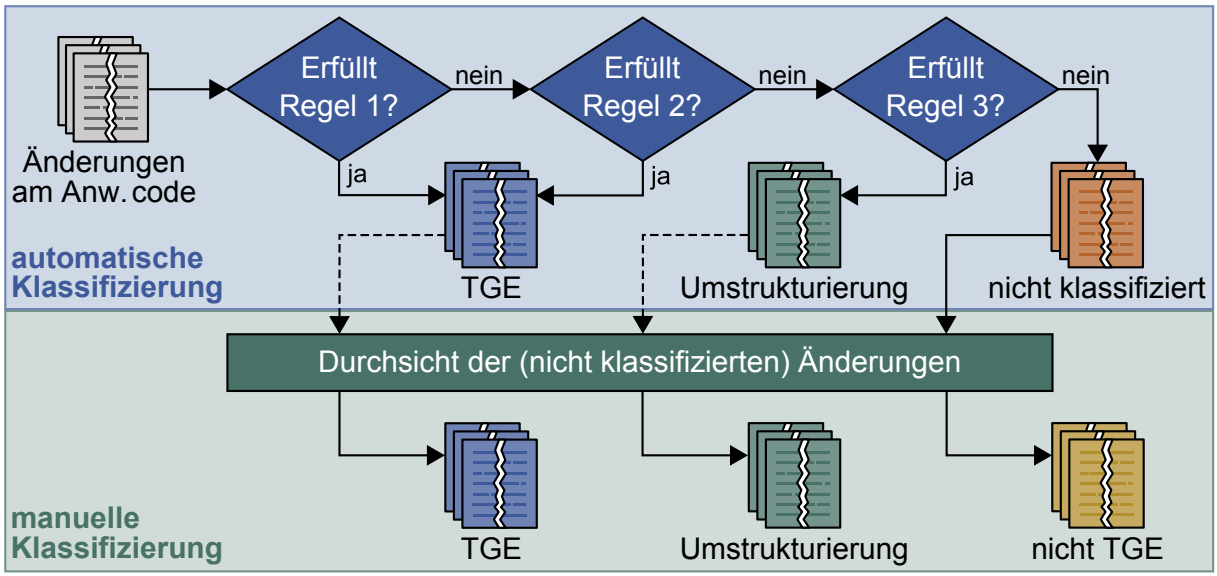


Abbildung 2.6: Der Klassifizierungsprozess.

Bestimmung der Änderungen werden diese klassifiziert. Wie man aus Abbildung 2.6 ersehen kann, ist der Klassifizierungsprozess zweigeteilt. Der automatische Teil der Klassifizierung wird noch vom PostProzessor im Zuge der Datenaufbereitung übernommen, zur manuellen Klassifizierung dient das in den PostProzessor integrierte grafische Klassifizierungswerkzeug. Für die automatische Klassifizierung prüft der PostProzessor zunächst für alle Änderungen an ausführbaren Elementen im Anwendungscode, ob sie den in Abschnitt 2.2 beschriebenen Regeln 1 und 2 genügen. Beide Regeln fordern, dass eine Änderung im Anwendungscode durch einen fehlschlagenden Testfall abgedeckt ist. Dieser Testfall muss entweder gemäß Regel 1 direkt vor oder gemäß Regel 2 direkt nach der Änderung ausgeführt worden sein. Um dies überprüfen zu können, muss der PostProzessor feststellen, welche Änderungen an Methoden und Konstruktoren durch einen fehlschlagenden Testfall abgedeckt werden. Zu diesem Zweck wird für jeden aufgezeichneten Testlauf der Zustand aller Java-Dateien zum Zeitpunkt des Testlaufs aus der Dateihistorie wieder hergestellt. Im Anschluss daran wird der jeweilige Testlauf erneut ausgeführt und die Aufrufspur der aufgerufenen Methoden und Konstruktoren ermittelt. Aus der Aufrufspur lassen sich nun die benötigten Informationen über die geforderte Testabdeckung ermitteln. Alle Änderungen, welche die Regeln 1 und 2 erfüllen, werden automatisch als *testgetriebene Änderungen* klassifiziert.

Für die verbleibenden Änderungen prüft der PostProzessor, ob sie eventuell Regel 3 entsprechen. Diese Regel erlaubt Umstrukturierungen am Code, sofern sie keine Auswirkung auf das Verhalten des Programms haben. Damit sich Regel 3 überprüfen lässt, verknüpft der PostProzessor die Informationen aus der Protokolldatei des Refactoring-Loggers mit den Informationen über Änderungen an den ausführbaren Elementen des Anwendungscodes. Auf diese Weise kann der PostProzessor feststellen, welche Änderungen Teil einer in Eclipse durchgeführten automatischen Umstrukturierung waren und sie dementsprechend als *Umstrukturierung* markieren.

Nach der Klassifizierung der automatischen Umstrukturierungen wird ein Datenpunkt in der Datenbank angelegt und die bisherigen Ergebnisse der automatischen Klassifikation werden unter diesem Datenpunkt gespeichert. Damit ist die automatische Klassifizierung abgeschlossen und der integrierte Datenpunktbetrachter zeigt den neuen Datenpunkt an.

Alle bis dahin nicht eindeutig klassifizierten Änderungen müssen später durch einen Gutachter manuell klassifiziert werden. Der Sinn der manuellen Klassifizierung liegt hauptsächlich darin, auch manuell durchgeführte Umstrukturierungen erkennen zu können. Auf diese Weise lassen sich aber auch einige nicht durch die Regeln 1 und 2 gedeckten Sonderfälle der testgetriebenen Entwicklung korrekt klassifizieren und die bereits automatisch klassifizierten Änderungen kontrollieren, wie die gestrichelten Pfeile in Abbildung 2.6 andeuten sollen. Zur manuellen Klassifizierung kann der Gutachter den

Datenpunkt in das grafische Klassifizierungswerkzeug des PostProzessors laden. Wie in Abbildung 2.7 zu sehen, bekommt er dort die Änderungen an den Dateien zusammen mit Informationen über die vor und nach der Änderung ausgeführten Testläufe angezeigt. Aufgrund dieser Informationen kann der Gutachter für alle verbleibenden Änderungen

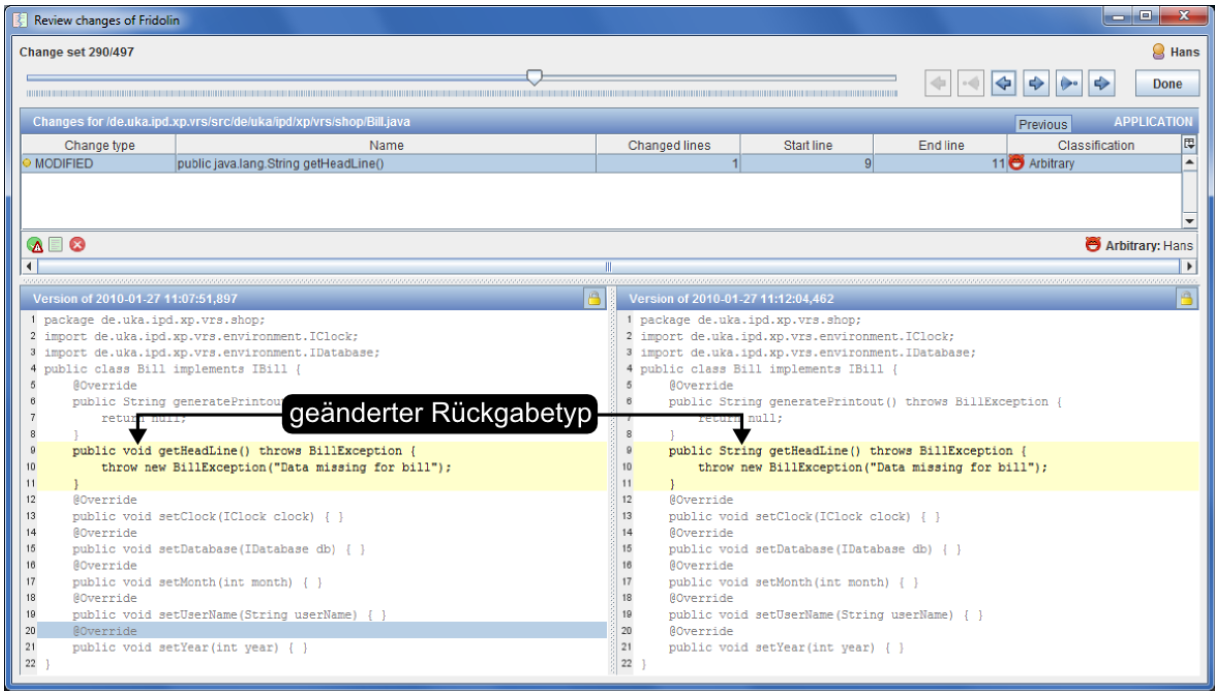


Abbildung 2.7: Bildschirmfoto des Klassifizierungswerkzeugs.

entscheiden, in welche der drei Klassen (testgetriebene Änderung, Umstrukturierung, nicht testgetriebene Änderung) sie fallen. Da der manuelle Begutachtungsprozess eine potentielle Fehlerquelle darstellt, unterstützt das Werkzeug die unabhängige Klassifikation von Änderungen durch mehrere unabhängige Gutachter. Die Auflösung eventuell entstehender Konflikte wird ebenfalls durch das grafische Klassifikationswerkzeug unterstützt.

2.4 Metriken

Ist die Datenanalyse abgeschlossen, lassen sich aus TestPulse zahlreiche Metriken zum (testgetriebenen) Entwicklungsprozess gewinnen. Zusätzlich werden in dieser Arbeit weitere manuell oder halbautomatisch ermittelte Metriken verwendet, die eventuell einer Erklärung bedürfen. Da diese Metriken in allen drei Studien verwendet werden, werden

sie in den folgenden Abschnitten an zentraler Stelle erläutert. Um die Übersicht zu erleichtern, wurden die Metriken drei Oberbegriffen, nämlich der Produktivität, Qualität und dem TGE-Prozess zugeordnet. Diese Zuordnung wird auch in den Abschnitten zu den Ergebnissen der drei Studien wieder aufgegriffen.

2.4.1 Produktivität

Zur Produktivität zählen die Bearbeitungszeit sowie die Änderungen am Code, d. h. die Anzahl der geänderten Dateien und die geänderten Zeilen Code und der Anteil der Änderungen am Testcode.

2.4.1.1 Bearbeitungszeit

Die Bearbeitungszeit ist die Zeit vom Austeilen der Aufgabenstellung bis zum Bestehen des Akzeptanztests abzüglich der Zeiten für das erstmalige Lesen der Aufgabenstellung, Pausen und Akzeptanztests. Sie wird in Minuten angegeben. Bei den beiden Studien mit Paaren kam es öfters vor, dass nur einer der Teilnehmer eine Pause gemacht hat. In diesen Fällen wurde die Zeit für diese Pause nur zur Hälfte abgezogen.

Die zur Berechnung der Bearbeitungszeit notwendigen Zeitdaten mussten die Teilnehmer in dafür vorgesehene Zeiterfassungsbögen in der Aufgabenstellung erfassen. Um die Zuverlässigkeit der Zeitdaten zu erhöhen, hat der Experimentleiter die Zeiterfassungsbögen während der Durchführung der Studien möglichst häufig auf Vollständigkeit kontrolliert. Während ihrer Pausen wurden die Teilnehmer daran erinnert, diese Pausen gewissenhaft zu erfassen.

2.4.1.2 Änderungen am Code

In allen drei Studien handelt es sich bei den Programmieraufgaben um Wartungsaufgaben, d. h. die Teilnehmer haben nie mit einem leeren Projekt begonnen sondern entwickelten eine vorhandene Codebasis weiter. Um die Veränderungen an dieser Codebasis quantifizieren zu können, werden in dieser Arbeit die Anzahl der geänderten Dateien und die geänderten Zeilen Code angegeben. Um diese zwei Metriken zu erheben, wurde die ursprünglich an die Teilnehmer ausgegebene Codebasis mit den am Ende von den Teilnehmern abgegebenen Lösungen verglichen. Zunächst wurden dazu alle Java-Dateien mit dem Formatierungswerkzeug Jalopy [Jal] in ein von uns definiertes, bereinigtes Format gebracht. Es handelt sich dabei um das gleiche Format, das auch von TestPulse verwendet wird, bevor es die Änderungen zwischen zwei aufeinanderfolgenden Versionen einer Java-Datei bestimmt (vgl. Abschnitt 2.3.3). Auch für die Angabe der Größe

```
1  package p;
2
3  import k.A;
4
5  /**
6   * @author Marvin
7   */
8  public class FormatMe {
9
10     final static int i = 0;
11
12     private A a = new A();
13
14     public void setA(A a) {
15         // don't accept null
16         if (a != null)
17             this.a = a;
18     }
19
20     public A getA() {
21         return a;
22     }
23 }
```

a) Eclipse-Standardformat

```
1  package p;
2  import k.A;
3  public class FormatMe {
4      static final int i = 0;
5      private A a = new A();
6      public A getA() {
7          return a;
8      }
9      public void setA(A a) {
10         if (a != null)
11             this.a = a;
12     }
13 }
```

b) Bereinigtes Format

Abbildung 2.8: Gegenüberstellung von Eclipse-formatiertem und bereinigtem Quelltext.

der Codebasen der in den Studien verwendeten vier Programmieraufgaben[7] wurden die Java-Dateien zuvor in dieses bereinigte Format gebracht.

Abbildung 2.8 verdeutlicht den Effekt, der durch das Anwenden unseres bereinigten Formats entsteht. Auf der linken Seite findet sich ein Stück Code, das mit dem in Eclipse integrierten Formatierungswerkzeug und dessen Standardeinstellungen[8] formatiert wurde. Die rechte Seite zeigt das gleiche Stück Code, nachdem es mit Jalopy und unseren Einstellungen formatiert wurde. Der Code ist nun deutlich kürzer. Er enthält weder Leerzeilen noch Kommentare. Zudem sind die Klassenelemente zunächst nach Typ und dann, wie man an den Methoden sieht, alphabetisch geordnet[9]. Die Modifizierer in Zeile 10, links bzw. Zeile 4, rechts wurden ebenfalls umsortiert. Die von uns für Jalopy eingestellten Formatierungsregeln bewirken noch andere Vereinheitlichungen, die

[7] Beschrieben in den Abschnitten 4.5, 5.6 und 6.5.

[8] Eclipse in der Version 3.6 (Helios), Profil „Eclipse [built-in]“.

[9] Dies lässt sich zumindest teilweise auch in Eclipse über „Save Actions“ durchsetzen.

in diesem kurzen Beispiel nicht sichtbar werden. Wichtig ist die Feststellung, dass auf diese Weise formatierter Code im Durchschnitt deutlich kürzer ist, als man es normalerweise gewohnt ist. Dazu trägt auch die längere maximal mögliche Zeilenlänge von 120 statt der üblichen 80 Zeichen bei.

Nachdem alle Java-Dateien aus der Codebasis und den Lösungen der Teilnehmer einheitlich formatiert wurden, wurde als nächstes für jeden Teilnehmer bestimmt, welche Java-Dateien modifiziert, gelöscht und hinzugefügt wurden. Die Zuordnung zwischen den Java-Dateien in der ausgeteilten Codebasis und der abgegebenen Lösung geschieht dabei über deren vollständigen[10] Namen. Da dies dazu geführt hätte, dass das Umbenennen oder Verschieben von Dateien sich in jeweils einer gelöschten und einer hinzugefügten Datei niederschlägt, wurden in diesen Fällen die Java-Dateien in der abgegebenen Lösung vor der Bestimmung der geänderten Dateien und Zeilen nach manueller Durchsicht so umbenannt beziehungsweise verschoben, dass sie über den vollständigen Namen einander korrekt zugeordnet werden konnten. Umbenannte oder verschobene Java-Dateien tragen damit mit nur jeweils einem Zähler zu den geänderten Dateien bei. Weiterhin wurden auf diese Weise nur die tatsächlichen Änderungen am Inhalt einer solchen Datei als geänderte Zeilen erfasst.

Das Zählen der geänderten Zeilen Code wurde automatisch mit einem Unix-Bash-Skript durchgeführt. Bei den hinzugefügten und gelöschten Java-Dateien misst es einfach die Länge der Java-Datei in Zeilen. Bei modifizierten Java-Dateien wird mittels des Diff-Algorithmus die Anzahl der geänderten Zeilen Code bestimmt. Als Einheit der geänderten Zeilen Code wird in dieser Arbeit Source Lines of Code kurz SLOC verwendet.

Aus den geänderten Zeilen Code wurde unter Verwendung der Einteilung in Anwendungs- und Testcode der Anteil der geänderten Zeilen Testcode an allen geänderten Zeilen bestimmt. Dieser Wert wird in Prozent angegeben.

2.4.2 Qualität

Zu den Qualitätsmetriken gehören in erster Linie die Änderungen an den Testfällen und als zweiter Indikator die Anzahl der benötigten Akzeptanztestläufe, um den Akzeptanztest zu bestehen.

2.4.2.1 Änderungen an den Testfällen

In allen Studien sollten die Teilnehmer unabhängig vom geforderten Entwicklungsprozess automatisierte Komponententests nutzen, um die Qualität ihrer Lösung zu gewährleisten. Die Programme zu allen Aufgaben wurden daher mit Testfällen ausgeliefert.

[10] Der Dateiname der Java-Datei inklusive allen Ordnern, die Teil der Paketstruktur sind.

Änderungen an den Testfällen quantifizieren wir in dieser Arbeit durch zwei Kennzahlen: Die Änderung der Anzahl von Testfällen und die Änderung der Anweisungsabdeckung des Anwendungscodes. Zur Bestimmung der einzelnen Anweisungsabdeckungen wurde EclEmma [Ecl] eingesetzt. Beim Durchführen der Messungen zur Anweisungsabdeckung ließ sich gleichzeitig die Anzahl der Testfälle ablesen. Die Änderung der Anzahl der Testfälle ergibt sich als die Anzahl der Testfälle in der Lösung eines Teilnehmers minus der Anzahl der Testfälle in der dazugehörigen ausgegebenen Codebasis. Gemäß diesem Schema berechnen wir auch die Änderung der Anweisungsabdeckung des Anwendungscodes. Als Anwendungscode zählen dabei, wie schon in Abschnitt 2.3.3 erwähnt, alle Java-Dateien, die nicht *Test*, *Stub*, *Dummy* oder *Mock* in ihrem Namen enthalten. Die Änderung der Anweisungsabdeckung wird in Prozentpunkten angegeben.

2.4.2.2 Anzahl der Akzeptanztestläufe

In allen drei Studien wurde mit Akzeptanztests sichergestellt, dass die abgegebenen Lösungen die in den Aufgabenstellungen geforderte Funktionalität und Qualität lieferten. In allen Aufgabenbeschreibungen wurde gefordert, dass der Akzeptanztest beim ersten Versuch zu bestehen ist. Erst wenn ein Teilnehmer den Akzeptanztest nicht bestand, wurde ihm mitgeteilt, dass er einen weiteren Versuch hat.

2.4.3 TGE-Prozess

Die Metriken zum TGE-Prozess entstammen alle aus TestPulse und umfassen die TGE-Prozesstreue, die Charakteristiken der Testläufe und die Anzahl der automatischen Umstrukturierungen.

2.4.3.1 TGE-Prozesstreue

Nach der vollständigen Datenanalyse mit TestPulse, wie in Abschnitt 2.3.3 beschrieben, liegen alle Änderungen an Methoden und Konstruktoren im Anwendungscode in drei Klassen eingeteilt vor: testgetriebene Änderungen, Umstrukturierungen, nicht testgetriebenen Änderungen. Ferner ist zu allen Änderungen deren Größe in Zeilen Code bekannt.

Wir bezeichnen nun mit T die Menge der testgetriebenen Änderungen, mit U die Menge der Umstrukturierungen und mit N die Menge der nicht testgetriebenen Änderungen. Ferner bezeichnet $|X|$ die Anzahl der Änderungen in einer der drei zuvor definierten Mengen und x_i die i-te Änderung in dieser Menge. Die Funktion $z(x_i)$ liefert die Anzahl der geänderten Zeilen zur Änderung x_i. Hiermit bilden wir zunächst die mit den Zeilen

Code gewichteten Summen der testgetriebenen Änderungen $\hat{T}$, Umstrukturierungen $\hat{U}$ und nicht testgetriebenen Änderungen $\hat{N}$:

$$\hat{T} = \sum_{i=1}^{|T|} z(t_i), \qquad \hat{U} = \sum_{j=1}^{|U|} z(u_j), \qquad \hat{N} = \sum_{k=1}^{|N|} z(n_k) \tag{2.6}$$

Diese Summen nennen wir von nun an die *akkumulierten* testgetriebenen Änderungen, Umstrukturierungen bzw. nicht testgetriebenen Änderungen. Aus ihnen lässt sich gemäß folgender Formel die TGE-Prozesstreue in Prozent berechnen:

$$\text{TGE-Prozesstreue } [\%] = \frac{\hat{T} + \hat{U}}{\hat{T} + \hat{U} + \hat{N}} \times 100 \tag{2.7}$$

Die TGE-Prozesstreue ist folglich definiert als die akkumulierten testgetriebenen Änderungen plus die akkumulierten Umstrukturierungen geteilt durch alle akkumulierten Änderungen umgerechnet in Prozent.

2.4.3.2 Anzahl der automatischen Umstrukturierungen

Wie ebenfalls bereits in den Abschnitten 2.3.2 und 2.3.3 angesprochen, protokolliert TestPulse die in Eclipse automatisch ausgeführten Umstrukturierungen[11]. Deren Anzahl lässt sich daher aus den Protokollen leicht bestimmen.

2.4.3.3 Testläufe

Aus den Protokollen des JUnit-Loggers von TestPulse lässt sich die Anzahl der Testläufe bestimmen. Zusätzlich wird der mittlere zeitliche Abstand der Testläufe sowie die dazugehörige Standardabweichung berechnet. Der mittlere Abstand und die Standardabweichung werden in Minuten angegeben. Aus dem mittleren Abstand $\bar{x}$ und der Standardabweichung s lässt sich der Variationskoeffizient der Testläufe v nach folgender Formel berechnen:

$$v\ [\%] = \frac{s}{\bar{x}} \times 100 \tag{2.8}$$

Von den eben genannten Metriken zu den Testläufen werden bei der Auswertung nur der mittlere Abstand der Testläufe und der Variationskoeffizient berücksichtigt. Die Anzahl der Testläufe und deren Standardabweichung finden sich aber zu jeder Studie im dazugehörigen Anhang wieder.

[11] In der Anleitung zur aktuellen Eclipse-Version 3.6 (Helios) findet sich eine vollständige Liste der verfügbaren automatischen Umstrukturierungen: http://help.eclipse.org/helios/index.jsp?topic=/org.eclipse.jdt.doc.user/reference/ref-menu-refactor.htm (Stand: 21. 02. 2011).

Kapitel 3

Verwandte Arbeiten

Die ersten zwei in dieser Arbeit vorgestellten Studien (s. Kapitel 4 und Kapitel 5) beleuchten Aspekte der Paarprogrammierung. Aus diesem Grund gibt der erste Abschnitt dieses Kapitels einen Überblick über die verwandten Arbeiten in diesem Bereich. Da sich alle Studien in dieser Arbeit mit der testgetriebenen Entwicklung auseinandersetzen, fasst Abschnitt 3.2 die verwandten Arbeiten zu diesem Thema zusammen. Der letzte Abschnitt in diesem Kapitel befasst sich mit Werkzeugen zur Untersuchung der testgetriebenen Entwicklung, die mit unserem eigenen Werkzeug TestPulse vergleichbar sind.

3.1 Arbeiten zur Paarprogrammierung

Der Überblick der Arbeiten zur Paarprogrammierung ist unterteilt in den folgenden Abschnitt über Studien, die Paarprogrammierung mit verschiedenen Ausprägungen der konventionellen Einzelprogrammierung vergleichen und in einen zweiten Abschnitt, der sich mit Studien über den Einfluss menschlicher Faktoren bei der Paarprogrammierung beschäftigt.

3.1.1 Vergleiche mit Einzelprogrammierung

Die Metaanalyse von Dybå und Kollegen [DAS+07] fasst die Ergebnisse aus insgesamt 15 Studien aus dem Zeitraum von 1998 bis 2007 zusammen, in denen die Einzel- mit der Paarprogrammierung verglichen wurden. Dybå und Kollegen konzentrierten sich bei ihrer Metaanalyse auf die Auswirkung der Paarprogrammierung auf die Dauer, den Aufwand und Qualität der Implementierung. Da nicht alle 15 Studien zu jedem der drei Maße eine Angabe enthielten, beruhen die Ergebnisse der Metaanalyse bei der Dauer und der Qualität auf den Angaben aus 11 Studien und beim Aufwand auf den Angaben aus 7 Studien. Um die Studien vergleichbar zu machen, wurde zu den in den Studien gefundenen Angaben über die Unterschiede zwischen Einzel- und Paarprogrammierung

bei Dauer, Aufwand und Qualität die jeweilige Effektgröße (Hedges' g) berechnet. Anschließend wurde für jedes der drei Maße die gemeinsame Effektgröße ermittelt. Hierzu verwendeten Dybå und Kollegen das gewichtete Mittel der einzelnen Effektgrößen der Studien, wobei das Gewicht einer Studie umgekehrt proportional zu ihrer Varianz gewählt wurde. Die so erhaltenen Effektgrößen betragen $0,40$ bei der Dauer, $0,38$ bei der Qualität und $-0,57$ beim Aufwand. Die Effektgrößen deuten auf einen moderaten Vorteil der Paarprogrammierung gegenüber der Einzelprogrammierung bei Dauer und Qualität hin, der durch einen etwas größeren, aber immer noch moderaten Nachteil beim Aufwand erkauft wird.

Die größte in der Metaanalyse untersuchte Studie stammt von Arisholm und Kollegen [AGDS07]. Arisholm und Kollegen berichten in dieser Studie über ein Quasi-Experiment mit dem sie die Auswirkungen der Programmiererfahrung und der Systemkomplexität auf die Effektivität der Paarprogrammierung untersucht haben. An dem Quasi-Experiment nahmen insgesamt 296 Java-Berater teil. Für ihre Teilnahme erhielten sie ein pauschales Entgelt, das ihrem regulären Gehalt für fünf Stunden Arbeit entsprach. Der Versuch hatte einen faktoriellen Entwurf (vgl. [SCC02, S. 263 f.]), d. h. es wurde mehr als eine unabhängige Variable variiert. In diesem Fall waren es drei unabhängige Variablen: die Programmiermethode, Programmiererfahrung der Teilnehmer und die Systemkomplexität. Durch die drei unabhängigen Variablen mit ihren verschiedenen Ausprägungen ergaben sich insgesamt zwölf Gruppen. Bei der Programmiermethode wurde zwischen den Ausprägungen der Einzel- und Paarprogrammierung unterschieden. Die Zuweisung der Programmiermethode erfolgte nicht randomisiert. Die Datenpunkte der 98 Einzelprogrammierer stammen aus einem älteren Experiment aus dem Jahr 2001 [AS04], wohingegen die Datenpunkte der 99 Programmiererpaare in den Jahren 2004 und 2005 gesammelt wurden. Bei der Programmiererfahrung der Teilnehmer gab es die Kategorien Anfänger, Fortgeschrittene und Experten. Die Einteilung in diese Kategorien erfolgte gemäß den Einschätzungen der Vorgesetzten der Teilnehmer. Paare wurden anschließend stets so gebildet, dass beide Partner die gleiche Programmiererfahrung hatten. Bei der Systemkomplexität entschied das Los, ob die Teilnehmer die Variante einer in Java implementierten Kaffeemaschinensteuerung mit zentraler Steuerung und sieben Klassen oder mit verteilter, objektorientierter Steuerung und zwölf Klassen erweitern mussten. Neben einer Aufwärmaufgabe und einer weiteren Aufgabe zu einem Geldautomaten, die als Vortest diente, um die Programmiererfahrung nochmals zu überprüfen, mussten alle Teilnehmer vier Aufgaben zur eben erwähnten Kaffeemaschinensteuerung bearbeiten. Die letzte Aufgabe diente wiederum nur der Beschäftigung der schnelleren Teilnehmer. Direkt ausgewertet wurden daher nur die ersten drei Aufgaben zur Kaffeemaschinesteuerung. Im Vortest schnitten die Teilnehmer aus den Jahren 2004 und 2005 besser ab. Da dieser Unterschied die Ergebnisse zur Einzel- und Paarprogrammierung

verfälscht hätte, wurde die Auswertung mit statistischen Verfahren diesbezüglich bereinigt. In der Auswertung der drei Aufgaben zur Kaffeemaschinesteuerung wurden drei abhängige Variablen betrachtet: Die Dauer der Implementierung, der dazugehörige Aufwand und die Korrektheit. Die Korrektheit war als binäres Maß definiert: Eine Lösung galt als korrekt, wenn in den automatisierten Akzeptanztests zu den ausgewerteten Aufgaben kein Fehler auftrat, ansonsten galt sie als fehlerhaft. Fehlerhafte Lösungen wurden bei der Auswertung der Dauer der Implementierung und dem Aufwand ausgeschlossen.

Arisholm und Kollegen fanden heraus, dass die Paar- gegenüber der Einzelprogrammierung über allen Gruppen hinweg den Aufwand um 84 Prozent erhöht. Betrachtet man jedoch nur die Gruppen, welche die dezentrale Steuerung bearbeiteten, so gab es unter den Programmierpaaren 48 Prozent mehr korrekte Lösungen als unter den Einzelprogrammierern. Zudem konnte bei Paaren aus Anfängern eine um 20 Prozent kürze Dauer der Implementierung als bei den einzeln arbeitenden Anfängern beobachtet werden. Aufgrund der Ergebnisse der Studie kamen Arisholm und Kollegen zu dem Schluss, dass die Paarprogrammierung den größten Nutzen bringt, wenn relativ unerfahrene Programmierer an komplexen, dezentral gesteuerten Systemen arbeiten.

Ebenso wie die erste in dieser Arbeit vorgestellte Studie thematisieren Arisholm und Kollegen Auswirkungen der Programmiererfahrung im Zusammenhang mit der Paarprogrammierung. Allerdings ist ihre Zielsetzung eine andere. In der Studie von Arisholm und Kollegen steht der Vergleich der Einzel- und Paarprogrammierung im Vordergrund. Die Systemkomplexität und die Programmiererfahrung wurden auf ihre Vermittlereffekte bezüglich dieses Vergleichs betrachtet. Der Vergleich von Einzel- und Paarprogrammierung spielt in unserer Studie jedoch keine Rolle. Unser primäres Ziel ist es vielmehr, die testgetriebenen Entwicklerprozesse von erfahrenen und unerfahrenen Entwicklerpaaren zu untersuchen und dort die Unterschiede aufzudecken.

3.1.2 Einfluss menschlicher Faktoren auf die Paarprogrammierung

Sfetsos und Kollegen untersuchten in dem in [SSAD09] beschriebenen Experiment den Einfluss der Persönlichkeit von Programmierern auf die Effektivität der Paarprogrammierung. Dazu verglichen sie in einem Experiment Programmiererpaare, in denen die Partner das gleiche Keirsey-Temperament hatten mit Programmiererpaaren, in denen die Partner unterschiedliche Keirsey-Temperamente hatten. An dem Experiment nahmen insgesamt 70 Informatikstudenten der Universität Thessaloniki teil. Vor dem eigentlichen Experiment führten alle Teilnehmer einen Keirsey-Temperamenttest durch. Danach wurden zufällig Paare gebildet, dabei entstanden 17 Paare mit Partnern mit gleichem Keirsey-Temperament und 18 Paare mit Partnern mit unterschiedlichen Keirsey-Temperamenten. Die Paare mussten innerhalb von zweieinhalb Stunden zwei Wartungs-

aufgaben an einer in Java geschriebenen Kaffeemaschinensteuerung durchführen. Die Autoren maßen die Bearbeitungszeit sowie die Anzahl bestandener Akzeptanztests für beide Aufgaben und vergaben Punkte für die Korrektheit der Aufgaben. Weiterhin waren die Paare verpflichtet worden, ihre Kommunikation in einer den Experimentunterlagen beigefügten Tabelle zu kodieren. Ein Vergleich der beiden Gruppen zeigte, dass die Paare mit Partner mit unterschiedlichen Keirsey-Temperamenten deutlich schneller waren ($d = 1,833$[1], $p < 0,001$[2]), mehr Akzeptanztests bestanden ($d = 1,519$, $p = 0,006$) und auch mehr Punkte für die Aufgaben erhielten ($d = 1,657$, $p = 0,023$) als die Paare mit Partnern mit gleichem Keirsey-Temperament. Die Analyse der von den Paaren erstellten Kommunikationsprotokolle ergab, dass Partner mit unterschiedlichen Keirsey-Temperamenten mehr miteinander kommuniziert hatten als Partner mit gleichem Keirsey-Temperament ($d = 1,425$, $p = 0,003$). Nach dem Experiment mussten die Teilnehmer in einem Fragebogen Auskunft über die Arbeit im Paar während des Experiments geben. Die Teilnehmer, die mit einem Partner mit einem unterschiedlichen Keirsey-Temperament gearbeitet hatten, bewerteten die Zusammenarbeit, Kommunikation und Informationsbeschaffung signifikant positiver als die Teilnehmer, denen ein Partner mit gleichem Keirsey-Temperament zugeordnet worden war.

Wie man an den angegebenen Werten für die Effektgrößen der von Sfetsos und Kollegen untersuchten Variablen sehen kann, sind die von ihnen beobachteten Unterschiede zwischen den Gruppen sehr groß. Dies wird besonders deutlich, wenn man sich vor Augen hält, dass Cohen bereits eine Effektgröße d von $0,8$ als groß definiert hat. [Coh88, S. 24. ff.]. Diese großen Unterschiede haben uns dazu veranlasst, die in Kapitel 5 vorgestellte Studie durchzuführen, um zu überprüfen, ob sich diese Ergebnisse bestätigen lassen.

Erst kürzlich veröffentlichten Hannay und Kollegen eine Studie [HAES10], in der ebenfalls die Auswirkungen der Persönlichkeit auf die Paarprogrammierung untersucht wurden. Die Studie von Hannay und Kollegen war in die bereits in Abschnitt 3.1.1 beschriebene Studie von Arisholm und Kollegen [AGDS07] integriert. Hannay und Kollegen konzentrierten sich auf die Daten der 98 Programmiererpaare aus der Studie von Arisholm und Kollegen und werteten die Daten zur Dauer der Implementierung, dem dazugehörigen Aufwand und der Korrektheit unter Berücksichtigung der Persönlichkeit der Teilnehmer erneut aus. Die Bestimmung der Persönlichkeit der Teilnehmer erfolgte gemäß dem Big-Five-Modell (vgl. [Gol92]). Wie der Name andeutet, kennt das Big-Five-Modell fünf Dimensionen in denen die wesentlichen Merkmale der menschlichen Persönlichkeit beschrieben werden: Extraversion, Verträglichkeit, Gewissenhaftigkeit,

[1] Cohens d wurden nach der Formel in Gleichung 2.4 aus den Daten in Tabelle 4 des Artikels von Sfetsos und Kollegen [SSAD09] berechnet.

[2] p-Wert des Wilcoxon- bzw. Mann-Whitney-U-Tests aus Tabelle 6 in [SSAD09].

Neurotizismus, Offenheit für Erfahrungen[3]. Die Werte zu den fünf Dimensionen wurden mit einem validierten Test, dem so genannten Big-Five-Factor-Marker mit 100 Fragen, kurz BFFM-100 genannt, erhoben[4]. Das Ausfüllen des BFFM-100 war für die Teilnehmer aus ethischen Gründen freiwillig. Daher fehlten zu elf Teilnehmern die Ergebnisse dieses Tests. Die fehlenden Daten dieser 11 Teilnehmer wurden mit einem statistischen Verfahren geschätzt.

Für jede Dimension und jedes Paar ermittelten Hannay und Kollegen, wie stark sie im Paar ausgeprägt war und wie stark sich die Ausprägung der Dimension zwischen den Partnern unterschied. Die Ausprägungsstärke einer Dimension eines Paares ergab sich aus der Summe der Ausprägungsstärken der beiden Partner. Der Unterschied zwischen den Partnern wurde mit dem Absolutbetrag der Differenz der Ausprägungsstärken der beiden Partner gemessen. Den stärksten Effekt unter den Persönlichkeitsmerkmalen konnte für den Unterschied in der Extraversion der beiden Partner festgestellt werden. Dennoch war dieser Effekt bedeutend kleiner als die Effekte anderer Metriken wie der Systemkomplexität, dem Herkunftsland der Teilnehmer und deren Kompetenz. Weiter fanden die Autoren Wechselwirkungen der Persönlichkeitsmerkmale mit den anderen Metriken, allerdings waren die Auswirkungen der verschiedenen Persönlichkeitsmerkmale insgesamt nicht konsistent.

Auch wenn diese Studie aufgrund des anderen Persönlichkeitstests und aufgrund der Unterschiede bei Aufbau und Auswertung nur schwer mit der Studie von Sfetsos und Kollegen und damit auch mit unserer zweiten Studie vergleichbar ist, fällt doch auf, dass der Effekt der Persönlichkeit von Hannay und Kollegen als eher klein eingeschätzt wird, wohingegen er bei Sfetsos und Kollegen sehr groß war. Dies verdeutlicht die Notwendigkeit weiterer Studien zu diesem Thema.

Chong und Hurlbutt [CH07] beobachteten im Rahmen einer naturalistischen Langzeitstudie zwei Entwicklerteams in zwei verschiedenen Firmen bei der Paarprogrammierung. Die Autoren begleiteten die Teams jeweils über einen Zeitraum von vier Monaten. Während dieses Zeitraums protokollierten sie pro Team einmal wöchentlich die Aktivitäten eines Programmiererpaars für 1,5 bis 3 Stunden. Zusätzlich nahmen sie die Konversationen der Paare auf, transkribierten die Aufnahmen und analysierten die Transkripte anschließend. Durch ihre Beobachtungen fanden die Autoren heraus, dass die bisher angenommene und gelehrte Rollenverteilung innerhalb eines Paares von Fahrer (hat die Tastatur/Maus, arbeitet auf niedrigem Abstraktionsniveau und denkt algorithmisch) und Beifahrer (überwacht den Fahrer, arbeitet auf hohem Abstraktionsniveau und denkt stra-

[3] Übersetzung der in [HAES10] verwendeten englischen Begriffe *Extraversion*, *Agreeableness*, *Conscientiousness*, *Neuroticism* und *Openness to experience*.

[4] Zum Big-Five-Modell existieren mehrere validierte Tests mit unterschiedlich vielen Fragen, wie z. B. [Gol92] oder [Sau94].

tegisch) zumindest bei Paaren, in denen beide Partner vergleichbare Erfahrung haben, ein Mythos ist. Vielmehr arbeiteten beide Partner der von ihnen beobachteten Paare gemeinsam auf dem gleichen Abstraktionsniveau. Bei Paaren, in denen ein Partner deutlich mehr Erfahrung als der andere hatte, konnten Chong und Hurlbutt eine Dominanz des erfahreneren Partners feststellen.

Bryant [Bry04] stellt in ihrem Artikel ein Schema vor, um Interaktionen und Äußerungen von Paarprogrammierern zu protokollieren und kategorisieren. Zur Evaluation ihres Schemas führte sie eine empirische Studie durch, in der sie insgesamt 14 einstündige Paarprogrammiersitzungen in einer Softwarefirma beobachtete. Die Partner in den Paaren hatten unterschiedliche Erfahrung mit der Paarprogrammierung, was sich laut den Ergebnissen der Studie auf die Interaktionsrate der Partner auswirkte: In Paaren, in denen beide wenig Erfahrung mit der Paarprogrammierung hatten, kam es zu 27 Prozent mehr Interaktionen pro Stunde als in Paaren, in denen beide Partner viel Erfahrung mit der Paarprogrammierung hatten. Hatte ein Partner viel und der andere wenig Erfahrung mit der Paarprogrammierung, lag die Interaktionsrate dazwischen. Ein weiteres Ergebnis der Studie legt einen Zusammenhang zwischen der Erfahrung in der Paarprogrammierung, der Rolle im Paar und dem gezeigten Verhalten nahe. Bei erfahren Paaren zeigte sich ein stabiles Verhaltensmuster. Welcher der beiden Partner die Rolle des Fahrers hatte, war dabei nicht von Bedeutung. Die Partner in unerfahrenen Paaren dagegen verhielten sich generell komplett unterschiedlich auch wenn sie die gleiche Rolle innehatten.

3.2 Arbeiten zur testgetriebenen Entwicklung

Bis auf die letzte in diesem Abschnitt vorgestellte Studie von Müller und dem Autor dieser Arbeit [MH07] vergleichen alle hier besprochenen Studien die testgetriebene Entwicklung entweder mit der sogenannten *iterativen Test-Danach-Entwicklung* oder der *konventionellen Entwicklung* nach dem Wasserfall-Modell.

Die iterative Test-Danach-Entwicklung ist genau wie die testgetriebene Entwicklung ein stark iterativer Prozess und unterscheidet sich von dieser nur durch die Tatsache, dass die Testfälle nicht vor sondern kurz nach dem Anwendungscode geschrieben werden.

Die konventionelle Entwicklung nach dem Wasserfallmodell (ab hier nur noch konventionelle Entwicklung genannt) zeichnet sich dadurch aus, dass zunächst in einer Entwicklungsphase der gesamte Anwendungscode geschrieben wird, der dann in einer separaten Testphase getestet und gegebenenfalls verbessert wird.

3.2.1 Fallstudien und Erfahrungsberichte zur testgetriebenen Entwicklung

Die Fallstudie von Maximilien und Williams [MW03] befasst sich mit der Einführung der testgetriebenen Entwicklung bei IBM. Maximilien und Williams verglichen die Defektdichte eines mit Hilfe von testgetriebener Entwicklung neu implementierten Java-Systems für Peripheriegeräte im Verkauf (Kartenleser, Drucker, etc.) mit der Defektdichte des Altsystems. Das beschriebene Altsystem wurde mit einem an das Wasserfall-Modell angelehnten Ad-hoc-Prozess entwickelt, in dem die Testphase im Anschluss an die Entwicklungsphase folgte. Die Defektdichte des Altsystems lag mit 7,0 Defekten pro tausend Zeilen Code deutlich höher als die Defektdichte von 3,7 Defekten pro tausend Zeilen Code des neu entwickelten Systems. Maximilien und Williams führen diese Verbesserung allein auf die Einführung der testgetriebenen Entwicklung zurück, obwohl es auch andere Erklärungen dafür geben könnte. Erstens wurden die beiden Systeme von Teams mit unterschiedlichen Eigenschaften entwickelt. Das Team des Altsystems wird als erfahren beschrieben, die Zahl der Entwickler wird aber nicht genannt. Das Team des neuen Systems bestand nach Angaben der Autoren aus neun jungen Entwicklern, von denen viele wenig Erfahrung mit testgetriebener Entwicklung und Java hatten. Obwohl die mangelnde Erfahrung eher gegen das jüngere Team spricht, ist es denkbar, dass dieses Team, auch aufgrund eines Neuigkeitseffekts bezüglich der testgetriebenen Entwicklung, motivierter war und daher Code mit besserer Qualität abgeliefert hat. Weiter war der Umfang der Projekte nicht vergleichbar. Das neue System hatte etwa 71.000 Zeilen Anwendungscode und zusätzlich etwa 34.000 Zeilen Testcode. Das Altsystem kann aus den Zahlen im Artikel auf etwa 11.000 Zeilen Anwendungscode geschätzt werden. Über die Menge an Testcode wird keine Aussage gemacht. Allerdings wird beschrieben, dass bei dem für das Altsystem verwendeten Ad-hoc-Prozess oftmals unter Zeitdruck die Testphase entfiel und keine Tests geschrieben wurden. Insgesamt darf daher bezweifelt werden, dass alle in dieser Fallstudie beschriebenen Unterschiede tatsächlich ihren Ursprung in der Verwendung der testgetriebenen Entwicklung haben.

Bhat und Nagappan berichten in [BN06] über zwei Fallstudien bei Microsoft. Die Autoren beschreiben zwei Projekte aus zwei Abteilungen, bei denen die Test-zuerst-Entwicklung verwendet wurde. Die Projekte werden mit ähnlichen Projekten mit konventioneller Entwicklung aus dem Hause Microsoft verglichen. Die Autoren folgern, dass die Projekte zwar deutlich mehr Zeit benötigten aber dafür die Fehlerrate im Code mehr als halbieren konnten.

3.2.2 Experimente zur testgetriebenen Entwicklung

Müller und Hagner [MH02] verglichen in ihrem Experiment testgetriebene mit konventioneller Entwicklung. An dem Experiment nahmen insgesamt 19 Studenten teil: Zehn in der Gruppe, welche die testgetriebene Entwicklung verwendete und neun in der Kontrollgruppe, die einen konventionellen Entwicklungsprozess einsetzte. Die Studenten hatten vor dem Experiment alle das Extreme Programming-Praktikum absolviert, in dem sie die grundlegenden Techniken des Extreme Programmings, wie die testgetriebene Entwicklung und die Paarprogrammierung erlernt hatten. Während des Experiments mussten die Teilnehmer die Methoden der zentralen Klasse einer Graph-Bibliothek implementieren. Dabei waren die Methodenrümpfe mit Kommentaren bereits vorgegeben. Die Studenten arbeiteten an Rechnern in einem Poolraum der Universität Karlsruhe. Die verwendete Programmiersprache war Java. Für die automatischen Komponententests wurde JUnit [JUn] eingesetzt. Die Teilnehmer arbeiteten an der Implementierung der Graph-Bibliothek bis sie der Meinung waren, eine funktionierende Lösung entwickelt zu haben. Anschließend wurden ihre Programme mit einem Akzeptanztest auf ihre Korrektheit getestet. Die Arbeit eines Teilnehmers war erst beendet, wenn der Akzeptanztest bestanden wurde. Bestand das Programm eines Teilnehmers den Akzeptanztest nicht, musste er sein Programm korrigieren und anschließend erneut testen lassen. Müller und Hagner verglichen die Entwicklungszeit für die gesamte Aufgabe und bis zum Erreichen des ersten Akzeptanztests. Bei einem Signifikanzniveau von 10 Prozent konnten die Autoren keinen statistisch signifikanten Unterschied in der Entwicklungszeit nachweisen. Weiterhin verglichen Müller und Hagner die Zuverlässigkeit der Programme in der Version vor dem ersten Akzeptanztest und der endgültig abgegebenen Version. Als Maß für die Zuverlässigkeit verwendeten sie den Anteil der bestandenen Zusicherungen von allen Zusicherungen des Akzeptanztests. Dabei schnitten die Programme der Gruppe, die testgetrieben entwickelt hatte, in der Version vor dem ersten Akzeptanztest signifikant schlechter ab ($p = 0{,}03$). Die Autoren untersuchten auch die Wiederverwendung existierender Methoden. Sie maßen die Anzahl wiederverwendeter Methoden, die Anzahl fehlgeschlagener Methodenaufrufe und die Anzahl der Methodenaufrufe, die mehr als einmal fehlschlugen. Bei letzterem Maß waren die Programme der TGE-Gruppe besser als die der Kontrollgruppe ($p = 0{,}086$).

Der Artikel von Edwards [Edw03] präsentiert Web-CAT, ein System zur automatischen Benotung des Anwendungs- und Testcodes von studentischen Programmen. Intention des Web-CAT-Systems ist es, die Studenten bei Programmieraufgaben zu mehr eigenen Tests zu ermutigen. Es bewertet daher neben der eigentlichen Lösung der Studenten auch die dazu geschriebenen Testfälle. Edwards verglich die Programme von 59 Studenten des Jahrgangs 2003 eines Informatikkurses über Programmiersprachen mit

den Programmen von 59 Studenten des gleichen Kurses aus dem Jahr 2001. Die Studenten des Jahrgangs 2003 waren in testgetriebener Entwicklung ausgebildet worden, mussten diese auch einsetzen und nutzten Web-CAT zur Abgabe der Programme. Die Studenten des Jahrgangs 2001 waren weder in testgetriebener Entwicklung ausgebildet noch verpflichtet ihre Programme zu testen. Sie nutzten das zuvor eingesetzte Benotungssystem Curator, das lediglich den Anwendungscode bewertet. Edwards beobachtete, dass die Programme der Studenten des Jahrgangs 2003 durchschnittlich 45 Prozent weniger Fehler aufwiesen als die Programme der Studenten des Jahrgangs 2001. Edwards schließt daraus, dass testgetriebene Entwicklung und Web-CAT in der Lage sind, die Qualität der von Studenten geschriebenen Programme zu erhöhen. Aus dem Artikel wird allerdings nicht klar, welcher Anteil dabei der beobachteten Qualitätsverbesserung auf die Einführung der testgetriebenen Entwicklung und welcher auf die Verwendung von Web-CAT zurückzuführen ist. Edwards lässt zudem die rivalisierende Hypothese außer Acht, dass die Qualitätsverbesserung alleine darin begründet sein könnte, dass die Studenten des Jahrgangs 2003 ihre Programme testeten und die Studenten des Jahrgangs 2001 nicht.

Pančur und Kollegen [PCTV03] verglichen in ihrer Studie die testgetriebene Entwicklung mit der iterativen Test-Danach-Entwicklung. An der Studie nahmen 38 Studenten in ihrem 8. Semester teil. 20 der Teilnehmer arbeiteten gemäß der testgetriebenen Entwicklung, 18 verwendeten die iterative Test-Danach-Entwicklung. Die Zuweisung der Entwicklungstechnik erfolgte durch die Experimentleiter unter Berücksichtigung der Noten und Programmiererfahrung der Studenten. Die Daten zur Studie wurden zwischen Februar und Juni 2003 in zwei Phasen gesammelt. In der ersten Phase bearbeiteten die Teilnehmer in Paaren und vermutlich in Heimarbeit drei kürzere und eine längere Aufgabe, deren Dauer im Artikel mit ungefähr drei Wochen angegeben wird. In der zweiten Phase bearbeiteten die Teilnehmer alleine ein halbtägige Aufgabe. An dieser Phase nahmen in der Gruppe mit testgetriebener Entwicklung nur noch 19 und in der Gruppe mit iterativer Test-Danach-Entwicklung nur noch 15 Teilnehmer teil. Die Daten wurden auf die Testabdeckung und externe Code-Qualität, gemessen in der Anzahl bestandener Testfälle von insgesamt 120 Testfällen, untersucht. In beiden Fällen konnte kein signifikanter Unterschied zwischen den beiden Gruppen nachgewiesen werden.

George und Williams [GW04] berichten in ihrem Artikel über ein Experiment mit 24 professionellen Programmieren. Die Programmierer ihres Experiments hatten in Paaren das insgesamt ca. 200 Zeilen Code umfassende Bowling-Scorekeeper-Problem zu lösen. Sechs Paare arbeiteten dabei mit testgetriebener Entwicklung, sechs nach einem wasserfallähnlichen Prozess. Wo die Paare an der Aufgabe arbeiteten, wird im Artikel nicht explizit erwähnt, jedoch ist anzunehmen, dass sie in den Büroräumen ihrer Arbeitgeber arbeiteten. George und Williams geben an, dass die Paare, die testgetriebene

Entwicklung verwendeten, eine bessere Qualität lieferten: Ihre Programme bestanden 18 Prozent mehr Testfälle des Blackbox-Akzeptanztests als die der Paare, die dem Wasserfall-ähnlichen Prozess folgten. Im Gegenzug benötigten die Paare mit testgetriebener Entwicklung laut George und Williams 16 Prozent mehr Entwicklungszeit. Weiterhin berichten die Autoren von einer mittleren positiven Korrelation zwischen der benötigten Entwicklungszeit und der Qualität der Programme. Insgesamt sind die Ergebnisse dieser Studie kritisch zu sehen. Hierfür gibt es mehrere Gründe: Erstens handelte es sich zumindest bei einem Teil der Teilnehmer um Anfänger in testgetriebener Entwicklung und Paarprogrammierung. Somit lassen die Ergebnisse der Studie keine Rückschlüsse auf Entwickler mit mehr Erfahrung in diesen Techniken zu. Weiterhin findet sich im Artikel keine Angabe, die klarstellt, ob die beobachteten Unterschiede bei Qualität und Entwicklungszeit überhaupt statistisch signifikant sind. Am kritischsten für die Gültigkeit der Studie ist jedoch die Tatsache, dass die Experimentbedingungen geändert wurden, nachdem bereits ein Drittel der Teilnehmer ihre Datenpunkte abgeliefert hatten. Die geänderten Bedingungen sahen vor, den verbleibenden 16 Teilnehmern die Testfälle des Akzeptanztests nicht mehr zur Verfügung zu stellen und eine bessere Fehlerbehandlung für die Programme zu fordern. Eine weitere Änderung war, dass die Teilnehmer in der Kontrollgruppe mit einem wasserfallähnlichen Prozess nun gebeten wurden, ebenfalls automatische Unit-Tests zu schreiben.

Geras und Kollegen verglichen in [GSM04] die testgetriebene Entwicklung mit der iterativen Test-Danach-Entwicklung. An dem in dieser Studie beschriebenen Experiment nahmen pro Gruppe sieben professionelle Entwickler teil. Aufgrund der geringen Teilnehmerzahl gab es keine statistisch signifikanten Ergebnisse.

Eines der aktuelleren Experimente zum Thema testgetriebene Entwicklung stammt von Erdogmus und Kollegen [EMT05]. Ziel des Experiments war ein Vergleich der testgetriebenen Entwicklung mit iterativer Test-Danach Entwicklung bezüglich Qualität und Produktivität. Die Teilnehmer dieses Experiments waren 35 Informatikstudenten im sechsten Semester, die an einem achtwöchigen Java-Kurs der Technischen Universität Turin[5] teilgenommen hatten. Elf Teilnehmer schieden jedoch vorzeitig aus, so dass am Ende die Datenpunkte von 24 Teilnehmern zur Verfügung standen. Davon befanden sich 11 in der Test-Zuerst-Gruppe und 13 in der Test-Danach-Gruppe. Die Zuweisung der Teilnehmer zu den Gruppen erfolgte durch die Autoren. Vermutlich war die Zuweisung auch zufällig, was jedoch nie explizit erwähnt wird. Als Aufgabe wurde das gleiche Bowling-Scorekeeper-Problem wie bei George und Williams [GW04] verwendet. Allerdings wurde die Aufgabe von den Autoren in mehrere Teilaufgaben unterteilt, welche einzeln abgegeben werden durften. Die Autoren verglichen die beiden Gruppen bezüglich ihrer Produktivität, der Qualität der abgegebenen Programme und der An-

[5] Politecnico di Torino

zahl geschriebener Testfälle pro Mannstunde. Als Maß für die Produktivität wurde die Anzahl gelieferter Teilaufgaben eines Teilnehmers pro Mannstunde definiert. Eine Teilaufgabe galt dabei als geliefert, sobald sie die Hälfte der Testfälle des dazugehörigen Akzeptanztests bestand. Als Maß für die Qualität wurde für jede abgegebene Teilaufgabe eines Teilnehmers zunächst der Anteil der bestanden Testfälle des dazugehörigen Akzeptanztests ermittelt. Daraus wurde das gewichtete Mittel gebildet, wobei die geschätzten Schwierigkeitsgrade der Teilaufgaben als Gewichte dienten. Erdogmus und Kollegen fanden heraus, dass die Teilnehmer der Test-Zuerst-Gruppe signifikant mehr Tests pro Mannstunde geschrieben hatten als die der Test-Danach-Gruppe ($p = 0{,}09$, $\alpha = 0{,}1$). Bei der Qualität und Produktivität gab es jedoch keine signifikanten Unterschiede ($p = 0{,}25$ bzw. $0{,}48$). Weiterhin untersuchten Erdogmus und Kollegen die Daten auf eine Korrelation zwischen der Anzahl geschriebener Testfälle pro Mannstunde und der Produktivität sowie zwischen der Anzahl geschriebener Testfälle pro Mannstunde und der Qualität. Diese Betrachtung ergab, dass die Produktivität linear von der Anzahl Testfälle pro Mannstunde abhängt (Spearmans $r = 0{,}587$, $p = 0{,}003$). Eine Korrelation zwischen der Anzahl Testfälle pro Mannstunde und der Qualität konnte nicht nachgewiesen werden.

Canfora und Kollegen untersuchten in ihrem Experiment [CCG+06] die Unterschiede zwischen testgetriebener und konventioneller Entwicklung bezüglich Produktivität der Entwickler und Qualität der Komponententests. Die Produktivität der Entwickler wurde mit drei Maßen gemessen: der mittleren Zeit für das Schreiben und Ausführen einer Zusicherung, der mittleren Zeit für das Schreiben und Ausführen einer Testsuite sowie Bearbeitungszeit für eine Aufgabe. Als Maße für die Qualität der Komponententests wurden die mittlere Anzahl Zusicherungen pro Methode im Anwendungscode und die Summe der Zusicherungen betrachtet. An dem Experiment nahmen insgesamt 28 professionelle Entwickler einer spanischen Softwarefirma teil. Der Aufbau des Experiments war gegenbalanciert, so dass jeder Entwickler zwei Datenpunkte lieferte. Die Teilnehmer hatten fünf Jahre Erfahrung mit der im Experiment verwendeten Programmiersprache Java, jedoch keine Erfahrung in testgetriebener Entwicklung. Daher wurden die Teilnehmer von den Autoren vor dem Experiment in dieser Technik geschult. Bei der Produktivität schnitt die testgetriebene gegenüber der konventionellen Entwicklung bei allen drei Maßen signifikant schlechter ab. Bei der Qualität der Komponententests konnte kein signifikanter Unterschied nachgewiesen werden.

Wenn wir mithilfe von Tabelle 3.1 alle bisher in diesem Abschnitt vorgestellten Experimente zur testgetriebenen Entwicklung zusammenfassend betrachten, werden zwei Dinge deutlich: Erstens wurden alle Experimente mit in der testgetriebenen Entwicklung unerfahrenen Teilnehmern durchgeführt. Zweitens wurde nur in zwei Experimenten durch simples Befragen der Teilnehmer die Prozesstreue geprüft. Einige Forscher

haben das Problem zwar diskutiert und geben mangelnde Prozesstreue als Gefahr für die Gültigkeit an, haben aber nicht dementsprechend gehandelt und die Prozesstreue geprüft. Gerade bei mit der testgetriebenen Entwicklung unerfahrenen Entwicklern, wie sie auch in unseren Studien vorkommen, ist es aber wichtig, auf die Prozesstreue zu achten: Gerade bei diesen Entwicklern besteht die Gefahr, dass Sie nicht in der Lage sind, den geforderten Entwicklungsprozess umzusetzen. Eine Überprüfung der Prozesstreue per Selbstauskunft der Teilnehmer ist ebenfalls unzureichend, da sie sich vermutlich positiv darstellen wollen und aber auch mangels Erfahrung selbst davon überzeugt sein können, den Prozess korrekt umgesetzt und befolgt zu haben.

Dass Erfahrung bei der testgetriebenen Entwicklung eine große Rolle spielt, zeigt eine in Zusammenarbeit von Müller und dem Autor dieser Arbeit [MH07] durchgeführte Studie. Um den Einfluss der Erfahrung auf den testgetriebenen Entwicklungsprozess zu untersuchen, wurden in einem Quasi-Experiment die testgetriebenen Entwicklungsprozesse von erfahrenen und unerfahrenen Entwickler verglichen. Die Gruppe der erfahrenen Entwickler bestand aus sieben professionellen Entwicklern, die der unerfahren Programmierer aus elf Informatik-Studenten, die zum Zeitpunkt der Studie im Mittel im vierten Semester waren. Die professionellen Entwickler hatten durchschnittlich 3,4 Jahre Berufserfahrung mit testgetriebener Entwicklung während alle bis auf einen Studenten die testgetriebene Entwicklung gerade erst im Extreme Programming-Praktikum erlernt hatten. Alle Teilnehmer der Studie mussten den Tür-Offen-Zustand einer in Java geschriebenen Fahrstuhlsteuerung implementieren. Für die Komponententests wurde JUnit verwendet. Die Entwicklungsumgebung war Eclipse. Die Daten über die testgetriebenen Entwicklungsprozesse der Teilnehmer wurden mithilfe des Vorgängerwerkzeugs von TestPulse gesammelt und analysiert. Die Autoren konnten zeigen, dass sich die erfahrenen Entwickler besser an die Regeln der testgetriebenen Entwicklung hielten und die Testfälle in kürzeren Abständen ausführten als die unerfahrenen Entwickler. Zudem erreichten die Komponententests der erfahrenen Entwickler eine höhere Anweisungs- und Blockabdeckung als die der unerfahrenen Entwickler. Diese Studie war Grundlage für die in Kapitel 4 vorgestellte Studie, in welcher der Einfluss der Erfahrung auf den testgetriebenen Entwicklungsprozess von Programmiererpaaren untersucht wurde.

3.3 Werkzeuge zur Untersuchung der testgetriebenen Entwicklung

In Abschnitt 2.3 dieser Arbeit wurde unser Werkzeug zur Untersuchung der testgetriebenen Entwicklung vorgestellt. Neben unserem Werkzeug finden sich in der in Literatur

Studie	*Teilnehmer*	*Erfahrung mit TGE*	*Prozess in KG*	*Unterschiede zur KG*	*Prozesstreue*	
					disk.?	*geprüft?*
[GW04]	2 × 12 Entw.	gering (3 Wochen) bis hoch	Wasserfall	18 % mehr erfolgreiche Testfälle, 16 % längere Dauer	ja	nein
[GSM04]	14 Entw.	keine Angabe	ITD	keine gefunden	nein	nein
[CCG+06]	28 Entw.	keine, Einführung vor Experiment	Wasserfall	längere Dauer	nein	nein
[MH02]	19 Stud.	1 Semester im Rahmen eines XP-Praktikums	Wasserfall	bessere Wiederverwendung existierender Methoden	ja	ja (Nachfrage)
[PCTV03]	38 Stud.	keine Angabe	ITD	keine gefunden	nein	nein
[Edw03]	118 Stud.	keine, 30 min. Einführung in Vorlesung	Wasserfall	45 % weniger Fehler	nein	nein
[EMT05]	24 Stud.	keine, nur 1 Semester JUnit	ITD	bessere Produktivität	ja	ja (Nachfrage)

Tabelle 3.1: Experimente zur testgetriebenen Entwicklung.

weitere vergleichbare Ansätze:

Wege [Weg04] stellt in seiner Dissertation den TDDMentor vor, ein Werkzeug, das basierend auf Einbuchungen in das Versionskontrollsystem CVS die Prozesstreue gegenüber der testgetriebenen Entwicklung beurteilt. Das Werkzeug verwendet statische Analyse, um die Änderungen an Methoden im Anwendungscode von einer Version zur nächsten zu erkennen. Die Änderungen werden als prozesskonform eingestuft, sofern sie an Methoden stattfanden, die durch einen JUnit-Testfall abgedeckt sind oder es sich um eine Umstrukturierung wie z. B. das Umbenennen einer Methode handelt. Der von Wege gewählte Ansatz hat die grundsätzliche Schwäche, zu grobgranular für die zuverlässige Bestimmung der Prozesstreue gegenüber der testgetriebenen Entwicklung zu sein. Obwohl die Integration neu geschriebenen Codes in das Versionskontrollsystem bei testgetriebener Entwicklung häufig geschehen soll [Bec05, S. 49 ff.], wird sie dennoch nicht im Minutentakt stattfinden. Die Tests sollen aber bei der testgetriebenen Entwicklung im Abstand von wenigen Minuten ausgeführt werden. Selbst wenn die Entwickler häufig genug, also nach jedem TGE-Zyklus, in das Versionskontrollsystem einbuchen würden, ließe sich nicht feststellen, ob die Tests für diese TGE-Zyklen tatsächlich vor dem Anwendungscode geschrieben wurden. Weges TDDMentor kann folglich nur überprüfen, ob der Testcode zeitnah zum Anwendungscode geschrieben wird, nicht aber, ob wirklich testgetrieben gearbeitet wird.

Das von Wang und Erdogmus [WE04] vorgestellte Werkzeug TestFirstGauge nutzt Hackystat [Hac], um Daten über den testgetriebenen Entwicklungsprozess zu sammeln. Das Werkzeug sammelt Daten über JUnit-Testläufe, die Namen und die Anzahl der geänderten Zeilen Code, der bearbeiteten Dateien und wie viel Zeit mit der Bearbeitung einer Datei verbracht wurde. Die gesammelten Daten werden sogenannten Programmierzyklen zugeordnet. Die Programmierzyklen werden dann nach Gemeinsamkeiten und Mustern untersucht um Rückschlüsse über die Arbeitsweise des jeweiligen Programmierers ziehen zu können. Ihr Werkzeug ist in der Lage, aus den gesammelten Daten die Länge der Programmierzyklen und deren Verteilung zu ermitteln und als Diagramm darzustellen. Auch kann das Verhältnis zwischen der Anzahl der Zeilen Test- und Anwendungscode sowie des Aufwandes für Test- und Anwendungscode angezeigt werden. Das Werkzeug kann jedoch im Gegensatz zu unserem Werkzeug nicht die Prozesstreue gegenüber der testgetriebenen Entwicklung bestimmen und wurde bisher nicht in einem Experiment eingesetzt.

Ebenso wie das zuvor vorgestellte Werkzeug nutzt Zorro [KJ06, Kou07, JK07] Hackystat, um die Aktivitäten des Entwicklers bei der testgetriebenen Entwicklung zu beobachten. Zorro sammelt Daten über Aktivitäten wie die Ausführung von Testfällen, Übersetzungsvorgänge und das Editieren von Test- und Anwendungscode. Die einzelnen Aktivitäten werden vor der Analyse mit Zorro von einer Zwischenschicht (Software Deve-

lopment Stream Analysis genannt) zunächst nach ihrem Zeitstempel zu einer Sequenz geordnet. Diese Sequenz wird danach in Episoden unterteilt, welche Zorro dann bezüglich der Prozesstreue gegenüber der testgetriebenen Entwicklung beurteilt. Zorro kennt 22 unterschiedliche Typen von Episoden, zusammengefasst in acht Kategorien, wie z. B. *Test-Zuerst*, *Umstrukturierung*, *Test-Danach* und *Lang*[6]. Jede Episode wird dann von Zorro als testgetrieben oder nicht testgetrieben klassifiziert. Bei der Hälfte der 22 Episodentypen ist die Typinformation ausreichend, um die ihnen zugeordneten Episoden zu klassifizieren. Episoden, die den verbleibenden 11 Episodentypen zugeordnet wurden, werden mittels einer Heuristik, welche die benachbarten Episoden mitbetrachtet, klassifiziert. Die Heuristik entscheidet optimistisch, d. h. wenn nur eine der benachbarten Episoden bereits als testgetrieben klassifiziert wurde, wird die fragliche Episode ebenfalls als testgetrieben klassifiziert. Die Genauigkeit der Episodenklassifizierung wurde in zwei Fallstudien mit Studenten evaluiert [KJE10]. In beiden Studien wurden die Teilnehmer bei der Bearbeitung sehr kleiner Programmieraufgaben in Java mit Eclipse und JUnit beobachtet. Verglichen wurde die Episodenklassifizierung Zorros mit der, welche die Autoren aufgrund von Bildschirmmitschnitten der Programmiersitzungen der Teilnehmer manuell durchgeführt hatten. Dabei wurden in der ersten Fallstudie mit sieben Teilnehmern durchschnittlich 89 Prozent der Episoden korrekt klassifiziert. In der zweiten Fallstudie mit zehn Teilnehmern waren es im Durchschnitt 70 Prozent. Allerdings gab es in dieser Fallstudie mit nur 52, 44 und 22 Prozent korrekt klassifizierter Episoden drei extreme Ausreißer, welche die Autoren hauptsächlich damit erklären, dass die entsprechenden drei Teilnehmer Testläufe in Eclipse starteten, obwohl der Code nicht übersetzbar war, was Zorro nicht einordnen kann.

Neben der Tatsache, dass Zorro wenigstens rudimentär evaluiert wurde, ist eine weitere Stärke von Zorro, dass es, im Gegensatz zu unserem System TestPulse, nicht nur für Java sondern auch für C# funktioniert. Zorro hat aber auch Schwächen: Eine wesentliche ist, dass Zorro nicht in der Lage ist, nachzuvollziehen, ob im Anwendungscode geänderte Methoden tatsächlich durch einen fehlschlagenden Testfall abgedeckt sind. Zorro wertet lediglich die Information aus, ob vor einer Änderung im Anwendungscode irgendein Testfall fehlschlägt. Bei erfahrenen Entwicklern, die sich gewissenhaft an die Regeln der testgetriebenen Entwicklung halten, mag diese Information ausreichend sein. Gerade Anfänger in der testgetriebenen Entwicklung neigen jedoch gelegentlich dazu, chaotisch vorzugehen. Dabei kann es durchaus vorkommen, dass zwar Testfälle aus den verschiedensten Gründen fehlschlagen, aber anschließend eben nicht der Anwendungscode geschrieben wird, um diese Testfälle zu erfüllen, sondern an ganz anderer Stelle weitergearbeitet wird. Eine weitere Schwäche von Zorro verbirgt sich hinter Zorros Metrik zur Bewertung des testgetriebenen Entwicklungsprozesses. Zorro gibt die

[6] Im Artikel [KJE10] heißen diese Episodentypen *Test First*, *Refactoring*, *Test Last* und *Long*.

Prozesstreue als Anteil der als testgetrieben eingeordneten Episoden von allen Episoden an. Die Episoden selbst können jedoch von sehr unterschiedlicher Länge sein. So gibt es einen speziellen, nicht testgetriebenen Episodentyp *Lang*, der mindestens 200 Aktivitäten umfasst. Dagegen umfasst keine Episode der Kategorie *Test-Zuerst* mehr als sechs Aktivitäten. Ein Entwickler der eine solche extrem lange Episode und eine Episode aus der Test-Zuerst-Kategorie abliefert, würde von Zorro einen Wert für die Prozesstreue von 50 Prozent erhalten. In Wirklichkeit waren jedoch höchstens sechs von insgesamt über 206 Aktivitäten prozesskonform. Ein Wert von etwa 3 Prozent wäre daher in diesem Beispiel angebrachter. Dieses Problem konnte übrigens bei der zuvor angesprochenen Evaluierung in den Fallstudien nicht negativ auffallen, da die Autoren ihre manuelle Klassifizierung auch auf Basis der Episoden durchführten.

Kapitel 4

Effekt der Erfahrung auf die testgetriebene Entwicklung im Paar

Im Extreme Programming ist es eine Aufgabe der Paarprogrammierung, die testgetriebene Entwicklung zu unterstützen. So soll der gerade nicht an Tastatur und Maus arbeitende Entwickler seinen Partner daran erinnern, Änderungen im Anwendungscode stets durch einen fehlschlagenden Testfall zu motivieren oder ihm helfen, Lücken in der Testabdeckung zu finden [Bec00]. Nachdem in [MH07] bereits Unterschiede zwischen den testgetriebenen Entwicklungsprozessen von erfahrenen und unerfahren Entwicklern nachgewiesen werden konnten, präsentiert dieses Kapitel eine Studie, welche die Unterschiede zwischen den testgetriebenen Entwicklungsprozessen von erfahrenen und unerfahrenen Entwicklerpaaren untersucht. Der unserer Studie zugrunde liegende Datensatz wurde bereits in Teilen [Höf08] als auch in dem hier präsentierten vollen Umfang ausgewertet und die Ergebnisse in [HP09] und [Höf10] veröffentlicht. Einige der hier ausgeführten Gedankengänge sind ebenfalls diesen Veröffentlichungen entlehnt. Nichtsdestotrotz ist die Auswertung des Datensatz speziell für diese Arbeit neu durchgeführt worden, damit diese Studie von den seither verbesserten Auswertungsmöglichkeiten von TestPulse profitieren kann.

4.1 Verfeinerung der Forschungshypothese FH_1

Motiviert wird diese Studie durch die folgende Forschungshypothese:

FH_1 Paare aus erfahrenen Entwicklern unterscheiden sich von Paaren aus unerfahren Entwicklern bezüglich ihres testgetriebenen Entwicklungsprozesses, ihrer Produktivität und der Qualität der Programme.

Allerdings ist diese Forschungshypothese zu allgemein formuliert, um mit Hypothesentests geprüft werden zu können. Daher übertragen wir die Forschungshypothese auf die

in Abschnitt 2.4 zum testgetriebenen Entwicklungsprozess, der Produktivität und der Qualität vorgestellten Metriken, um die Alternativhypothesen für die Hypothesentests abzuleiten. Um diesen Vorgang leichter nachvollziehbar zu machen, haben wir die Metriken in der ersten Spalte von Tabelle 4.1 nochmals aufgelistet. Wie man dort sehen kann, handelt es sich um insgesamt elf Metriken. Deswegen wollen wir nicht jede Alternativhypothese einzeln aufführen, sondern formulieren zunächst die folgende Schablone für die Alternativhypothesen:

$\mathbf{H_A^{[Abk.\ Metrik]}}$ Paare aus erfahrenen Entwicklern unterscheiden sich von Paaren aus unerfahren Entwicklern bezüglich *[ihrer/ihres] [Metrik]*.

Durch Einsetzen der Metriken lassen sich aus dieser Schablone die Alternativhypothesen erstellen, wie z. B.:

$\mathbf{H_A^{TGE}}$ Paare aus erfahrenen Entwicklern unterscheiden sich von Paaren aus unerfahren Entwicklern bezüglich ihrer TGE-Prozesstreue.

In der letzten Spalte von Tabelle 4.1 finden sich die Kurzschreibweisen aller Alternativhypothesen. Diese stehen dabei stets unter der Kurzschreibweisen der entsprechenden Nullhypothesen, welche sich durch logische Negation der Alternativhypothesen ergeben. Vor jeder der Kurzschreibweisen findet sich ein innerhalb dieses Kapitels eindeutiger Schlüssel über den wir die entsprechende Hypothese referenzieren werden. Innerhalb der Kurzschreibweisen ist der Name der Metrik durch eine korrespondierende Abkürzung ersetzt. Diese Abkürzung findet sich auch als hochgestellter Text im Schlüssel der Hypothese wieder. Die Indizes in der Kurzschreibweise stehen für die in der Studie betrachteten Gruppen. Der Index EXP steht dabei für die Experten-Gruppe, die Indizes A06 und A08 für die Anfänger_{06}- und Anfänger_{08}-Gruppe.

Bleibt zum Schluss dieses Abschnitts noch die Frage zu klären, warum in den Kurzschreibweisen der Hypothesen drei Gruppen vorkommen, obwohl, geht man nach den ausformulierten Versionen, nur zwei Gruppen nötig sind. Der Grund hierfür ist, dass wir die Studie ursprünglich für zwei Gruppen, nämlich eine Experten- und eine Anfänger-Gruppe angelegt hatten. Die Paare der beiden Anfänger-Gruppen sollten hierbei zusammengefasst werden. Jedoch zeigten sich zwischen den beiden Anfänger-Gruppen Unterschiede bei den Vorkenntnissen. Deswegen entschieden wir uns, die Gruppen weiterhin getrennt zu betrachten und die beiden Anfänger-Gruppen jeweils einzeln mit der Experten-Gruppe und schließlich, aufgrund der Unterschiede bei den Vorkenntnissen auch untereinander zu vergleichen.

Weitere Details zu den Unterschieden zwischen den Anfänger-Gruppen werden in den nächsten Abschnitten folgen.

Metrik	*Null- & Alternativhypothese*	
	Schl.	*Kurzschreibweise*
Kategorie: TGE-Prozess		
TGE-Prozesstreue [%]	H_0^{TGE}	$TGE_{EXP} = TGE_{A06} = TGE_{A08}$
	H_A^{TGE}	$\neg(TGE_{EXP} = TGE_{A06} = TGE_{A08})$
Anzahl der automatischen Umstrukturierungen	H_0^{AU}	$AU_{EXP} = AU_{A06} = AU_{A08}$
	H_A^{AU}	$\neg(AU_{EXP} = AU_{A06} = AU_{A08})$
Mittlerer Abstand der Testläufe [min]	H_0^{MAT}	$MAT_{EXP} = MAT_{A06} = MAT_{A08}$
	H_A^{MAT}	$\neg(MAT_{EXP} = MAT_{A06} = MAT_{A08})$
Variationskoeffizient des Abstands der Testläufe [%]	H_0^{VAT}	$VAT_{EXP} = VAT_{A06} = VAT_{A08}$
	H_A^{VAT}	$\neg(VAT_{EXP} = VAT_{A06} = VAT_{A08})$
Kategorie: Produktivität		
Bearbeitungszeit [min]	H_0^{BZ}	$BZ_{EXP} = BZ_{A06} = BZ_{A08}$
	H_A^{BZ}	$\neg(BZ_{EXP} = BZ_{A06} = BZ_{A08})$
Netto-Änderungen [SLOC]	$H_0^{NÄ}$	$NÄ_{EXP} = NÄ_{A06} = NÄ_{A08}$
	$H_A^{NÄ}$	$\neg(NÄ_{EXP} = NÄ_{A06} = NÄ_{A08})$
Anteil der Netto-Änderungen im Testcode [%]	H_0^{TC}	$TC_{EXP} = TC_{A06} = TC_{A08}$
	H_A^{TC}	$\neg(TC_{EXP} = TC_{A06} = TC_{A08})$
Anzahl der geänderten Dateien	H_0^{GD}	$GD_{EXP} = GD_{A06} = GD_{A08}$
	H_A^{GD}	$\neg(GD_{EXP} = GD_{A06} = GD_{A08})$
Kategorie: Qualität		
Änderung der Anzahl der Testfälle	$H_0^{\Delta TF}$	$\Delta TF_{EXP} = \Delta TF_{A06} = \Delta TF_{A08}$
	$H_A^{\Delta TF}$	$\neg(\Delta TF_{EXP} = \Delta TF_{A06} = \Delta TF_{A08})$
Änderung der Anweisungs-abdeckung [%]	$H_0^{\Delta AÜ}$	$\Delta AÜ_{EXP} = \Delta AÜ_{A06} = \Delta AÜ_{A08}$
	$H_A^{\Delta AÜ}$	$\neg(\Delta AÜ_{EXP} = \Delta AÜ_{A06} = \Delta AÜ_{A08})$
Anzahl der Akzeptanz-testläufe	H_0^{AT}	$AT_{EXP} = AT_{A06} = AT_{A08}$
	H_A^{AT}	$\neg(AT_{EXP} = AT_{A06} = AT_{A08})$

Tabelle 4.1: Übersicht über die Hypothesen.

4.2 Versuchsaufbau

Um die Hypothesen überprüfen zu können, haben wir ein Quasi-Experiment mit einem Nachtest gemäß dem Inter-Subjekt-Entwurf geplant (vgl. [SCC02, S. 116]). Inter-Subjekt-Entwurf mit einem Nachtest bedeutet dabei, dass die Teilnehmer jeweils nur in einer Gruppe einen Datenpunkt abliefern. Die Gruppen sind folglich komplett disjunkt. Von einem Quasi-Experiment spricht man, wenn die Teilnehmer, im Gegensatz zum Experiment, nicht zufällig den Gruppen zugewiesen werden. In unserem Fall entschieden Programmiererfahrung und die Herkunft der Teilnehmer über deren Gruppenzugehörigkeit. Normalerweise sind Experimente Quasi-Experimenten vorzuziehen, da bei Experimenten Selektionseffekte bei der Gruppeneinteilung ausgeschlossen werden können. Nach Shadish und Kollegen ist ein Quasi-Experiment jedoch angebracht, wenn es nicht möglich ist, die Teilnehmer zufällig auf Gruppen zu verteilen, da die unabhängige Variable nicht manipuliert werden kann [SCC02, S. 276]. Auf unsere unabhängige Variable, die Programmiererfahrung der Paare, trifft genau dies zu. Die abhängigen Variablen dieses Quasi-Experiments sind die in Tabelle 4.1 zuvor aufgelisteten Metriken.

4.3 Versuchsdurchführung

Insgesamt wurden Datenpunkte von 23 Programmiererpaaren gesammelt. Die ersten neun Datenpunkte der Anfänger_{06}-Gruppe wurden im Jahr 2006 von den 18 Teilnehmern des Extreme Programming-Praktikums gesammelt. Die sieben Datenpunkte der Anfänger_{06}-Gruppe stammen von den 14 Teilnehmern des Extreme Programming-Praktikums 2008. Die verbleibenden sieben Datenpunkte der Experten-Gruppe wurden ebenfalls 2008 gesammelt und stammen von 14 professionellen Softwareentwicklern. Da der Versuch bei den beiden Anfänger-Gruppen identisch ablief, folgt an dieser Stelle zunächst die Ablaufbeschreibung für diese beiden Gruppen, bevor wir die Abweichungen im Ablauf bei der Experten-Gruppe erläutern.

Die Vorbereitungen für den Versuch begannen in den Anfänger-Gruppen bereits am letzten Tag der Praktika. An diesem Tag bekamen die Teilnehmer Zettel ausgehändigt, auf denen sie ihre drei bevorzugten Programmierpartner angeben sollten. Basierend auf den dort geäußerten Präferenzen der Teilnehmer wurden vom Experimentleiter die Paare zusammengestellt. Dabei konnte nur ein Paar nicht gemäß den Präferenzen der Teilnehmer gebildet werden. Innerhalb der ersten zwei Wochen nach den Extreme Programming-Praktikum wurden die Paare einzeln zu einer Programmiersitzung mit Videoaufzeichnung in das Büro des Experimentleiters eingeladen. Dort fanden sie einen wie in Abbildung 4.1 skizzierten Arbeitsplatz vor. Dieser Aufbau mit Kameras ermöglichte es uns, Daten über die Interaktion der Partner zu sammeln. Ein Teil des Videomaterials

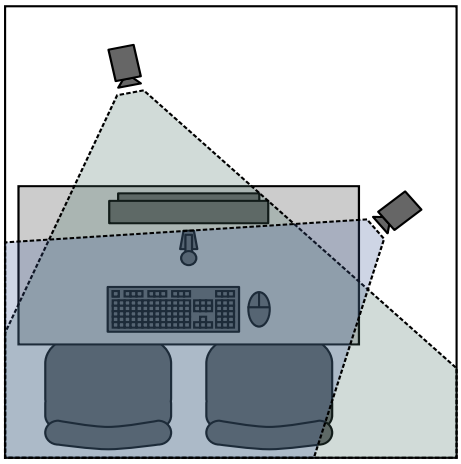


Abbildung 4.1: Der Arbeitsplatz der Paare.

wurde ausgewertet und die Ergebnisse in [Höf08] veröffentlicht. Für diese Arbeit wurden die Videos jedoch nicht weiter ausgewertet, da der Erkenntnisgewinn im Vergleich zum enormen Aufwand der Videoauswertung gering ausfiel.

Der Grund, aus dem wir den Aufbau dennoch hier erwähnen, ist, dass dadurch eine Umgebung entsteht, die sicher nicht dem gewohnten Arbeitsumfeld der Teilnehmer entspricht. Des Weiteren war durch die Aufnahmekapazität der eingesetzten Kameras implizit ein Zeitlimit für die Bearbeitung der Aufgabe von maximal sieben Stunden gegeben. Zusätzlich fand sich in den Experimentunterlagen ein Hinweis, dass sich die Aufgabe in ca. vier bis fünf Stunden lösen ließe. Diesen Umstand sowie den Einfluss der Kameras auf die Teilnehmer werden wir in Abschnitt 4.7.4 diskutieren.

Sobald die Teilnehmer sich am zuvor in Abbildung 4.1 abgebildeten Arbeitsplatz eingerichtet hatten, wurde ihnen zunächst eine Vertraulichkeitserklärung vorgelegt. Diese garantierte den Teilnehmern den Schutz ihrer Daten und deren anonyme Veröffentlichung. Die Teilnehmer sicherten mit ihrer Unterschrift zu, nicht mit Dritten über den Inhalt der Studie zu sprechen. Hatten sie die Vertraulichkeitserklärung unterschreiben, erhielten sie den Teilnehmerfragebogen, der die Vorkenntnisse der Teilnehmer erfragte. Nachdem sie diesen Fragebogen ausgefüllt hatten, wurde ihnen die Aufgabenbeschreibung ausgeteilt und die Videoaufzeichnung gestartet. In der Aufgabenbeschreibung gab es eine Tabelle zur Zeiterfassung. Jeder Teilnehmer musste dort Unterbrechungen, wie z. B. Kaffee- oder Toilettenpausen eintragen.

Jedes der Paare musste die in Abschnitt 4.5 näher beschriebene Aufgabe bearbeiten. Sobald die Partner der Meinung waren, eine fehlerfreie Lösung entwickelt zu haben, konnten sie den Experimentleiter um die Durchführung des automatischen Akzeptanztests bitten. Da sie den Akzeptanztest beim ersten Versuch bestehen sollten, wurden Sie zunächst gefragt, ob sie sich wirklich sicher waren, eine fehlerfreie Lösung gefunden zu

haben. Wurde diese Frage klar bejaht, wurde ein Akzeptanztest durchgeführt. Anderenfalls wurden die Partner gebeten ihr Programm nochmals zu testen.

Bestand die Lösung des Paares den Akzeptanztest, erhielten die Teilnehmer einen Fragebogen, in dem sie unter anderem zu ihren Eindrücken von der Studie und möglichen Problemen während des Programmierens befragt wurden. Falls der Akzeptanztest nicht bestanden wurde, forderte der Experimentleiter die Teilnehmer auf, die Fehler zu korrigieren und sich wieder zu melden, sobald sie alle Fehler entfernt hatten. Dies wurde so lange wiederholt, bis die Lösung den Akzeptanztest bestand oder das Paar angesichts des Zeitlimits von sieben Stunden aufgab. Pro Anfänger-Gruppe gab es jeweils ein Paar, das nicht in der Lage war, eine Lösung zu entwickeln, die den Akzeptanztest bestand. Die Datenpunkte dieser Paare wurden aus dem Datensatz entfernt. Zudem mussten wir die Datenpunkte von zwei weiteren Paaren aus der Anfänger_{06}-Gruppe entfernen, weil ihre Protokolle aufgrund technischer Probleme mit den Eclipse-Plugins von TestPulse irreparabel beschädigt wurden. Somit verblieben pro Anfänger-Gruppe auswertbare Datenpunkte von sechs Paaren.

Für die Paare in der Experten-Gruppe wurde der Versuch soweit irgendwie möglich wie in den Anfänger-Gruppen durchgeführt. Dennoch gab es einige wenige unvermeidliche Unterschiede zu den Anfänger-Gruppen. Der wichtigste ist wohl, dass die Teilnehmer der Experten-Gruppe für ihre Arbeit bezahlt wurden. Bis auf einen Teilnehmer wurden alle Teilnehmer dieser Gruppe von einer Softwarefirma zur Verfügung gestellt, die sich auf agile Softwareentwicklung spezialisiert hat. Sie durften während ihrer Arbeitszeit teilnehmen und wurden dafür von ihrem Arbeitgeber normal bezahlt. Ein weiterer Entwickler nahm in seiner Freizeit am Quasi-Experiment teil und wurde von uns für seinen Aufwand entschädigt.

Die Paareinteilung mussten wir aus praktischen Erwägungen dem Chef der Softwarefirma überlassen. Er versprach zwar, auf die Wünsche seiner Untergebenen Rücksicht zu nehmen, sagte aber auch, dass deren Terminplan im Zweifel Vorrang habe. Der Entwickler, der in seiner Freizeit teilnahm wurde dabei einem Entwickler aus der Softwarefirma zugeteilt, mit dem er bereits früher zusammen gearbeitet hatte.

Ein weiterer Unterschied betrifft den Arbeitsplatz der Teilnehmer. Nur das Paar mit dem einzigen Teilnehmer, der nicht bei der besagten Softwarefirma angestellt war, arbeitete wie die Teilnehmer der Anfänger-Gruppen im Büro des Experimentleiters. Die verbleibenden Experten-Paare arbeiteten an einem äquivalenten Arbeitsplatz in einem Konferenzraum ihrer Softwarefirma.

Schließlich gab es noch einen Unterschied zwischen dem Teilnehmerfragebogen der Anfänger-Gruppen und dem der Experten-Gruppe. Die Version der Experten war um die Fragen zur Programmiererfahrung im privaten Bereich gekürzt worden. Der Grund hierfür war, dass wir vom Chef der Softwarefirma darauf hingewiesen wurden, dass einige

Paare der Experten-Gruppe eventuell nur sechs bis sieben Stunden für den Versuch zur Verfügung stehen könnten. Daher wollten wir die Zeit für das Ausfüllen von sonstigen Dokumenten minimieren und haben den Block mit Fragen zur allgemeinen Programmiererfahrung für die Experten gestrichen und nur nach der industriellen Programmiererfahrung gefragt. Wir taten dies in der Annahme, dass die Experten den bei weitem größten Teil ihrer Programmiererfahrung im industriellen Umfeld gesammelt haben. Der Preis hierfür ist jedoch, dass ein Vergleich der allgemeinen Programmiererfahrung nicht möglich war. Folglich wird dies im nächsten Abschnitt auch nicht diskutiert.

Im Übrigen waren die Versuchsbedingungen für die Experten die gleichen wie für die Anfänger. Insbesondere mussten auch ihre Lösungen den zuvor angesprochen Akzeptanztest bestehen. Wie schon in den Anfänger-Gruppen, gab es dabei auch unter den Experten ein Paar, das keine Lösung entwickeln konnte, die den Akzeptanztest bestand. Auch deren Datenpunkt wurde aus dem Datensatz entfernt. Somit verblieben im gesamten Datensatz drei Gruppen mit je sechs Datenpunkten. Sofern dies nicht ausdrücklich anderes erwähnt wird, beziehen sich alle Angaben in den folgenden Abschnitten des Kapitels sowie im dazugehörigen Anhang A auf die Teilnehmer, deren Datenpunkte im Datensatz belassen wurden.

4.4 Beschreibung der Teilnehmer

Wie bereits im vorigen Abschnitt erwähnt, handelt es sich bei den Teilnehmern in den Gruppen Anfänger_{06} und Anfänger_{08} um Studenten aus den Extreme Programming-Praktika der Jahre 2006 und 2008. Ursprünglich war geplant, die Teilnehmer dieser Gruppen zu einer einzigen Anfänger-Gruppe zusammenzufassen. Bei der Auswertung der Teilnehmer-Fragebögen zeigten sich jedoch in einigen für den Versuch relevanten Bereichen signifikante Unterschiede zwischen den beiden Gruppen. So hatten die Teilnehmer der Anfänger_{08}-Gruppe vor dem Extreme Programming-Praktikum mehr Programmiererfahrung mit Java und testgetriebener Entwicklung als die Teilnehmer der Anfänger_{06}-Gruppe sammeln können, wie die p-Werte der entsprechenden Wilcoxon-Tests von 0,013 und 0,037 belegen. Zudem hatten die Teilnehmer der Anfänger_{08}-Gruppe länger studiert als die Teilnehmer der Anfänger_{06}-Gruppe ($p = 0{,}014$).

Im Durchschnitt hatten die zwölf Teilnehmer in der Anfänger_{06}-Gruppe vor dem Praktikum bereits 6,5 Semester Informatik studiert. Ihre allgemeine Programmiererfahrung belief sich auf 4,8 Jahre, davon 2,0 Jahre Programmiererfahrung mit Java. Drei der Teilnehmer hatten schon in einer Firma Java programmiert. Fünf der Teilnehmer in der Anfänger_{06}-Gruppe gaben an, vor dem Praktikum Erfahrung mit Paarprogrammierung gesammelt zu haben. Drei davon gaben an, diese Erfahrung in der Industrie gemacht zu

haben. Ein Teilnehmer aus dieser Gruppe gab weiterhin an, JUnit zuvor in der Industrie angewendet zu haben. Mit sonstigen Techniken zur Testautomatisierung und mit testgetriebener Entwicklung war noch keiner der Teilnehmer aus der Anfänger_{06}-Gruppe vor dem Praktikum in Kontakt gekommen.

Die zwölf Teilnehmer der Anfänger_{08}-Gruppe hatten durchschnittlich 7,7 Semester studiert. Bis auf einen Studenten der Elektrotechnik studierten alle Teilnehmer Informatik. Vor dem Extreme Programming-Praktikum hatten sie durchschnittlich 6,1 Jahre programmiert, davon 4,0 Jahre mit Java. Zwei der Teilnehmer in dieser Gruppe hatten auch industrielle Programmiererfahrung mit Java vorzuweisen. Jeweils drei Teilnehmer der Anfänger_{08}-Gruppe hatten zuvor Erfahrung mit Testautomatisierung und JUnit gesammelt, wiederum jeweils ein Teilnehmer hatte diese Erfahrung auch in einer Firma erworben. Mit Paarprogrammierung und testgetriebener Entwicklung hatten vier Teilnehmer dieser Gruppe schon vor dem Praktikum gearbeitet, wobei keiner von Ihnen industrielle Programmiererfahrung in diesen Techniken aufweisen konnte.

Die professionellen Softwareentwickler in der Experten-Gruppe haben alle ein Diplom der Informatik, das sie nach einer durchschnittlichen Studiendauer von 13,3[1] Semestern erreicht hatten. Bei der industriellen Programmiererfahrung zeigte sich in allen erfassten Teilbereichen die größere Erfahrung der Experten-Gruppe gegenüber den beiden Anfänger-Gruppen, wie die p-Werte der entsprechenden paarweisen Wilcoxon-Tests von durchgehend kleiner 0,001 zeigen. Bis zum Zeitpunkt des Quasi-Experiments konnten die Teilnehmer in der Experten-Gruppe im Mittel 7,8 Jahre industrielle Programmiererfahrung sammeln, dies schließt 7,2 Jahre industrielle Programmiererfahrung mit Java ein. In ihrer bisherigen Berufslaufbahn hatten sie durchschnittlich 5,5 Jahre mit Paarprogrammierung, 5,6 Jahre mit JUnit sowie 3,0 Jahre mit testgetriebener Entwicklung gearbeitet. Mit weiteren Techniken zur Testautomatisierung hatten sie im Durchschnitt 5,3 Jahre Erfahrung.

4.5 Aufgabe Fahrstuhlsteuerung

Bei der Aufgabe Fahrstuhlsteuerung muss die Steuerungssoftware für ein Fahrstuhlsystem vervollständigt werden. Die Steuerungssoftware verarbeitet Transportanfragen von den Nummerntasten aus dem inneren der Kabine und von den Kontrollfeldern mit den Hoch- und Runter-Tasten auf den einzelnen Stockwerken eines Gebäudes. Den Kern der Steuerungssoftware bildet ein endlicher Automat mit den vier Zuständen *Aufzug fährt hoch*, *Aufzug fährt runter*, *Aufzug wartet* und *Tür offen*. Die Aufgabenbeschreibung ent-

[1] Diese Zahl beruht auf den Daten von zehn Teilnehmern, da zwei Teilnehmer diese Frage nicht beantwortet haben.

Metrik	*Codebereich*		
	Anwendung	*Test*	*Gesamt*
Anzahl der Klassen & Schnittstellen	10	7	17
Anzahl der Methoden	65	87	152
Länge [SLOC]	596	367	963
Anweisungsabdeckung [%]	90,8	–	–
Anzahl der Testfälle	–	82	–

Tabelle 4.2: Kennzahlen der Aufgabe Fahrstuhlsteuerung

hält ein Zustandsübergangsdiagramm in dem die Bedingungen für die Zustandsübergänge und die beim jeweiligen Zustandsübergang auszuführenden Aktionen erläutert sind. Damit der Programmieraufwand überschaubar bleibt, wurde an die Teilnehmer ein Programmskelett ausgegeben, in dem alle Zustände bis auf den *Tür offen*-Zustand bereits implementiert sind. Wie in Tabelle 4.2 zusammengefasst, umfasst der Anwendungscode dieses Programmskeletts 10 Klassen und Schnittstellen mit 65 Methoden und 596 Zeilen nach dem in Abschnitt 2.4.1.2 beschriebenen Verfahren bereinigten Code. Der Testcode des Programmskeletts besteht aus 7 Klassen und Schnittstellen mit 87 Methoden und 367 Zeilen bereinigtem Code. Die im Testcode vorhandenen 82 Testfälle decken 90,8 Prozent der Anweisungen des Anwendungscodes ab.

Einige der Testfälle des Programmskeletts verwenden ein Mock-Objekt [MFC01], um die Logik, die den Fahrstuhl steuert, von der Logik zu trennen, die eingehende Transportanfragen verwaltet. Das Mock-Objekt bietet jedoch nicht genügend Funktionalität an, um die gesamte Fahrstuhlsteuerung testgetrieben entwickeln zu können. Die Paare mussten folglich am Mock-Objekt Änderungen vornehmen, um die Fahrstuhlsteuerung komplett testen zu können.

4.6 Ergebnisse

Im Folgenden werden wir die Ergebnisse der Auswertung des Datensatzes präsentieren. Im Rahmen der Auswertung wurden zur Überprüfung in Abschnitt 4.1 aufgeführten Hypothesen jeweils drei paarweise Vergleiche mit einem zweiseitigen Wilcoxon-Test durchgeführt. Das Signifikanzniveau aller drei Vergleiche sollte 0,05 nicht überschreiten, daher wurden die p-Werte nach Bonferroni-Holm korrigiert (vgl. Abschnitt 2.1.2.2).

Die p-Werte zu den Hypothesentests werden wir bei der Besprechung der Ergebnisse immer nennen, Lage- und Streuungsmaße zu den erhobenen Metriken werden wir aber

nur besprechen, wenn sie uns erwähnenswert erscheinen. Eine vollständige Übersicht über Lage- und Streuungsmaße findet sich jedoch, zusammen mit einer Übersicht über alle *p*-Werte, in Anhang A.

4.6.1 TGE-Prozess

Unsere anfängliche Vermutung war, dass sich die Unterschiede bei der Programmiererfahrung am stärksten auf den relativ schwer zu erlernenden testgetriebenen Entwicklungsprozess auswirken, wie das schon in [MH07] der Fall war. Deswegen wollen wir die Besprechung der Ergebnisse mit den Metriken des testgetriebenen Entwicklungsprozesses beginnen.

4.6.1.1 TGE-Prozesstreue

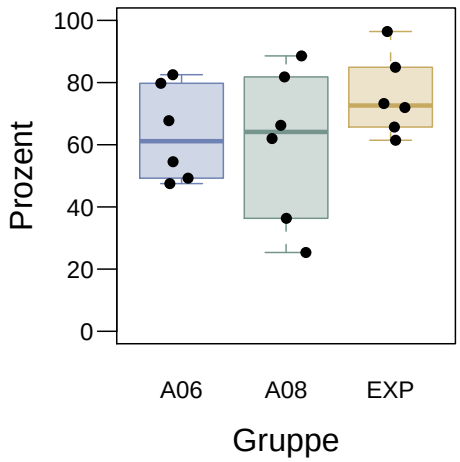


Abbildung 4.2: TGE-Prozesstreue.

Abbildung 4.2 zeigt die TGE-Prozesstreue der Teilnehmer in den einzelnen Gruppen. Auf den ersten Blick scheint die Prozesstreue innerhalb der Experten-Gruppe höher zu sein als in den beiden Anfänger-Gruppen. Jedoch ist die Überlappung der Boxen relativ groß, so dass es angesichts der geringen Güte nicht zu erwarten ist, dass wir einen statistisch signifikanten Unterschied nachweisen können. Dementsprechend überrascht es nicht, dass die Wilcoxon-Tests keine signifikanten Ergebnisse liefern. Der *p*-Wert für den Vergleich von Experten- und Anfänger$_{06}$-Gruppe ist 0,721, für den Vergleich der Experten- und Anfänger$_{08}$-Gruppe ist er 0,788 und für beide Anfänger-Gruppen beträgt

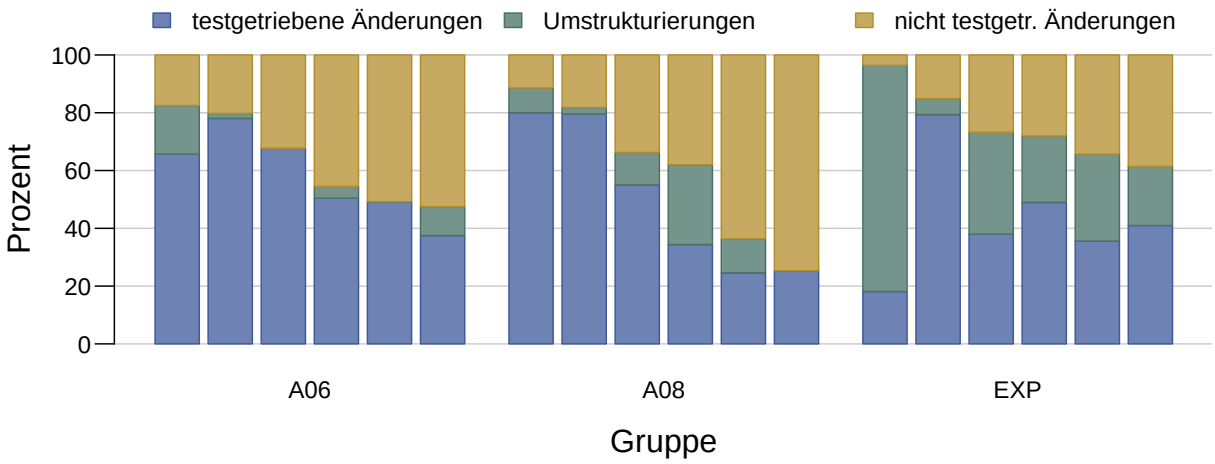


Abbildung 4.3: Anteile der testgetriebenen Änderungen, Umstrukturierungen und nicht testgetriebenen Änderungen an allen Änderungen.

der *p*-Wert 0,931. Somit können wir die Nullhypothese H_0^{TGE} nicht ablehnen. Der Nachweis eines Unterschieds zwischen den Gruppen bezüglich der TGE-Prozesstreue ist uns nicht gelungen.

4.6.1.2 Anzahl der automatischen Umstrukturierungen

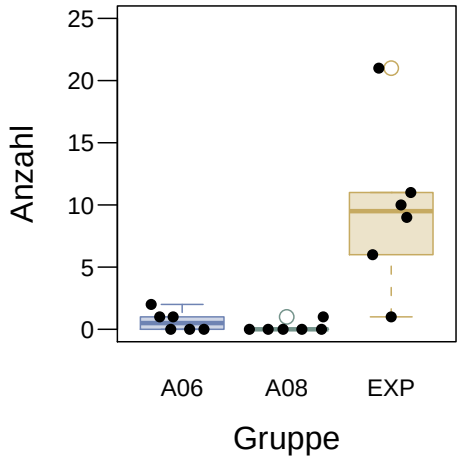


Abbildung 4.4: Anzahl der automatischen Umstrukturierungen.

Während der detaillierten Auswertung der TGE-Prozesstreue fiel uns auf, dass die Experten-Paare einen höheren Anteil an Umstrukturierungen aufwiesen als die Anfänger-Paare. Dies lässt sich bei Betrachtung von Abbildung 4.3 nachvollziehen. Wir haben dies

zum Anlass genommen, uns die TestPulse-Protokolle genauer anzusehen. Hier zeigten sich deutliche Unterschiede bei der Häufigkeit der Verwendung der von Eclipse angebotenen automatischen Umstrukturierungen.

Wie man in Abbildung 4.4 sehen kann, wurden in den beiden Anfänger-Gruppen die von Eclipse angebotenen Umstrukturierungen kaum genutzt. In der Anfänger$_{06}$-Gruppe hat nur ein Paar zwei automatische Umstrukturierungen durchgeführt. Zwei weitere Paare habe jeweils genau eine automatische Umstrukturierung durchgeführt. In der Anfänger$_{08}$-Gruppe gab es sogar nur ein Paar mit einer automatischen Umstrukturierung. Der Rest der Anfänger-Paare hat die automatischen Umstrukturierungen gar nicht verwendet. Eine andere Arbeitsweise konnten wir bei den Experten-Paaren beobachten: Hier gibt es lediglich ein Paar mit nur einer automatischen Umstrukturierung. Der Rest hat sie hingegen deutlich häufiger genutzt, wie der Median von 10,8 verdeutlicht. Um unsere Annahme zu überprüfen, haben wir die Nullhypothese, dass sich die Gruppen bezüglich der automatischen Umstrukturierungen gleichen, getestet. Die Ergebnisse der einzelnen Wilcoxon-Tests bestätigen das Bild. Der Unterschied zwischen der Anfänger$_{06}$- und der Experten-Gruppe ist mit einem p-Wert von 0,024 signifikant; ebenso der Unterschied zwischen der Anfänger$_{08}$- und der Experten-Gruppe mit einem p-Wert von 0,014. Zwischen den beiden Anfänger-Gruppen ist der Unterschied dagegen nicht signifikant ($p = 0{,}248$). Daher lehnen wir die Nullhypothese H_0^{AU} ab und nehmen an, dass zwischen den Anfänger-Gruppen und der Experten-Gruppe ein Unterschied bezüglich der automatischen Umstrukturierungen besteht.

4.6.1.3 Testläufe

Zunächst betrachten wir den mittleren Abstand der Testläufe in den Gruppen, der in Abbildung 4.5 dargestellt ist. Beim mittleren Abstand der Testläufe lässt sich nur schwer ein Unterschied zwischen den Gruppen ausmachen. Der Median liegt bei allen vier Gruppen zwischen drei bis vier Minuten und die Boxen überlappen sich zu großen Teilen. Folglich zeigt sich beim Test der Nullhypothese H_0^{MAT} weder zwischen Anfänger$_{06}$- und Experten-Gruppe noch zwischen Anfänger$_{08}$- und Experten-Gruppe ein signifikanter Unterschied ($p = 0{,}721$ und $p = 1$). Ebenso lässt sich kein signifikanter Unterschied zwischen den beiden Anfänger-Gruppen zeigen ($p = 1$).

Befassen wir uns als nächstes mit dem in Abbildung 4.6 gezeigten Variationskoeffizienten des Abstands der Testläufe. Dabei sticht ins Auge, dass der Variationskoeffizient des Abstands der Testläufe immer deutlich über 100 Prozent liegt, was bedeutet, dass die Standardabweichung des Abstands der Testläufe deren Mittelwert stets überschreitet. Was die Unterschiede zwischen den Gruppen angeht, sieht man, dass die Boxen der beiden Anfänger-Gruppen nah beieinander liegen, während sich die der Experten-Gruppe

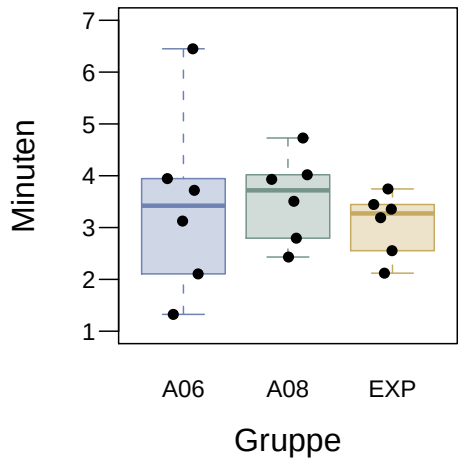


Abbildung 4.5: Mittlerer Abstand der Testläufe.

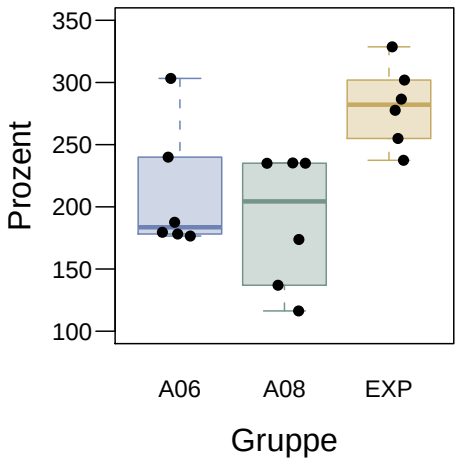


Abbildung 4.6: Variationskoeffizient des Abstands der Testläufe.

deutlich höher befindet. Auch der Median der Experten-Gruppe ist mit 282,1 Prozent höher als der der Anfänger$_{06}$-Gruppe mit 183,6 und der der Anfänger$_{08}$-Gruppe mit 204,4 Prozent. In der Tat zeigt sich, dass wir die Nullhypothese H_0^{VAT} zu Gunsten der Alternativhypothese H_A^{VAT} ablehnen können, da der p-Wert für den Vergleich von Anfänger$_{08}$- und der Experten-Gruppe von 0,006 auf einen signifikanten Unterschied zwischen diesen Gruppen hinweist. Für den Vergleich von Anfänger$_{06}$- und Experten-Gruppe liegt der p-Wert allerdings mit 0,13 knapp über dem Signifikanzniveau von 5 Prozent. Der Unterschied zwischen den beiden Anfänger-Gruppen ist mit einem p-Wert von 0,394 ebenfalls nicht signifikant.

Der Vollständigkeit halber sei an dieser Stelle erwähnt, dass sich die beiden Anfänger-Gruppen bezüglich der Anzahl der Testfälle ebenfalls signifikant von der Experten-Gruppe unterscheiden. Wir hatten diese Metrik jedoch zusammen mit der Standardabweichung der Testfälle wegen ihrer eingeschränkten Aussagekraft über den TGE-Prozess in Abschnitt 2.4.3.3 bewusst ausgeklammert. Wen diese Ergebnisse trotzdem interessieren, der findet im Anhang A in den Tabellen A.4 und A.3 die vollständigen Ergebnisse und in Abbildung A.1 den dazugehörigen Boxplot.

4.6.2 Produktivität

Bei der Produktivität liegt die Vermutung nahe, dass sich die größere Programmiererfahrung der Expertenpaare zu ihren Gunsten bemerkbar macht. Die folgenden Abschnitte werden zeigen, ob dies tatsächlich der Fall war.

4.6.2.1 Bearbeitungszeit

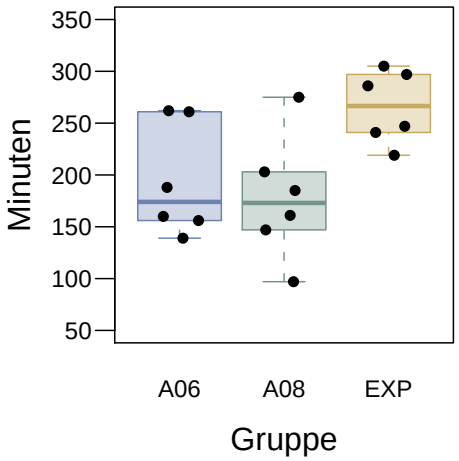


Abbildung 4.7: Bearbeitungszeit.

Als erste der Produktivitätsmetriken betrachten wir die Bearbeitungszeit. Abbildung 4.7 zeigt die Verteilung der Bearbeitungszeit in den einzelnen Gruppen. Betrachtet man die Lage der Boxen und der Mediane, scheinen die Paare in der Experten-Gruppe für die Entwicklung mehr Zeit gebraucht zu haben, als die Paare in den anderen zwei Gruppen. Tatsächlich lässt sich die Nullhypothese H_0^{BZ} ablehnen, da nicht alle Gruppen einander gleichen. Allerdings ist nur der Unterschied zwischen der Experten- und der Anfänger$_{08}$-Gruppe signifikant ($p = 0{,}045$). Bei der Experten- und der Anfänger$_{06}$-Gruppe ist der p-Wert zwar mit 0,13 niedrig, signifikant ist der Unterschied damit aber nicht. Zwischen den Anfänger-Gruppen besteht ebenfalls kein signifikanter Unterschied ($p = 0{,}937$).

4.6.2.2 Änderungen am Code

Bei den Änderungen am Code betrachten wir zunächst die Netto-Änderungen am gesamten Code. Diese sind in Abbildung 4.8 abgebildet. Auf den ersten Blick sieht es so aus, als hätten die Paare in der Experten-Gruppe mehr Änderungen am Code durchgeführt als die Paare in den Anfänger-Gruppen. Die Nullhypothese $H_0^{NÄ}$ kann auch zugunsten der Alternativhypothese $H_A^{NÄ}$ abgelehnt werden, jedoch zeigt sich bei genauerer Betrachtung, dass nur der Unterschied zwischen der Anfänger$_{08}$- und der Experten-Gruppe signifikant ist ($p = 0{,}045$). Wie auch schon bei der Bearbeitungszeit ist der p-Wert mit 0,155 für den Vergleich von Anfänger$_{06}$- und Experten-Gruppe recht niedrig, aber eben nicht niedrig genug um auf einen signifikanten Unterschied zwischen diesen Gruppen

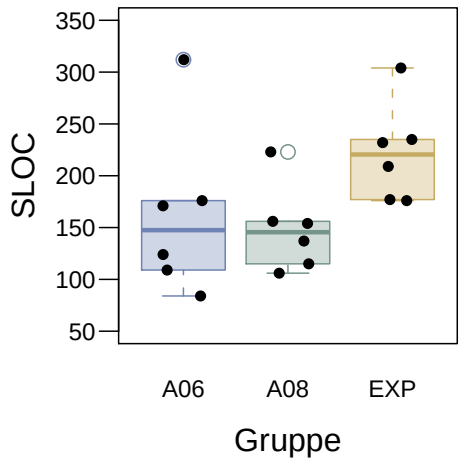


Abbildung 4.8: Netto-Änderungen.

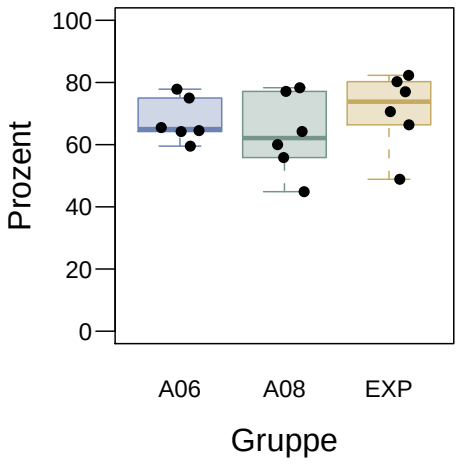


Abbildung 4.9: Anteil der Netto-Änderungen im Testcode.

schließen zu können. Ebenso besteht zwischen den Anfänger-Gruppen kein signifikanter Unterschied, wie der p-Wert von 0,937 zeigt.

Nehmen wir dieses Ergebnis mit den bereits präsentierten Erkenntnissen zu den automatischen Umstrukturierungen und der Bearbeitungszeit zusammen, lässt sich als Zwischenergebnis festhalten, dass die Experten anscheinend bereit waren, mehr Zeit in die Umstrukturierung des Codes zu investieren, als dies in den Anfänger-Gruppen der Fall war. Da Umstrukturierungen immer den Code verändern, ist es auch nicht weiter verwunderlich, dass sich auch bei den Netto-Änderungen ein Unterschied zwischen den Gruppen zeigt, auch wenn dieser nur zwischen der Anfänger_{08}- und der Experten-Gruppe signifikant ist.

Betrachtet man in Abbildung 4.9 den Anteil der Netto-Änderungen, die im Testcode stattgefunden haben, sieht man, dass die meisten Paare mehr am Testcode verändert haben als am Anwendungscode. Nur bei zwei Paaren liegt der Anteil der Netto-Änderungen im Testcode knapp unter 50 Prozent, beim Rest der Paare liegt der Anteil deutlich über dieser Marke. Ein signifikanter Unterschied zwischen den Gruppen lässt sich jedoch nicht nachweisen. Die drei p-Werte aller paarweisen Vergleiche betragen 0,929. Somit kann die Nullhypothese H_0^{TC} nicht verworfen werden.

Als letzte Metrik in diesem Abschnitt wollen wir die Anzahl der geänderten Dateien betrachten. Wie man in Abbildung 4.10 sehen kann, streut die Anzahl der geänderten Dateien bei der Experten-Gruppe mit vier bis sechs geänderten Dateien über einen geringeren Bereich als bei den Anfänger-Gruppen. So liegt die Anzahl der geänderten Dateien bei der Anfänger_{06}-Gruppe zwischen zwei und sieben beziehungsweise bei der Anfänger_{08}-Gruppe zwischen drei und sieben, wobei die Mehrheit in diesen beiden Gruppen nur zwischen zwei und drei Dateien geändert hat. Jedoch zeigt sich weder

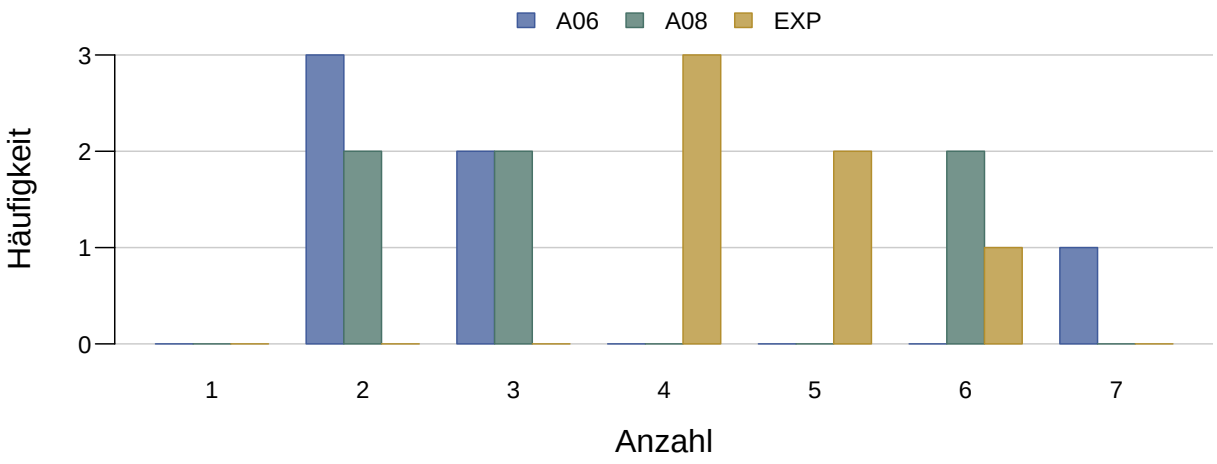


Abbildung 4.10: Anzahl der geänderten Dateien.

zwischen der Anfänger$_{06}$- und der Experten-Gruppe noch zwischen der Anfänger$_{08}$- und der Experten-Gruppe ein signifikanter Unterschied ($p = 0{,}183$ und $p = 0{,}577$). Erwartungsgemäß ist auch der Unterschied zwischen den beiden Anfänger-Gruppen nicht signifikant ($p = 0{,}672$). Daher können wir die Nullhypothese H_0^{GD} nicht ablehnen und müssen annehmen, dass die beobachteten Unterschiede bei der Anzahl der geänderten Dateien zufällig sind.

4.6.3 Qualität

Nachdem sich beim TGE-Prozess bereits zeigen ließ, dass die Experten-Paare mehr automatische Umstrukturierungen durchgeführt haben, ist es interessant zu sehen, ob diese sich in einer Verbesserung der Qualität, insbesondere bei den Änderungen an den Testfällen, niedergeschlagen haben.

4.6.3.1 Änderungen an den Testfällen

Werfen wir daher zunächst einen Blick auf die Änderung der Anzahl der Testfälle in Abbildung 4.11. Man sieht, dass sich bei allen Paaren die Anzahl der Testfälle erhöht hat. Dabei wurden von den Experten-Paaren zu den ursprünglich 82 Testfällen tendenziell mehr Testfälle hinzugefügt als von den Paaren in den Anfänger-Gruppen. Der Median der Änderung der Anzahl der Testfälle liegt in der Experten-Gruppe mit 11 Testfällen etwas höher als in der Anfänger$_{06}$-Gruppe mit 7 Testfällen und der Anfänger$_{08}$-Gruppe mit 7,5 Testfällen. Die Nullhypothese $H_0^{\Delta TF}$ lässt sich aber dennoch nicht ablehnen: Die paarweisen Wilcoxon-Tests liefern mit einem p-Wert von jeweils 0,373 für die Vergleiche der beiden Anfänger-Gruppen mit der Experten-Gruppe sowie einem p-Wert von

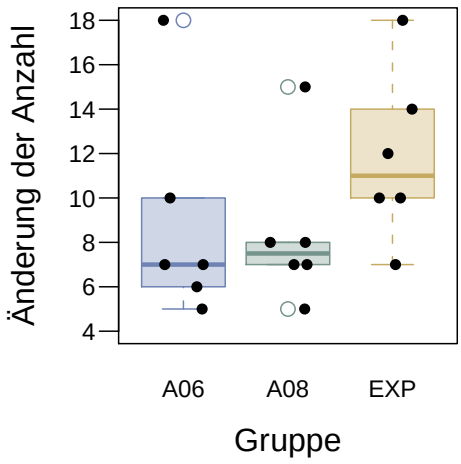


Abbildung 4.11: Änderung der Anzahl der Testfälle.

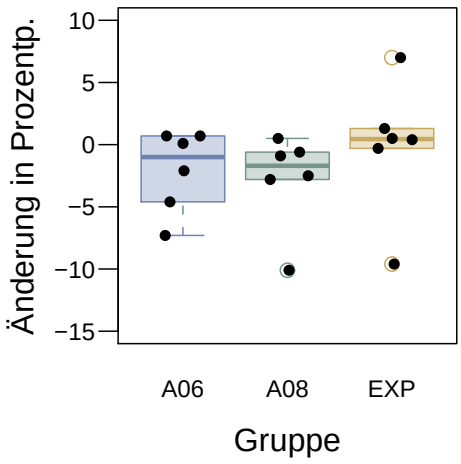


Abbildung 4.12: Änderung der Anweisungsabdeckung des Anwendungscodes.

0,87 für den Vergleich der beiden Anfänger-Gruppen untereinander keine signifikanten Ergebnisse.

Bei der in Abbildung 4.12 dargestellten Änderung der Anweisungsabdeckung zeigt sich ein ähnlicher Trend wie schon bei der Änderung der Anzahl der Testfälle. Wieder liegt die Experten-Gruppe vorne, wenn auch die Gruppen insgesamt noch enger beieinander liegen. Auffällig ist zudem, dass sich die hohe Anweisungsabdeckung des ausgeteilten Programmskeletts von 90,8 Prozent bei vielen Paaren verschlechtert hat. So liegt der Median der Änderung der Anweisungsabdeckung in beiden Anfänger-Gruppen unter dem Nullpunkt. In der Anfänger_{06}-Gruppe beträgt er -1 und in der Anfänger_{08}-Gruppe $-1{,}7$. Lediglich in der Experten-Gruppe liegt der Median mit 0,5 leicht über dem Nullpunkt.

Diese Unterschiede sind jedoch nicht groß genug, um als signifikant gelten zu können. So ist der p-Wert des Vergleichs der Experten- mit der Anfänger_{06}-Gruppe genau wie der p-Wert des Vergleichs der Anfänger-Gruppen 0,941. Für den Vergleich der Experten- mit der Anfänger_{08}-Gruppe ergibt sich ein p-Wert von $0{,}326$. Damit können wir die Nullhypothese $H_0^{\Delta AÜ}$ ebenfalls nicht ablehnen.

4.6.3.2 Anzahl der Akzeptanztestläufe

Betrachten wir als letzte Metrik die Anzahl der Akzeptanztestläufe, die benötigt wurden, um den Akzeptanztest zu bestehen. Wie schon in Abschnitt 4.3 erwähnt, wurde allen Teilnehmern mitgeteilt, dass der Akzeptanztest möglichst beim ersten Versuch zu bestehen ist. Wie man in Abbildung 4.13 sehen kann, haben 13 von 18 Paaren dieses

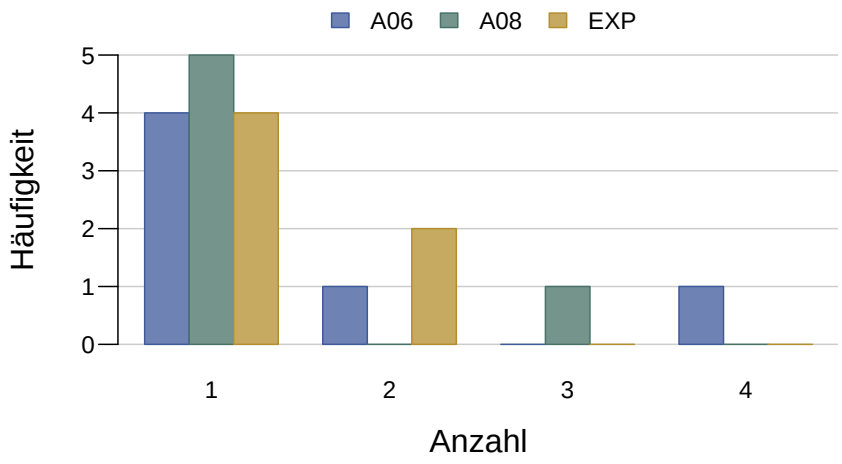


Abbildung 4.13: Die Anzahl der Akzeptanztestläufe.

Ziel auch erreicht. Nur insgesamt fünf Paare haben den Akzeptanztest nicht beim ersten Versuch bestanden.

Insgesamt ist die Verteilung der Anzahl der Akzeptanztestläufe in den Gruppen derart ähnlich, dass der p-Wert für alle paarweisen Gruppenvergleiche sein Maximum von 1 erreicht. Folglich müssen wir weiter davon ausgehen, dass die Nullhypothese H_0^{AT} gilt.

4.7 Validität der Studie

Bevor wir im nächsten Abschnitt die Ergebnisse der Studie noch einmal zusammenfassen und diskutieren möchten, befassen wir uns zunächst mit den Einschränkungen, die für diese Studie und die Übertragbarkeit ihrer Ergebnisse gelten. Christensen unterscheidet [Chr07, S. 215–257] vier grundlegende Arten der Validität einer Studie, an denen wir uns orientieren. Bevor wir die Validität dieser Studie diskutieren, listen wir die vier Arten der Validität kurz auf und geben jeweils eine kurze Definition, welche die Essenz dessen wiedergibt, was Christensen zur entsprechenden Art der Validität ausführt:

Validität des Konstrukts Die Validität des Konstrukts besagt, inwiefern die Schlussfolgerungen, die wir über ein komplexeres Konstrukt machen, aufgrund der Metriken, die wir zur Messung des Konstrukts herangezogen haben, Gültigkeit besitzen.

Statistische Validität Die statistische Validität besagt, inwieweit die Aussagen zur Kovarianz der unabhängigen und den abhängigen Variablen belastbar sind.

Interne Validität Die interne Validität beschäftigt sich mit der Sicherheit, mit der wir davon ausgehen können, dass die beobachteten Effekte bei den abhängigen Variablen durch Störvariablen beeinflusst wurden und somit nicht, wie es idealerweise

Klasse	*automatisch*		*manuell*			
			ohne Konflikt		*mit Konflikt*	
testgetriebene Änderungen	398	45,9 %	4	0,5 %	3	0,3 %
Umstrukturierungen	130	15,0 %	21	2,4 %	16	1,8 %
nicht testgetr. Änderungen		–	278	32,1 %	17	2,0 %
Summe	528	61,0 %	303	35,0 %	34	4,1 %

Tabelle 4.3: Anteile der automatisch, manuell ohne Konflikt und manuell mit Konflikt klassifizierten Änderungen.

sein sollte, alleine auf die Manipulation der unabhängigen Variable zurückzuführen sind.

Externe Validität Bei der externen Validität bewerten wir die Übertragbarkeit der in einer Studie gefundenen Ergebnisse auf andere Personen, Umgebungen und Aufgaben.

4.7.1 Validität des Konstrukts

Die Validität des Konstrukts kann durch Metriken mit wenig Aussagekraft zum ihnen zugeordneten Konstrukt gefährdet sein. Eine Metrik, die bezüglich ihrer Aussagekraft kritisch zu sehen ist, ist die Anzahl der Akzeptanztests als ein Maß für die Qualität der Programme. Obwohl die Teilnehmer in der Aufgabenstellung darauf hingewiesen wurden, dass der Akzeptanztest beim ersten Versuch bestanden werden soll, kann es sein, dass Teilnehmer insgeheim davon ausgingen, mehrere Versuche zum Bestehen zu haben und sich daher leichtfertig zum Akzeptanztest meldeten. Wir haben versucht, dem entgegenzuwirken, indem wir die Paare vor dem Akzeptanztest gefragt haben, ob sie sich sicher seien, eine fehlerfreies Programm entwickelt zu haben. Paaren, die den ersten Versuch nicht bestanden, mussten wir zwangsläufig mindesten einen weiteren Versuch zugestehen. Von da an wussten die Teilnehmer, dass sie in Wirklichkeit mehrere Versuche zum Bestehen des Akzeptanztests hatten. Daher hat ab zwei Akzeptanztestläufen die benötigte Anzahl der Akzeptanztestläufe nur noch eine schwache Aussagekraft über die Qualität des Programms, da mangelnder Respekt vor dem Akzeptanztest stets als rivalisierende Erklärung für die hohe Anzahl der Versuche in Frage kommt.

Ein weiterer Faktor, der die Validität des Konstrukts gefährden kann, sind Messfehler. Hierfür sind die von den Teilnehmern ausgefüllten Zeiterfassungsbögen anfällig, welche die Grundlage für die Messung der Bearbeitungszeit bilden. Deswegen haben

wir, während die Teilnehmer die Aufgabe bearbeiteten, immer wieder einen Blick in die Zeitprotokolle geworfen und die Teilnehmer in ihren Pausen daran erinnert, diese Pausen korrekt zu erfassen. Später haben wir zusätzlich die Videoaufzeichnungen dazu genutzt, um die Zeitprotokolle auf ihre Korrektheit zu prüfen. Dabei konnten wir nur geringfügige Abweichungen feststellen, die sich zumeist durch die Verwendung ungenau gehender Armbanduhren oder Uhren in Mobiltelefonen durch die Teilnehmer erklären ließen.

Eine weitere Stelle, an der es zu Messfehlern gekommen sein kann, ist die manuelle Klassifizierung aller von TestPulse nicht automatisch klassifizierbaren Änderungen. Um die Fehlklassifizierung von Änderungen möglichst zu vermeiden, wurden die manuelle Klassifizierung für alle Datenpunkte zunächst durch den Autor dieser Arbeit und eine am Lehrstuhl beschäftigte wissenschaftliche Hilfskraft unabhängig voneinander ausgeführt. In einer gemeinsamen Sitzung wurden die manuellen Klassifizierungen zusammengeführt und dabei auftretende Konflikte, d. h. unterschiedliche Klassifizierungen der gleichen Änderung, beseitigt.

Wie in Tabelle 4.3 zu sehen ist, mussten insgesamt 39,1 Prozent aller Änderungen manuell klassifiziert werden. Allerdings traten nur bei 4,1 Prozent aller Änderungen Konflikte auf. Bezogen auf die manuell klassifizierten Änderungen heißt dies, dass bei jeder zehnten Änderung ein Konflikt auftrat. Diese Konflikte ließen sich jedoch in allen Fällen ohne lange Diskussionen auflösen.

4.7.2 Statistische Validität

Die Gründe unserer Entscheidung für den zweiseitigen Wilcoxon-Test und die Bonferroni-Holm-Korrektur der p-Werte haben wir bereits in den Abschnitten 2.1.2.1 und 2.1.2.2 dargelegt. Daher wollen wir diese Gründe an dieser Stelle nicht weiter diskutieren. Vielmehr wollen wir die Irrtumswahrscheinlichkeiten betrachten, die sich ergeben, wenn wir eine Nullhypothese ablehnen bzw. wenn uns dies nicht gelingt.

Die Wahrscheinlichkeit einen Fehler 1. Art zu begehen, ist durch das von uns gewählte Signifikanzniveau von 5 Prozent nach oben begrenzt. Durch die Bonferroni-Holm-Korrektur ist auch gewährleistet, dass die Gesamtfehlerrate der pro Nullhypothese notwendigen drei paarweisen Wilcoxon-Tests das Signifikanzniveau nicht überschreitet.

Um sagen zu können, wie groß die Wahrscheinlichkeit, einen Fehler 2. Art zu machen, schlimmstenfalls sein kann, müssen wir zunächst die Güte berechnen. Hierbei müssen wir beachten, dass bei der Bonferroni-Holm-Korrektur der p-Werte der kleinste p-Wert der drei Gruppenvergleiche mit der Anzahl der Gruppen, in unserem Fall also drei, multipliziert wird bevor er mit dem Signifikanzniveau verglichen wird. Dadurch hat der entsprechende Wilcoxon-Test ein effektives Signifikanzniveau von $\frac{5}{3\times100}$.

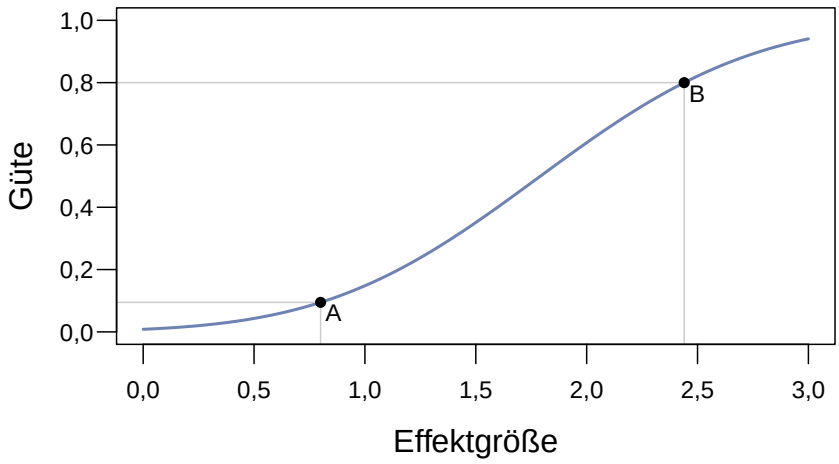


Abbildung 4.14: Zusammenhang zwischen Güte und Effektgröße für den Wilcoxon-Test bei einer Gruppengröße von 6 und einem Signifikanzniveau von $\frac{5}{3\times 100}$.

Weiter müssen wir bei der Berechnung die ARE des Wilcoxon-Tests von 0,864 gegenüber dem t-Test berücksichtigen. Damit betrachten wir den ungünstigsten Fall und haben somit eine Abschätzung der Güte nach unten. Wie Punkt A in Abbildung 4.14 zeigt, ergibt sich unter diesen Voraussetzungen bei einer Gruppengröße von 6 für große Effekte von 0,8 eine Güte von mindestens 9,5 Prozent. Dies bedeutet, dass selbst für große Effekte die Wahrscheinlichkeit für einen Fehler 2. Art bei bis zu 90,5 Prozent liegen kann. Eine Güte von 80 Prozent bzw. einer Wahrscheinlichkeit für einen Fehler 2. Art von 20 Prozent, wie es üblicherweise erwünscht ist (vgl. [Coh88, S. 531]), wird, wie an Punkt B zu sehen, erst für eine Effektgröße von 2,440 erreicht.

Zusammengefasst bedeutet dies, dass signifikante Unterschiede zwischen den Gruppen wahrscheinlich auch zwischen den Populationen existieren und vermutlich auch groß sind, da wir sie sonst nicht hätten nachweisen können. In allen Fällen, in denen wir keinen signifikanten Unterschied nachweisen konnten, ist die Irrtumswahrscheinlichkeit jedoch hoch, so dass auch hier Unterschiede zwischen den Populationen vorhanden sein könnten, die wir jedoch aufgrund der geringen Güte nicht aufdecken konnten.

4.7.3 Interne Validität

Die interne Gültigkeit unseres Quasi-Experiments könnte durch mehrere Störfaktoren eingeschränkt sein. Der erste dieser Störfaktoren betrifft den bereits in Abschnitt 4.3 angesprochenen Ausschluss von fünf Datenpunkten aus dem Datensatz. Man spricht hierbei von der sogenannten Sterblichkeit. Steht diese mit einer bestimmten Eigenschaft einer Gruppe in Zusammenhang, was bei nicht zufälliger Einteilung der Gruppen der Fall

Gruppe	*trifft voll zu*	*trifft eher zu*	*weiß nicht*	*trifft eher nicht zu*	*trifft überhaupt nicht zu*
A06	2	5	5	0	0
A08	4	5	1	2	0
EXP	5	2	2	1	0
Summe	11	12	8	3	0

Tabelle 4.4: Verteilung der Bewertungen der Aussage „Die Programmierung im Experiment hat mir gefallen.“.

Gruppe	*trifft voll zu*	*trifft eher zu*	*weiß nicht*	*trifft eher nicht zu*	*trifft überhaupt nicht zu*
A06	4	4	3	1	0
A08	5	6	1	0	0
EXP	8	1	1	0	0
Summe	17	11	5	1	0

Tabelle 4.5: Verteilung der Bewertungen der Aussage „Ich würde jederzeit wieder mit meinem Partner zusammenarbeiten.“.

sein kann, kann sie das Ergebnis der Studie systematisch verfälschen. Ganz ausschließen lässt sich das auch für unser Quasi-Experiment nicht, kann jedoch als unwahrscheinlich angesehen werden. Die Datenpunkte von drei Paaren mussten wir ausschließen, da diese Paare nicht in der Lage waren, in der ihnen zur Verfügung stehenden Zeit eine Lösung zu entwickeln, die den Akzeptanztest bestand. Da es in jeder Gruppe genau ein solches Paar gab, ist eine systematische Beeinflussung zu Gunsten einer der drei Gruppen zumindest unwahrscheinlich. Weitere zwei Datenpunkte waren wegen einer Fehlfunktion der Eclipse-Plugins von TestPulse nicht auswertbar. Dies betraf nur die Anfänger_{06}-Gruppe. Welche Datenpunkte aber beschädigt wurden, hat nichts mit einer bestimmten Eigenschaft der dazugehörigen Paare oder der Gruppe zu tun, kann also als quasi zufällig angesehen werden. Dadurch ist eine systematische Beeinflussung der Ergebnisse der Studie durch den Wegfall dieser Datenpunkte wiederum sehr unwahrscheinlich.

Ein weiterer Störfaktor für die interne Gültigkeit könnte aus einer unterschiedlichen Motivation der Anfänger gegenüber den Experten entstehen. Schließlich wurden die Anfänger im Gegensatz zu den Experten für ihre Teilnahme nicht bezahlt. Um Motivationsunterschiede zu ergründen, haben wir alle Teilnehmer im Nachtest-Fragebogen die Aussage „Die Programmierung im Experiment hat mir gefallen.“ auf einer Likert-Skala von *trifft voll zu* bis *triff überhaupt nicht zu* bewerten lassen. Die Ergebnisse hierzu sind in Tabelle 4.4 dargestellt. Wie man dort sieht, hat die Mehrheit der Teilnehmer Spaß an der Teilnahme gehabt. Als Vorgriff auf die externe Validität sei hier gesagt, dass damit eine schlechten Motivation der Teilnehmer aufgrund der Überschreitung der in der Vertraulichkeitserklärung genannten Arbeitszeit als unwahrscheinlich gelten kann. Allerdings sieht man in Tabelle 4.4 auch, dass es in der Experten-Gruppe etwas mehr Teilnehmer gab, die die Aussage mit *trifft voll zu* bewerteten. Daher kann ein Motivationsunterschied zwischen den Gruppen nicht ausgeschlossen werden.

Eine weitere Gefahr für die interne Validität stellt die Tatsache dar, dass sich die Partner in den Anfänger-Paaren oft erst wenige Wochen kannten, während sie sich in den Experten-Paaren meist mehrere Jahre kannten. Dadurch könnten die Partner in den Experten-Paaren besser miteinander harmoniert haben als dies in den Anfänger-Paaren der Fall war. Um dies zu prüfen, sollten die Teilnehmer im Nachtest-Fragebogen die Aussage „Ich würde jederzeit wieder mit meinem Partner zusammenarbeiten.“ bewerten. Wie in Tabelle 4.5 zu sehen ist, konnten sich die meisten Teilnehmer eine erneute Zusammenarbeit mit ihrem Partner jederzeit vorstellen. Immerhin 28 Teilnehmer bewerteten die Aussage mit *trifft voll zu* bzw. mit *trifft eher zu*. Allerdings stammen alleine acht der Teilnehmer, die *trifft voll zu* angaben, aus der Experten-Gruppe. Daher muss eine größere Harmonie in den Paaren der Experten-Gruppe als Gefahr für die interne Validität in Betracht gezogen werden.

Weiterhin ist die interne Gültigkeit dadurch eingeschränkt, dass die Experten nicht nur

Gruppe	*trifft voll zu*	*trifft eher zu*	*weiß nicht*	*trifft eher nicht zu*	*trifft überhaupt nicht zu*
A06	1	2	2	4	3
A08	0	2	0	4	6
EXP	0	1	3	1	5
Summe	1	5	5	9	14

Tabelle 4.6: Verteilung der Bewertungen der Aussage „Ich fühlte mich durch die Kameras gestört und beobachtet.“.

Gruppe	*trifft voll zu*	*trifft eher zu*	*weiß nicht*	*trifft eher nicht zu*	*trifft überhaupt nicht zu*
A06	1	1	3	4	3
A08	1	0	2	3	6
EXP	0	0	3	6	1
Summe	2	2	8	13	10

Tabelle 4.7: Verteilung der Bewertungen der Aussage „Ich war aufgeregt, weil ich an einem Experiment teilnahm und konnte deshalb nicht so arbeiten, wie ich es gewohnt bin.“.

mehr Erfahrung in der Paarprogrammierung und der testgetriebenen Entwicklung sondern auch mehr allgemeine Programmiererfahrung als die Anfänger hatten. Dies hätte sich nur lösen lassen, indem man als Anfänger anstelle von Studenten in agilen Entwicklungsmethoden unerfahrene Softwareentwickler genommen hätte und in einem Kurs in Paarprogrammierung und testgetriebener Entwicklung ausgebildet hätte. Diese ließen sich aber leider nicht finden. Daher müssen wir hinnehmen, dass zumindest ein Teil der beobachteten Effekte auch in Unterscheiden in der allgemeinen Programmiererfahrung begründet sein können.

4.7.4 Externe Validität

Zunächst ist die externe Validität unserer Studie dadurch eingeschränkt, dass die Teilnehmer nicht zufällig sondern nach ihrer Verfügbarkeit ausgewählt wurden. Die Ergeb-

nisse sind daher nur auf Populationen übertragbar, die in ihren Eigenschaften denen der Teilnehmer in den Gruppen ähneln. Weiter sind aufgrund der relativ kurzen Dauer des Quasi-Experiments Rückschlüsse auf Langzeiteffekte nur schwer möglich. Zudem schränken die in der Aufgabe verwendete Programmiersprache Java und das dort eingesetzt Testrahmenwerk JUnit die Übertragbarkeit der Ergebnisse auf objektorientierte Sprachen ein, für die ein ausgereiftes Testrahmenwerk zur Verfügung steht.

Eine weitere Bedrohung für die externe Validität des Quasi-Experiments geht von der mit Kameras ausgestatteten Arbeitsumgebung aus. Die Kameras könnten die Teilnehmer gestört haben, so dass die Teilnehmer möglicherweise nicht ihr übliches Arbeitsverhalten gezeigt haben. Um zu ermitteln, inwieweit dies der Fall war, haben wir die Teilnehmer die Aussage „Ich fühlte mich durch die Kameras gestört und beobachtet.“ auf der gleichen Likert-Skala wie auch schon im Abschnitt zuvor zu bewerten. Wie man in Tabelle 4.6 sehen kann, gab es zwar vereinzelt Teilnehmer, die sich durch die Kameras gestört fühlten, die meisten Teilnehmer nahmen die Kameras jedoch nicht als störend wahr. Insgesamt ist die Einschränkung der externen Validität durch die Kameras als gering einzuschätzen.

Neben den Kameras könnte auch Aufregung im Allgemeinen, bedingt durch die für die Teilnehmer ungewohnte Teilnahme am Quasi-Experiment das Arbeitsverhalten der Teilnehmer beeinflusst haben. Diesbezüglich haben wir die Teilnehmer gebeten, die Aussage „Ich war aufgeregt, weil ich an einem Experiment teilnahm und konnte deshalb nicht so arbeiten, wie ich es gewohnt bin.“ zu bewerten. Aus Tabelle 4.7 kann man erkennen, dass die Mehrheit der Teilnehmer diese Aussage als eher oder überhaupt nicht zutreffend empfanden. Daher ist die Gefahr für die externe Validität durch die mögliche Aufregung der Teilnehmer als gering zu erachten.

4.8 Zusammenfassung und Diskussion der Ergebnisse

In diesem Kapitel haben wir ein Quasi-Experiment vorgestellt, das sich mit den Unterschieden zwischen den testgetriebenen Entwicklungsprozessen von erfahrenen und unerfahrenen Programmiererpaaren auseinandergesetzt hat. Die erste Erkenntnis, die wir mit dieser Studie gewinnen konnten, ist, dass die Paare der Experten-Gruppe ihren Code häufiger umstrukturiert haben, als die Paare in den Anfänger-Gruppen. Wir konnten einen signifikanten Unterschied bei den automatischen Umstrukturierungen nachweisen und auch der Anteil der Umstrukturierungen im TGE-Prozess scheint bei den Expertenpaaren tendenziell größer zu sein.

Unklar ist, was die Expertenpaare zu den häufigeren Umstrukturierungen bewogen hat und ob diese Umstrukturierungen einen Nutzen hatten. Bei den Qualitätsmetriken ließen

sich zumindest keine signifikante Unterschiede nachweisen, allerdings ist anzunehmen, dass sich ein eventuell besser strukturierter Code erst langfristig positiv bemerkbar machen würde. Langzeiteffekte lassen sich jedoch bei der Bearbeitung einer kurzen Aufgabe nicht nachweisen.

Kurzfristig machten sich die häufigeren Umstrukturierungen eher negativ in Form einer längeren Bearbeitungszeiten in der Experten-Gruppe im Vergleich mit den beiden Anfänger-Gruppen bemerkbar. Hier konnten wir zwar nur einen signifikanten Unterschied zwischen der Experten-Gruppe und der Anfänger_{08}-Gruppe aufdecken, jedoch weist der Trend für den Vergleich von Experten- und Anfänger_{06}-Gruppe in die gleiche Richtung.

Weiterhin konnten wir zeigen, dass die Paare der Experten-Gruppe signifikant mehr Netto-Änderungen durchgeführt hatten als die Paare der Anfänger_{08}-Gruppe. Wiederum zeigte sich bei der Gegenüberstellung von Experten- und Anfänger_{06}-Gruppe eine ähnliche Tendenz aber kein signifikanter Unterschied. Die Ursache ist vermutlich auch hier in den häufigeren Umstrukturierungen in der Experten-Gruppe zu sehen.

Des Weiteren gab es bei den Metriken zum TGE-Prozess einen signifikanten Unterschied beim Variationskoeffizienten des Abstands der Testläufe. Dieser war in der Experten-Gruppe signifikant größer als in der Anfänger_{08}-Gruppe. Auch im Vergleich zur Anfänger_{06}-Gruppe war der Variationskoeffizient des Abstands der Testläufe in der Experten-Gruppe größer, allerdings war der Unterschied zwischen diesen beiden Gruppen nicht signifikant. Dieses Ergebnis ist überraschend, da wir erwartet hatten, dass der TGE-Prozess der Paare in den Anfänger-Gruppen stärkeren Schwankungen unterliegt, als der TGE-Prozess der Paare in der Experten-Gruppe. Es ist jedoch möglich, dass diese größeren Schwankungen in der Experten-Gruppe ebenfalls mit den häufigeren Umstrukturierungen in Zusammenhang stehen.

Nehmen wir diese Ergebnisse und die aus [MH07] zusammen, zeigt sich, dass sich die testgetriebenen Entwicklungsprozesse von erfahrenen und unerfahrenen Entwicklern sowohl in Kombination mit der Einzel- als auch mit der Paarprogrammierung deutlich unterscheiden. Daher sollte von der Übertragung von den mit Studenten oder vergleichbar unerfahrenen Entwicklern gewonnen Ergebnissen abgesehen werden. Ist diese Übertragbarkeit dennoch erwünscht, so sollte sie für den konkreten Einzelfall empirisch nachgewiesen werden.

Obgleich ein direkter Vergleich der beiden Studien auf den ersten Blick vielversprechend erscheint, gibt es, von der Einzel- und Paarprogrammierung abgesehen, einige entscheidende Punkte, die diesen Vergleich ungültig machen: So berechnet TestPulse einige Metriken wie z. B. die TGE-Prozesstreue inzwischen anders als die Vorgängerversion dieses Werkzeugs. Weiterhin verwenden die Studien zwar prinzipiell die gleiche Aufgabe, allerdings wurde der Code inzwischen von Java 1.4 auf Java 5 portiert und das

alte Testrahmenwerk JUnit 3 durch das mit Annotationen arbeitende JUnit 4 ersetzt, wobei auch die Testfälle aufgeräumt wurden. Nicht zuletzt wurden bei der Datensammlung zu dieser Studie Kameras eingesetzt und bei [MH07] nicht.

Aus diesem Grund wäre für die Zukunft ein multifaktorielles Experiment interessant, das beide Studien vereinigt und somit in der Lage ist aufzudecken, wie die Paarprogrammierung die testgetriebenen Entwicklungsprozesse von erfahren und unerfahren Entwicklern beeinflusst.

Kapitel 5

Effekt der Persönlichkeit auf die testgetriebene Paarprogrammierung

Dieses Kapitel schildert ein Experiment, das sich mit der Frage beschäftigt, ob Programmierpaare, in denen die Programmiere unterschiedliche Persönlichkeitsmerkmale haben, anders arbeiten als Paare, in denen die Programmierer gleiche Persönlichkeitsmerkmale haben. Wie Sfetsos und Kollegen [SSAD09] nutzen wir den Keirsey-Temperamenttest (Keirsey-Temperament-Sorter), um die Persönlichkeitsmerkmale der Teilnehmer zu bestimmen. Daher soll der folgende Abschnitt zunächst die Grundlagen zu diesem Test erläutern.

5.1 Der Keirsey-Temperamenttest

Nach dem Typenmodell des amerikanischen Psychologen David Keirsey gibt es vier Dimensionen, welche die menschliche Persönlichkeit prägen. Sein Typenmodell basiert im Wesentlichen auf dem von Myers-Briggs [MBM95] entwickelten Typenmodell, welches seine Ursprünge in der Jung'schen Typenlehre hat.

5.1.1 Die vier Dimensionen

Jede Dimension in Keirseys Typenmodell besitzt zwei gegensätzliche Pole, die jeweils durch einen Buchstaben gekennzeichnet werden. Die folgenden Abschnitte zeigen, von welchen (englischen) Bezeichnungen für die Pole die Buchstaben abgeleitet wurden und erläutern die Pole jeder Dimension kurz:[1]

E vs. I Extraversion (*E*xtraversion) versus Introversion (*I*ntroversion)

Extravertierte Menschen beziehen ihre Energie aus der Geselligkeit und der In-

[1] Eine ausführliche Erklärung der Pole liefert Keirsey [KB90] auf den Seiten 18–35.

teraktion mit anderen Menschen. Introvertierte Menschen dagegen brauchen und schätzen die Privatsphäre und meiden eher den Kontakt zu anderen Menschen.

S vs. N Empfindung (*S*ensing) versus Intuition (I*n*tuition)
Zur Empfindung neigende Menschen haben einen ausgeprägten Sinn für Tatsachen und die Realität und nehmen diese in vielen Details war. Für Sie sind Erfahrungen aus der Vergangenheit von Bedeutung. Menschen, die zur Intuition neigen, sind dagegen eher verträumt geben sich ihren Ahnungen hin. Sie interessiert und fasziniert das Mögliche und die Zukunft.

T vs. F Denken (*T*hinking) versus Fühlen (*F*eeling)
Menschen die zum Denken tendieren, treffen Entscheidungen aufgrund von objektiven Kriterien. Ihnen sind Richtlinien und Normen von Bedeutung. Menschen mit einer Tendenz zum Fühlen sind bei Entscheidungen von subjektiven Kriterien und persönlichen Präferenzen gesteuert. Solchen Menschen sind Anteilnahme und individuelle Wertvorstellungen wichtig.

P vs. J Wahrnehmung (*P*ercieving) versus Urteilen (*J*udging)
Zur Wahrnehmung neigende Menschen, verspüren gegenüber Entscheidungen ein gewisses Unbehagen und schätzen nicht abgeschlossene Angelegenheiten. Urteilende Menschen hingegen schätzen abgeschlossene Angelegenheiten, sind eher erleichtert, wenn eine Entscheidung getroffen wurde.

Um die Ausprägung der vier Dimensionen bei Menschen messen zu können, entwickelte Keirsey den Keisey-Temperamenttest. Ziel des Tests ist die Bestimmung des so genannten Temperaments eines Menschen und damit der wichtigsten Züge seiner Persönlichkeit.

Ergebnis des aus 70 Fragen mit jeweils zwei gegensätzlichen Antwortmöglichkeiten bestehenden Keirsey-Temperamenttests[2] ist eine vierstellige Buchstabenkombination, die eines der insgesamt 16 Temperamente bezeichnet. Jeder Buchstabe kennzeichnet dabei einen Pol der vier Dimensionen. Es kann auch vorkommen, dass sich aus dem Keirsey-Temperamenttest heraus in einer oder mehreren Dimensionen keine eindeutige Tendenz zu einem der beiden Pole ergibt. In diesem Fall wird der Buchstabe *X* gebraucht, um einen Mischtyp zu kennzeichnen.

5.1.2 Die vier Basistemperamente

Schon Myers-Briggs beobachtete bei ihrer Arbeit, dass einige Temperamente einander ähnlicher sind als andere. Keirsey griff diese Beobachtung auf und fasste jeweils vier

[2] Siehe [KB90, S. 6 ff.] für die deutsche Fassung und [Kei98] für das englische Original.

ähnliche Temperamente zu einem Basistemperament zusammen. Die Basistemperamente heißen:

NF Idealist (Idealist)

NT Rationalist (Rational)

SJ Beschützer (Guardian)

SP Kunsthandwerker (Artisan)

Die beiden Buchstaben eines Basistemperaments geben an, welche zwei Pole aus den Polen der vier Dimensionen ein Basistemperament prägen. Während die Temperamente innerhalb der Basistemperamente einander ähneln, sind die Gegensätze zwischen den Temperamenten unterschiedlicher Basistemperamente groß. Die größten Unterschiede treten dabei zwischen Basistemperamenten NT und SP sowie den Basistemperamenten NF und SJ auf.

5.2 Verfeinerung der Forschungshypothese FH_2

Nachdem die Grundlagen zum Keirsey-Temperamenttest bekannt sind, können wir an dieser Stelle dazu übergehen aus der Forschungshypothese testbare Hypothesen abzuleiten. Die bereits in der Einleitung vorgestellte Forschungshypothese zu dieser Studie lautet wie folgt:

FH_2 Paare, in denen die Programmierer unterschiedliche Persönlichkeitsmerkmale haben, unterscheiden sich von Paaren, in denen die Programmierer gleiche Persönlichkeitsmerkmale haben, bezüglich ihrer Produktivität, der Qualität der abgelieferten Programme und ihres testgetriebenen Entwicklungsprozesses.

Um aus der Forschungshypothese die Alternativhypothesen abzuleiten, übertragen wir sie auf die in Abschnitt 2.4 zur Produktivität, Qualität und dem testgetriebenen Entwicklungsprozess vorgestellten Metriken. Um diesen Vorgang leichter nachvollziehbar zu machen, haben wir die Metriken in der ersten Spalte von Tabelle 5.1 nochmals aufgelistet. Wie auch schon im vorhergehenden Kapitel handelt es sich um elf Metriken, weswegen wir nicht jede Alternativhypothese einzeln aufführen wollen, sondern zunächst die folgende Schablone für die Alternativhypothesen formulieren:

$H_A^{[Abk.\ Metrik]}$ Paare, in denen die Programmierer unterschiedliche Persönlichkeitsmerkmale haben, unterscheiden sich von Paaren, in denen die Programmierer gleiche Persönlichkeitsmerkmale haben, bezüglich *[ihres/ihrer] [Metrik]*.

Durch Einsetzen der Metriken lassen sich aus dieser Schablone die Alternativhypothesen erstellen, wie z. B.:

$\mathbf{H_A^{BZ}}$ Paare, in denen die Programmierer unterschiedliche Persönlichkeitsmerkmale haben, unterscheiden sich von Paaren, in denen die Programmierer gleiche Persönlichkeitsmerkmale haben, bezüglich ihrer Bearbeitungszeit.

In der letzten Spalte von Tabelle 5.1 finden sich die Kurzschreibweisen aller Alternativhypothesen. Diese stehen dabei stets unter den Kurzschreibweisen der entsprechenden Nullhypothesen, welche sich durch logische Negation der Alternativhypothesen ergeben. Vor jeder der Kurzschreibweisen findet sich ein innerhalb dieses Kapitels eindeutiger Schlüssel über den wir die entsprechende Hypothese referenzieren werden. Innerhalb der Kurzschreibweisen ist der Name der Metrik durch einen korrespondierende Abkürzung ersetzt. Diese Abkürzung findet sich auch als hochgestellter Text im Schlüssel der Hypothese wieder. Die Indizes in der Kurschreibweise stehen für die in der Studie auftretenden Gruppen. Der Index Gleich steht dabei für die Gruppe mit Paaren, die aus Programmieren mit gleichen Persönlichkeitsmerkmalen bestand und der Index Ungleich für die Gruppe mit Paaren, die aus Programmierern mit unterschiedlichen Persönlichkeitsmerkmalen bestand.

5.3 Versuchsaufbau

Um die Hypothesen prüfen zu können, haben wir ein Experiment gemäß dem Inter-Subjekt-Entwurf mit einem Nachtest ausgearbeitet (vgl. [Chr07, S. 309] und [SCC02, S. 258]). Bei diesem Design erfolgt die Gruppeneinteilung zufällig. Weiter nimmt jeder Teilnehmer nur in einer Gruppe teil und liefert genau einen Datenpunkt. Die unabhängige Variable in unserem Aufbau ist die Zusammensetzung der Paare bezüglich der Persönlichkeitsmerkmale der Programmierer. Für die unabhängige Variable wurden zwei Ausprägungen betrachtet, nämlich Paare aus Programmieren mit unterschiedlichen und Paare aus Programmierern mit gleichen Persönlichkeitsmerkmalen. Die abhängigen Variablen sind die in Tabelle 5.1 zuvor aufgeführten Metriken.

5.4 Versuchsdurchführung

Für dieses Experiment haben wir Datenpunkte von 17 Programmiererpaaren gesammelt. Bei 20 der Teilnehmer handelt es sich um Studenten des an unserem Lehrstuhl gehaltenen Extreme Programming-Praktikums des Jahres 2007. Die weiteren 14 Teilnehmer

Metrik	*Null- & Alternativhypothese*	
	Schl.	*Kurzschreibweise*
Kategorie: Produktivität		
Bearbeitungszeit [min]	H_0^{BZ}	$BZ_{Gleich} = BZ_{Ungleich}$
	H_A^{BZ}	$BZ_{Gleich} \neq BZ_{Ungleich}$
Netto-Änderungen [SLOC]	$H_0^{NÄ}$	$NÄ_{Gleich} = NÄ_{Ungleich}$
	$H_A^{NÄ}$	$NÄ_{Gleich} \neq NÄ_{Ungleich}$
Anteil der Netto-Änderungen im Testcode [%]	H_0^{TC}	$TC_{Gleich} = TC_{Ungleich}$
	H_A^{TC}	$TC_{Gleich} \neq TC_{Ungleich}$
Anzahl der geänderten Dateien	H_0^{GD}	$GD_{Gleich} = GD_{Ungleich}$
	H_A^{GD}	$GD_{Gleich} \neq GD_{Ungleich}$
Kategorie: Qualität		
Änderung der Anzahl der Testfälle	$H_0^{\Delta TF}$	$\Delta TF_{Gleich} = \Delta TF_{Ungleich}$
	$H_A^{\Delta TF}$	$\Delta TF_{Gleich} \neq \Delta TF_{Ungleich}$
Änderung der Anweisungs-abdeckung [%]	$H_0^{\Delta AÜ}$	$\Delta AÜ_{Gleich} = \Delta AÜ_{Ungleich}$
	$H_A^{\Delta AÜ}$	$\Delta AÜ_{Gleich} \neq \Delta AÜ_{Ungleich}$
Anzahl der Akzeptanz-testläufe	H_0^{AT}	$AT_{Gleich} = AT_{Ungleich}$
	H_A^{AT}	$AT_{Gleich} \neq AT_{Ungleich}$
Kategorie: TGE-Prozess		
TGE-Prozesstreue [%]	H_0^{TGE}	$TGE_{Gleich} = TGE_{Ungleich}$
	H_A^{TGE}	$TGE_{Gleich} \neq TGE_{Ungleich}$
Anzahl der automatischen Umstrukturierungen	H_0^{AU}	$AU_{Gleich} = AU_{Ungleich}$
	H_A^{AU}	$AU_{Gleich} \neq AU_{Ungleich}$
Mittlerer Abstand der Testläufe [min]	H_0^{MAT}	$MAT_{Gleich} = MAT_{Ungleich}$
	H_A^{MAT}	$MAT_{Gleich} \neq MAT_{Ungleich}$
Variationskoeffizient des Abstands der Testläufe [%]	H_0^{VAT}	$VAT_{Gleich} = VAT_{Ungleich}$
	H_A^{VAT}	$VAT_{Gleich} \neq VAT_{Ungleich}$

Tabelle 5.1: Übersicht über die Hypothesen.

sind Studenten und Betreuer des Extreme Programming-Praktikums des Jahres 2007 der Universität Bonn.

Der Versuch wurde in Karlsruhe wie folgt durchgeführt: Bereits am letzten Tag des Extreme Programming-Praktikums wurden die Teilnehmer nach und nach paarweise in einen separaten Raum gebeten und es wurde ihnen eine Vertraulichkeitserklärung zur Unterschrift übergeben. Diese sicherte den Teilnehmern einen vertraulichen Umgang mit ihren persönlichen Daten und eine Veröffentlichung der Ergebnisse in anonymisierter Form zu. Im Gegenzug verpflichteten sich die Teilnehmer mit ihrer Unterschrift dazu, nicht mit Unbeteiligten über den Inhalt der Studie zu sprechen. Weiterhin informierte die Vertraulichkeitserklärung über den erwarteten Arbeitsaufwand von etwa vier bis fünf Stunden.

Nach der Vertraulichkeitserklärung haben wir den Teilnehmern der Keirsey-Test ausgehändigt. Wie es Keirsey [Kei98, S. 4 ff.] in seinem Buch vorsieht, wurde ihnen genügend Zeit gegeben, den Test in Ruhe auszufüllen. Direkt im Anschluss wurde der Test ausgewertet und den Teilnehmern die Beschreibung ihres Keirsey-Temperaments zu lesen gegeben (vgl. [KB90, S. 6 ff.]). Anschließend daran durften die Teilnehmer bewerten, wie gut das ermittelte Keirsey-Temperament ihrer Meinung nach auf sie zutrifft[3]. Danach konnten die Teilnehmer bis zum Tag des eigentlichen Experiments pausieren.

In der Zwischenzeit wurden die Teilnehmer anhand der Keirsey-Temperamente in Paare eingeteilt. Um diese Einteilung möglichst zufällig zu gestalten aber dennoch in etwa gleich große Gruppen zu erhalten, wurden Lose angefertigt, auf denen auf der einen Seite das Keirsey-Temperament stand und auf der anderen Seite die Identifikationsnummer des Teilnehmers. Diese Lose wurden mit der Identifikationsnummer nach unten auf einen Tisch verteilt. Dann wurde eine ansonsten am Experiment nicht beteiligte Person gebeten, daraus eine möglichst gleiche Anzahl an Paaren mit gleichen und unterschiedlichen Keirsey-Temperamenten zu bilden, ohne die Zettel dabei umzudrehen.

Am Tag des Experiments bekamen die Teilnehmer ihren Partner mitgeteilt und die Teilnehmerfragebögen ausgehändigt. Im Anschluss erhielten die Paare die Aufgabenstellung und durften anfangen, die in Abschnitt 5.6 genauer beschriebene Experimentaufgabe zu lösen. Sobald ein Paar der Meinung war, eine vollständig korrekte Lösung für die Aufgabe entwickelt zu haben, durften sie beim Experimentleiter einen Akzeptanztest beantragen. Für den Fall, dass das eingereichte Programm den Akzeptanztest bestand, war das betreffende Paar mit der Aufgabe fertig. Anderenfalls musste das Paar die Fehler im Programm korrigieren und sich zu einem weiteren Akzeptanztest melden. Dieser Prozess wurde solange wiederholt, bis das Paar eine fehlerfreie Lösung vorweisen konnte oder aufgab.

[3] Auf die Ergebnisse hierzu werden wir in Abschnitt 5.8.3 zu sprechen kommen.

In Bonn stand für die Ermittlung der Keirsey-Temperamente und das Experiment nur ein Tag zur Verfügung. Daher wurden die Keirsey-Temperamente der Teilnehmer in Bonn direkt am Morgen vor dem Experiment ermittelt. Anschließend wurden die Teilnehmer in die Mittagspause geschickt und die Einteilung der Paare nach dem gleichen Schema wie in Karlsruhe durchgeführt. Das eigentliche Experiment fand nach der Mittagspause statt und lief nach dem gleichen Experimentplan wie in Karlsruhe ab.

Ein Paar aus Bonn schaffte es nicht, den Akzeptanztest zu bestehen, und gab auf. Bei einem anderen Paar aus Bonn wurden wegen technischer Probleme die Protokolldateien von TestPulse beschädigt, so dass deren zuverlässige Auswertung nicht möglich ist. Beide Datenpunkte wurden aus dem Datensatz entfernt. Alle Angaben in den folgenden Abschnitten dieses Kapitels beziehen sich daher, sofern nicht explizit anders erwähnt, nur auf die Teilnehmer deren Datenpunkte im Datensatz verblieben sind.

5.5 Beschreibung der Teilnehmer

Die 14 Teilnehmer aus der Gleich-Gruppe hatten bis zum Zeitpunkt des Extreme Programming-Praktikums im Mittel 3,8 Jahre studiert, bei den 16 Teilnehmern der Ungleich-Gruppe waren es im Mittel 4,6 Jahre. Alle Teilnehmer studierten Informatik, vier von ihnen zusammen mit Mathematik. Die allgemeine Programmiererfahrung vor dem Praktikum belief sich in der Gleich-Gruppe auf durchschnittlich 7,8 Jahre, in der Ungleich-Gruppe auf durchschnittlich 7,0 Jahre. Davon hatten die Teilnehmer der Gleich-Gruppe im Durchschnitt 3,4 Jahre, die Teilnehmer der Ungleich-Gruppe 3,6 Jahre mit Java gearbeitet. Zwei der Teilnehmer aus der Gleich- und einer aus der Ungleich-Gruppe hatten jedoch vor dem Praktikum noch keinen Kontakt mit Java gehabt.

Mit Paarprogrammierung hatten die Teilnehmer vor den Extreme Programming-Praktikum insgesamt sehr wenig Erfahrung sammeln können. In der Gleich-Gruppe hatten gerade einmal zwei Teilnehmer zuvor im Paar programmiert, in der Ungleich-Gruppe waren es nur drei Teilnehmer. Etwas mehr Teilnehmer konnten Erfahrungen mit Testautomatisierung aufweisen. Hiermit hatten sechs Teilnehmer in der Gleich- und vier in der Ungleich-Gruppe bereits zuvor gearbeitet. Mit JUnit hatten vier Teilnehmer in der Gleich- und ebenfalls vier in der Ungleich-Gruppe Erfahrung. Ebenso viele Teilnehmer pro Gruppe hatten vor den Extreme Programming-Praktikum mit testgetriebener Entwicklung entwickelt.

Industrielle Programmiererfahrung war zwar bei einigen wenigen Teilnehmern vorhanden, war aber insgesamt so gering, dass sie an dieser Stelle nicht weiter aufgeführt wird. Der interessierte Leser sei stattdessen auf den Anhang B verwiesen. Wir haben die Gruppen bezüglich ihrer Vorbildung mit zweiseitigen Wilcoxon-Tests verglichen. Bei

Gruppe	*Basistemperament*			
	NF	*NT*	*SJ*	*SP*
Gleich	4 (8)	8 (0)	2 (26)	0 (0)
Ungleich	5 (7)	5 (6)	4 (16)	2 (7)
Summe	9 (15)	13 (6)	6 (42)	2 (7)

Tabelle 5.2: Verteilung der Basistemperamente in den Gruppen.

einem Signifikanzniveau von 5 Prozent konnten wir dabei keine Unterschiede feststellen. Lediglich bei der der Studiendauer ergab sich mit p-Wert von 0,057 ein beinahe signifikantes Ergebnis. Die p-Werte aller hierzu durchgeführten Hypothesentests finden sich im Anhang in Tabelle B.1.

Außer den Daten zur Vorbildung waren als Grundlage für die Paareinteilung die Keirsey-Basistemperamente der Teilnehmer ermittelt worden. Die Verteilung der Basistemperamente ist in Tabelle 5.2 aufführt. Zum Vergleich haben wir in Klammern die entsprechenden Zahlen aus der Studie von Sfetsos und Kollegen angegeben [SSAD09, Tabelle 2]. Wie man in der Tabelle sehen kann, waren 13 der insgesamt 30 Teilnehmer Rationalisten (NT) gefolgt von Idealisten (NF) mit neun und den Beschützern (SJ) mit sechs Teilnehmern. Kunsthandwerker (SP) waren nur zwei der Teilnehmer. Wie man in der Tabelle ebenfalls erkennen kann, waren die Basistemperamente nicht gleichmäßig auf die beiden Gruppen verteilt. So befand sich beispielsweise in der Gleich-Gruppe kein einziger Kunsthandwerker.

Bedingt durch die geringe Anzahl an Kunsthandwerkern im Zusammenspiel mit der zufälligen Zuordnung der Paare gelang es nicht alle denkbaren Kombinationen zu besetzen. Mit welcher Häufigkeit die Kombinationen auftraten, ist in Tabelle 5.3 dargestellt. Wiederum sind die Zahlen aus der Studie von Sfetsos und Kollegen in Klammern angegeben [SSAD09, Tabelle 3]. Wie man dort sieht, gab es in unserer Studie kein Paar aus zwei Kunsthandwerkern und keines aus einem Rationalisten und einem Kunsthandwerker. Wir werden dies genau wie die ungleichmäßige Verteilung der Basistemperamente in Abschnitt 5.8 diskutieren.

5.6 Aufgabe Wettbüro

Die Aufgabe Wettbüro wurde von Marmé in einer Studienarbeit [Mar07] für die Verwendung in dieser Studie entwickelt. Bei dieser Aufgabe soll die Softwarelösung eines Wettbüros, das sich auf Fußballwetten spezialisiert hat, um drei verschieden Arten von

Basistemp. 1	*Basistemp. 2*			
	NF	*NT*	*SJ*	*SP*
NF	2 (4)	3 (0)	1 (6)	1 (1)
NT	–	4 (0)	2 (5)	0 (1)
SJ	–	–	1 (13)	1 (5)
SP	–	–	–	0 (0)

Tabelle 5.3: Verteilung der Basistemperamente in den Paaren.

Metrik	*Codebereich*		
	Anwendung	*Test*	*Gesamt*
Anzahl der Klassen & Schnittstellen	19	12	31
Anzahl der Methoden	85	131	216
Länge [SLOC]	776	1163	1939
Anweisungsabdeckung [%]	91	–	–
Anzahl der Testfälle	–	111	–

Tabelle 5.4: Kennzahlen der Aufgabe Wettbüro

Systemwetten erweitert werden. Systemwetten sind Wetten, bei denen gleichzeitig auf den Ausgang mehrerer Spiele gesetzt werden kann. Das an die Teilnehmer ausgegebene Programm enthält bereits eine abstrakte Oberklasse für alle Wetten, von der die zu implementierenden Systemwetten erben sollen. Somit steht die Schnittstelle für die Systemwetten bereits fest. Zudem zeigt die im Programm bereits implementierte Einzelwette exemplarisch wie die abstrakte Oberklasse zu nutzen ist.

Wie in Tabelle 5.4 aufgelistet, enthält der Anwendungscode des Programms in der an die Teilnehmer ausgegebenen Version 19 Klassen und Schnittstellen mit 85 Methoden und 776 Zeilen nach dem in Abschnitt 2.4.1.2 beschriebenen Verfahren bereinigten Code. Der dazugehörige Testcode besteht aus 12 Klassen und Schnittstellen mit 131 Methoden und 1163 Zeilen bereinigtem Code. Im Testcode sind 111 Testfälle definiert, die 91 Prozent der Anweisungen des Anwendungscodes abdecken.

Mit dem Programm erhielten die Teilnehmer eine Aufgabenbeschreibung, in der neben der Problemstellung eine genaue Erklärung der Systemwetten inklusive Rechenbeispielen enthalten ist.

5.7 Ergebnisse

Dieser Abschnitt beschreibt die Ergebnisse der Auswertung des Datensatzes. Die in Abschnitt 5.2 aufgestellten Hypothesen wurden mit zweiseitigen Wilcoxon-Tests und einem Signifikanzniveau von 5 Prozent überprüft.

Wie schon in Kapitel 4 werden die *p*-Werte zu den Hypothesentests im Text genannt. Die Lage- und Streuungsmaße zu den erhobenen Metriken erwähnen wir dagegen nur, wenn sie uns besonders interessant erscheinen. Auch zu diesem Kapitel existiert eine vollständige Übersicht über Lage- und Streuungsmaße. Sie findet sich zusammen mit einer Übersicht über alle *p*-Werte in Anhang B.

5.7.1 Produktivität

Bei der Studie von Sfetsos und Kollegen [SSAD09] waren die Paare mit ungleichen Basistemperamenten der Partner deutlich schneller fertig geworden als die Paare mit gleichem Basistemperament der Partner. Wir wollen in den folgenden Abschnitten sehen, ob sich ein ähnlicher Effekt in unserer Studie zeigt.

5.7.1.1 Bearbeitungszeit

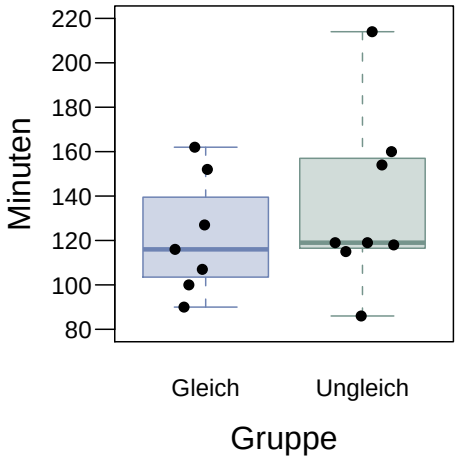


Abbildung 5.1: Bearbeitungszeit.

Wir beginnen die Auswertung der Produktivitätsmetriken mit der Bearbeitungszeit, die in Abbildung 5.1 dargestellt ist. Insgesamt sehen die Ergebnisse der beiden Gruppen recht ähnlich aus. Die Boxen überlappen sich weitgehend und die Mediane der beiden

Gruppen liegen mit 116 Minuten bei der Gleich- und 119 Minuten bei der Ungleich-Gruppe nah beieinander. Lediglich ein sehr langsames Paar in der Ungleich-Gruppe, das etwas über 3,5 Stunden gebraucht hat, um die Aufgabe zu lösen, fällt aus dem Gesamtbild heraus. Dementsprechend lässt sich kein signifikanter Unterschied zwischen den Gruppen nachweisen, wie das Ergebnis des Wilcoxon-Tests mit einem p-Wert von 0,524 zeigt. Damit kann auch die Nullhypothese H_0^{BZ} nicht abgelehnt werden.

5.7.1.2 Änderungen am Code

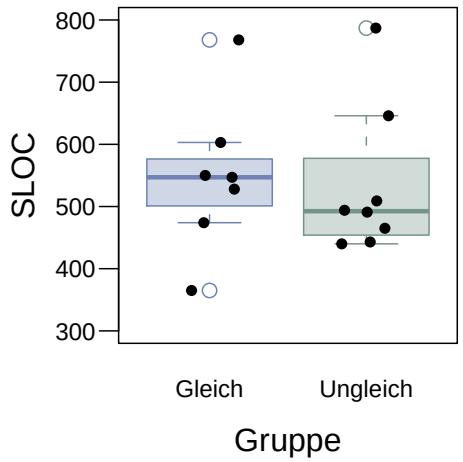


Abbildung 5.2: Netto-Änderungen

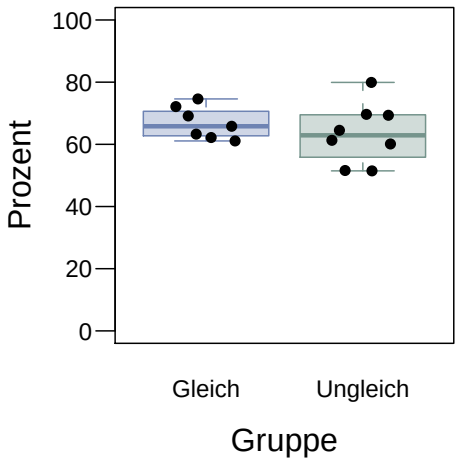


Abbildung 5.3: Anteil der Netto-Änderungen im Testcode.

Ähnlich gering wie bei der Bearbeitungszeit sind die Unterschiede bei den Netto-Änderungen an den Programmen. Wie in Abbildung 5.2 zu sehen, überlappen sich auch hier die Boxen der beiden Gruppen weitgehend. Dessen ungeachtet liegen die Mediane hier etwas weiter auseinander. Der Wilcoxon-Test für den Vergleich der Netto-Änderungen der beiden Gruppen liefert einen p-Wert von 0,536. Damit ist der Unterschied zwischen den Gruppen nicht signifikant und folglich lässt sich auch die Nullhypothese $H_0^{NÄ}$ nicht ablehnen.

Betrachten wir nun wie sich die Netto-Änderungen der Paare auf Anwendungs- und Testcode verteilen. In Abbildung 5.3 ist der Anteil der Netto-Änderungen dargestellt, die in Dateien des Testcodes stattfanden. Es fällt zunächst auf, dass in allen Paaren mehr Änderungen im Test- als im Anwendungscode durchgeführt wurden. Das Paar mit dem geringsten Anteil führte immer noch 51,6 Prozent der Netto-Änderungen im Testcode durch. Beim Paar mit dem höchsten Anteil können 79,9 Prozent der Netto-Änderungen

dem Testcode zugeordnet werden. Diese beiden Extremwerte finden sich beide unter den Paaren aus der Ungleich-Gruppe. In der Gleich-Gruppe schwanken die Werte nur zwischen 61,1 und 74,6 Prozent. Dennoch liegen die Mediane der beiden Gruppen mit 65,8 Prozent in der Gleich- und 62,9 Prozent in der Ungleich-Gruppe beinahe auf gleicher Höhe. Daher verwundert es nicht, dass sich bezüglich des Anteils der Netto-Änderungen im Testcode kein signifikanter Unterschied zwischen den beiden Gruppen zeigen lässt ($p = 0{,}463$). Wir nehmen daher weiter an, dass die Nullhypothese H_0^{TC} gilt.

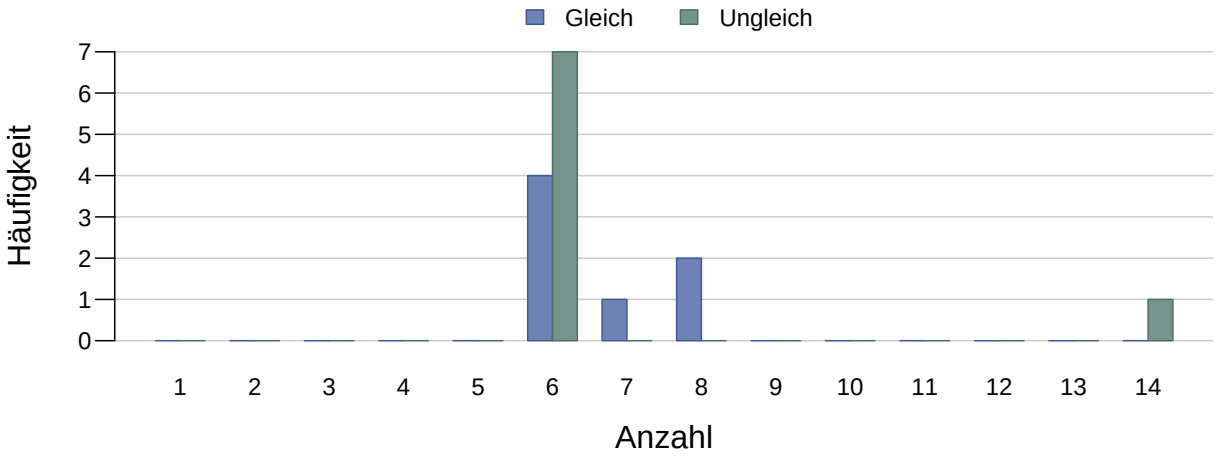


Abbildung 5.4: Anzahl der geänderten Dateien

Die letzte Produktivitätsmetrik, der wir uns widmen wollen, ist die Anzahl der geänderten Dateien. Wie man in Abbildung 5.4 erkennen kann, wurden zumeist sechs bis acht Dateien geändert. Nur ein Paar aus der Ungleich-Gruppe änderte insgesamt 14 Dateien. Ansonsten liegen die Werte in den beiden Gruppen so nah beieinander, dass sich kein signifikanter Unterschied zwischen den Gruppen nachweisen lässt ($p = 0{,}334$). Wir müssen daher davon ausgehen, dass sich die Gruppen in Wirklichkeit in diesem Punkt nicht unterscheiden und die Nullhypothese H_0^{GD} somit wahr ist.

5.7.2 Qualität

In der Studie von Sfetsos und Kollegen [SSAD09] waren die Paare mit ungleichen Basistemperamenten der Partner den Paare mit gleichem Basistemperament der Partner auch bei der Qualität überlegen. Nach unseren Ergebnissen zur Produktivität darf man jedoch skeptisch sein, ob sich in unserer Studie ein Unterschied bei der Qualität nachweisen lässt.

5.7.2.1 Änderungen an den Testfällen

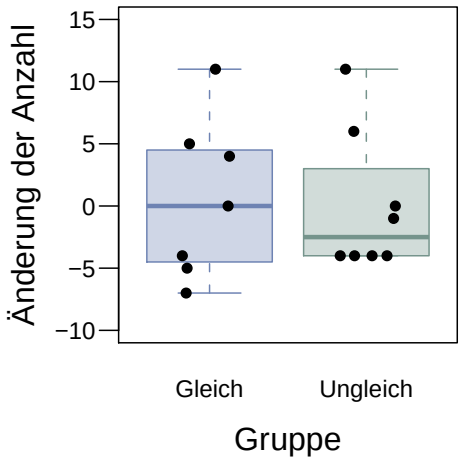


Abbildung 5.5: Änderung der Anzahl der Testfälle.

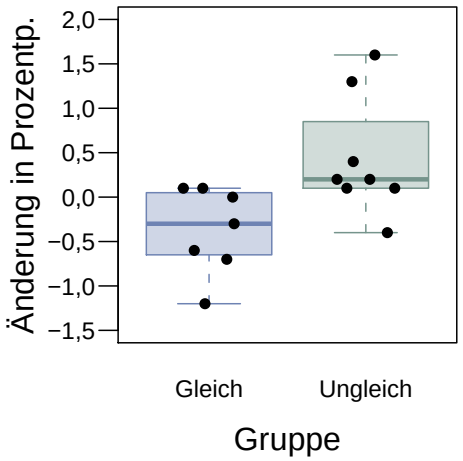


Abbildung 5.6: Änderung der Anweisungsabdeckung

Betrachten wir zuerst die Änderung der Anzahl der Testfälle in Abbildung 5.5. Hier fällt auf, dass bei mehr als der Hälfte der Paare die Anzahl der ursprünglich 111 Testfälle abgenommen hat. Der Hauptgrund hierfür ist im mit der Aufgabenstellung ausgegebenen Programm zu finden. Durch Umstrukturieren des Programmcodes ergab sich für die Paare die Möglichkeit, die abstrakte Oberklasse, welche die Schnittstelle für die Wetten vorgab mit der bereits implementierten konkreten Klasse für die Einzelwette und den zu implementierten Klassen für die Systemwetten zu einer einzigen, komplexen Klasse zusammenzufassen. Eine Möglichkeit, die von vielen der Paare erkannt und zumindest teilweise umgesetzt wurde. Werden die Klassen im Anwendungscode derart zusammengefasst, dann lassen sich auch die entsprechenden Testklassen zusammenlegen, wodurch sich die Anzahl der Testmethoden bzw. Testfälle reduziert. Um dann wieder auf die anfänglichen 111 Testfälle zu kommen, mussten Paare, die diese Umstrukturierung durchgeführt hatten, entsprechend mehr neue Testfälle schreiben.

Wenn man einen zweiten Blick auf Abbildung 5.5 wirft, um die Unterschiede zwischen den Gruppen zu suchen, muss man feststellen, dass keine großen Unterschiede sichtbar sind. Zwar liegt der Median der Gleich-Gruppe exakt bei 0 während er in der Ungleich-Gruppe bei $-2{,}5$ liegt, doch die Boxen der beiden Gruppen überlappen sich quasi vollständig. Folglich lässt sich die Nullhypothese $H_0^{\Delta TF}$ aufgrund des p-Werts des entsprechenden Wilcoxon-Tests von 0,953 nicht ablehnen und wir nehmen daher an,

dass sich die zu den Gruppen gehörigen Populationen in diesem Punkt nicht unterscheiden.

Gehen wir nun zur Betrachtung der Änderung der Anweisungsabdeckung über, die in Abbildung 5.6 dargestellt ist. Dort ist ein deutlicher Unterschied zwischen den Gruppen sichtbar. In der Gleich-Gruppe konnten vier Paare die hohe anfängliche Anweisungsabdeckung von 91 Prozent nicht halten, während sie in der Ungleich-Gruppe von sieben Paaren zumindest leicht verbessert wurde. Die Mediane der Änderung der Anweisungsabdeckung von $-0{,}3$ in der Gleich- und 0,2 in der Ungleich-Gruppe bestätigen den ersten Eindruck, genauso wie die Tatsache, dass sich die Boxen der beiden Gruppen nicht überlappen. Der Vergleich der beiden Gruppen mittels eines Wilcoxon-Tests liefert einen p-Wert von 0,012 und weist damit auf einen signifikanten Unterschied zwischen den Gruppen hin. Wir lehnen damit die Nullhypothese $H_0^{\Delta AÜ}$ ab und nehmen stattdessen die Alternativhypothese $H_A^{\Delta AÜ}$ an.

5.7.2.2 Anzahl der Akzeptanztestläufe

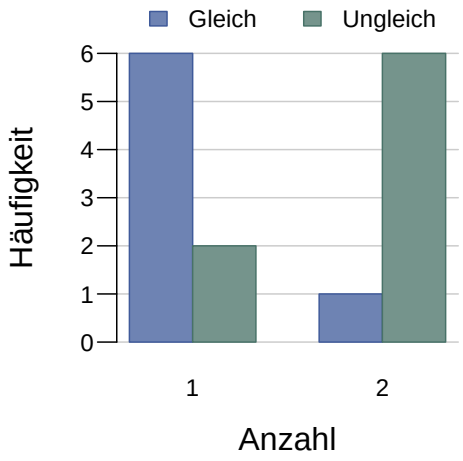


Abbildung 5.7: Anzahl der Akzeptanztestläufe

Gegen Ende des vorherigen Abschnitts konnten wir zeigen, dass es zwischen den Gruppen einen signifikanten Unterschied bei der Änderung der Anweisungsabdeckung gibt. Um eine mögliche Erklärung dieses Unterschieds zu finden, sehen wir uns hierzu zunächst die Anzahl der Akzeptanztestläufe in Abbildung 5.7 an. Von den sieben Paaren der Gleich-Gruppe haben sechs einen und eines zwei Akzeptanztestläufe zum Bestehen gebraucht. Von den acht Paaren der Ungleich-Gruppe bestanden dagegen nur zwei Paare

den Akzeptanztest beim ersten Mal. Die verbleibenden sechs Paare mussten alle zu einem zweiten Akzeptanztest antreten. Dieser Unterschied ist nicht nur optisch auffällig, sondern auch signifikant ($p = 0{,}027$). Damit lässt sich die Nullhypothese H_0^{AT} zugunsten der Alternativhypothese H_A^{AT} ablehnen.

Der mögliche Zusammenhang mit der Änderung der Anweisungsabdeckung liegt im Akzeptanztest selbst verborgen. Der zu dieser Aufgabe gehörige automatische Akzeptanztest testet die Lösungen über eine bereits im ausgegebenen Programmcode enthaltene öffentliche Schnittstelle, welche die Teilnehmer nicht ändern durften. Der Akzeptanztest umfasst dabei 195 Testfälle, wobei die einzelnen Testfälle feingranular gehalten sind und im Fehlerfall eine klar verständliche Fehlermeldung ausgeben. Diese Fehlermeldungen wurden den Paaren ausgehändigt, falls der Akzeptanztest fehlschlug.

Dadurch hat der Akzeptanztest möglicherweise den Seiteneffekt, dass er Paaren, die den Akzeptanztest beim ersten Versuch nicht bestehen, über Gebühr dabei unterstützt, die Qualität ihrer eigenen Tests zu verbessern. Zudem wurden die Paare, falls ihre Lösung den Akzeptanztest nicht bestand, vom Experimentleiter gebeten, ihr korrigiertes Programm ausführlich zu testen, bevor sie sich zum nächsten Versuch anmelden. Vermutlich ist die Anweisungsabdeckung der Ungleich-Gruppe also nur besser, weil deutlich mehr Paare der Ungleich-Gruppe zwei Akzeptanztestläufe benötigt haben. Um dies vollständig zu verhindern, hätten wir den Paaren nur mitteilen dürfen, ob ihre Lösung den Akzeptanztest bestanden hat oder nicht. Gegen diese Variante hatten wir uns jedoch bewusst entschieden, da wir der Meinung waren, dass ein Ablehnen einer Lösung ohne jeglichen Hinweis auf das Problem sehr frustrierend sein kann und sich damit eventuell die Quote der Abbrecher erhöht.

5.7.3 TGE-Prozess

Abschließend betrachten wir die Metriken zum testgetriebenen Entwicklungsprozess. Diese wurden bei Sfetsos und Kollegen [SSAD09] nicht untersucht, weswegen wir bezüglich der Ergebnisse keine Erwartungshaltung haben.

5.7.3.1 TGE-Prozesstreue

Sehen wir uns als erstes die TGE-Prozesstreue der Paare in Abbildung 5.8 an. Abgesehen von ein paar Ausreißern nach unten, ist die TGE-Prozesstreue der Paare hoch. Bei vier Paaren liegt die TGE-Prozesstreue über der 90-Prozentmarke. Besonders auffällig ist dabei das Paar mit der höchsten TGE-Prozesstreue von 99,4 Prozent. Ziehen wir Abbildung 5.9 für eine detailliertere Betrachtung der TGE-Prozesstreue mit hinzu, dann fällt noch das dritte Paar von rechts in der Ungleich-Gruppe auf, das den größten Anteil

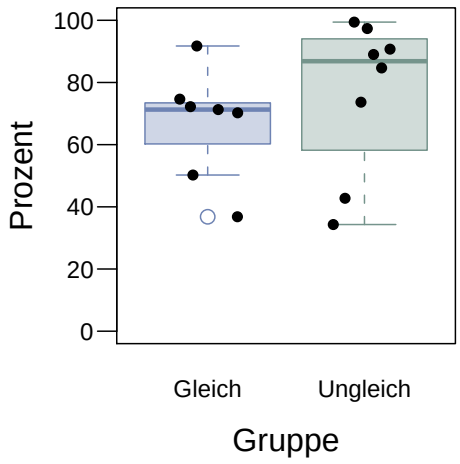


Abbildung 5.8: TGE-Prozesstreue

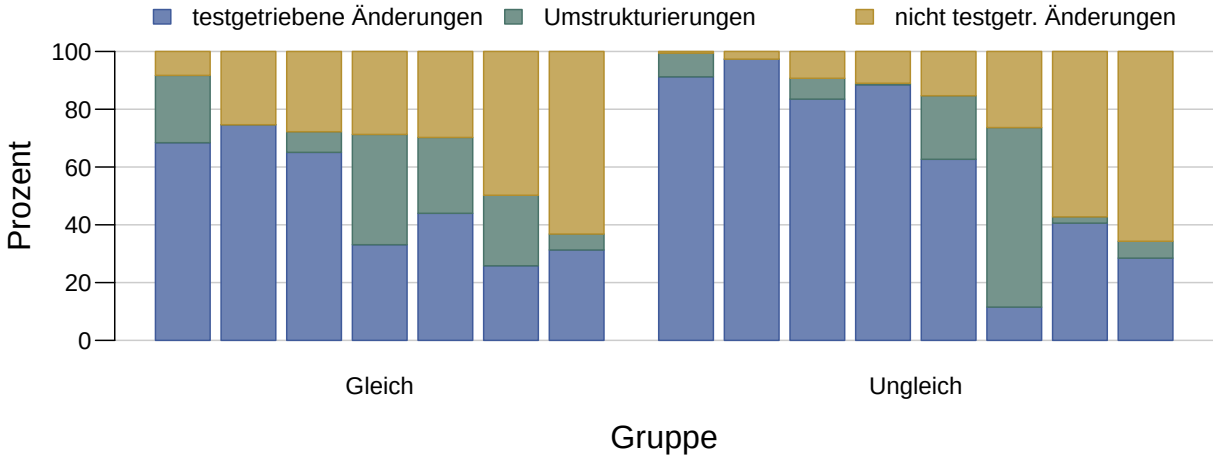


Abbildung 5.9: Anteile der testgetriebenen Änderungen, Umstrukturierungen und nicht testgetriebenen Änderungen.

der TGE-Prozesstreue durch Umstrukturierungen erreicht hat.

Weiterhin ergibt sich aus beiden Abbildungen der Eindruck, dass die TGE-Prozesstreue in der Gleich-Gruppe etwas niedriger ist als in der Ungleich-Gruppe. Dieser Eindruck wird durch die Lage der Mediane verstärkt. In der Gleich-Gruppe liegt er bei 71,3 Prozent während er in der Ungleich-Gruppe bei 86,8 Prozent liegt. Dennoch ist der sichtbare Unterschied zwischen den Gruppen nicht signifikant, wie der p-Wert des dazugehörigen Wilcoxon-Tests von 0,281 zeigt. Aus diesem Grund können wir die Nullhypothese H_0^{TGE} nicht ablehnen und müssen davon ausgehen, dass der beobachtete Unterschied rein zufällig zustande gekommen ist.

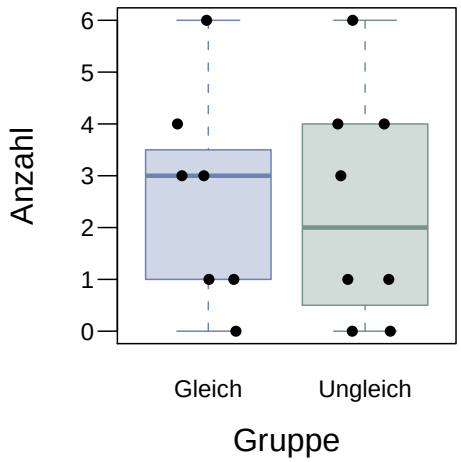


Abbildung 5.10: Anzahl der automatischen Umstrukturierungen.

5.7.3.2 Anzahl der automatischen Umstrukturierungen

Als nächstes wollen wir uns die Anzahl der automatischen Umstrukturierungen anschauen. Wie man in Abbildung 5.10 sehen kann, gibt es diesbezüglich keinen auffälligen Unterschied zwischen den beiden Gruppen. Die Boxen sind in etwa gleich groß und in etwa auf gleicher Höhe. Ebenso liegen die Mediane mit drei automatischen Umstrukturierungen bei der Gleich- und zwei automatischen Umstrukturierungen bei der Ungleich-Gruppe nur unwesentlich auseinander. Tatsächlich zeigt der dazugehörige Wilcoxon-Test, dass es keinen signifikanten Unterschied zwischen den Gruppen bezüglich der Anzahl der automatischen Umstrukturierungen gibt ($p = 0{,}906$). Darum gehen wir davon aus, dass die Nullhypothese H_0^{AU} gilt.

5.7.3.3 Testläufe

Wenden wir uns zunächst Abbildung 5.11 zu, welche die Anzahl der Testläufe darstellt. Wie man dort sehen kann, ist der mittlere Abstand der Testläufe in der Gleich-Gruppe tendenziell größer als in der Ungleich-Gruppe. Dabei ist der Unterschied immerhin so groß, dass es nur zu einer geringen Überlappung der beiden Boxen kommt. Die Mediane liegen mit einem Wert von 3,5 Minuten bei der Gleich- und 2,9 Minuten bei der Ungleich-Gruppe etwa 37 Sekunden auseinander. Doch trotz der sichtbaren Unterschiede liegt der p-Wert für den Vergleich der beiden Gruppen mit 0,232 über dem Signifikanzniveau von 5 Prozent. Damit gelingt es uns nicht die Nullhypothese H_0^{MAT} abzulehnen.

Betrachten wir als letzte Metrik zum TGE-Prozess den in Abbildung 5.12 dargestellten Variationskoeffizienten der Testläufe. Dabei fällt ins Auge, dass die Variationskoeffizienten bei allen Paaren über 100 Prozent liegen, d. h., dass die Standardabweichung der Testläufe immer deren mittleren Abstand überschreitet. Ansonsten erkennt man ei-

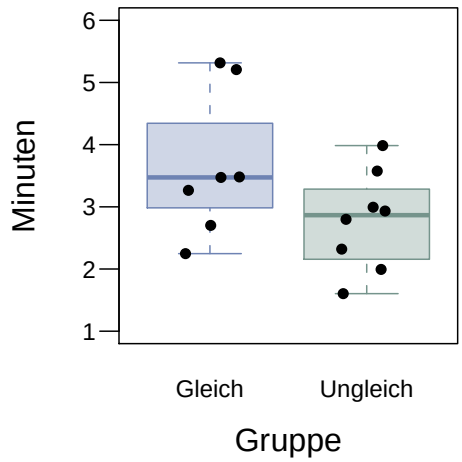


Abbildung 5.11: Mittlerer Abstand der Testläufe

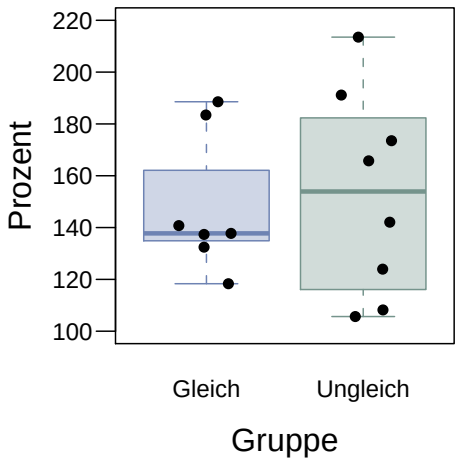


Abbildung 5.12: Variationskoeffizient der Testläufe

ne leicht höhere Streuung der Datenpunkte auf Seiten der Ungleich-Gruppe, wodurch die Box der Gleich-Gruppe komplett von der Box der Ungleich-Gruppe überdeckt wird. Daher ist es nicht sonderlich überraschend, dass der sichtbare Unterschied der Gruppen sich als nicht signifikant erweist ($p = 0{,}867$) und wir somit an der Nullhypothese H_0^{VKT} festhalten.

5.8 Validität der Studie

Wie auch für die vorherige Studie gelten für diese Studie Validität dieser Studie gewisse Einschränkungen, die wir in diesem Abschnitt diskutieren. Wir halten uns dabei an die in Abschnitt 4.7 eingeführte Untergliederung in die Validität des Konstrukts sowie die statistische, interne und externe Validität.

5.8.1 Validität des Konstrukts

Die größte Gefahr für die Validität des Konstrukts geht von der Wahl des Keirsey-Temperaments als Maß für die Persönlichkeitsmerkmale der Teilnehmer aus. Eine häufig geäußerte Kritik am Keirsey-Temperamenttest ist, dass die 16 Temperamente eine viel zu grobe Kategorisierung darstellen, als dass sie in der Lage wären, die Persönlichkeit eines Menschen zutreffend zu beschreiben. Wir haben die Teilnehmer daher selbst beurteilen lassen, wie zutreffend sie sich durch ihr Keirsey-Temperament beschrieben fühlen. Dazu haben wir ihnen mit dem Ergebnis des Keirsey-Temperamenttests die detaillierte Beschreibung ihres Temperaments [KB90] zu lesen gegeben und sie anschließend gebeten, den Satz „Die Beschreibung meines Temperaments ist…“ mit *zutreffend*, *eher zu-*

Gruppe	*zutreff-end*	*eher zu-treffend*	*eher nicht zu-treffend*	*nicht zu-treffend*
Gleich	10	4	0	0
Ungleich	5	6	4	1
Summe	15	10	4	1

Tabelle 5.5: Verteilung der Antworten zu dem zu vervollständigenden Satz „Die Beschreibung meines Temperaments ist…“.

Klasse	*automatisch*		*manuell*			
			ohne Konflikt		*mit Konflikt*	
testgetriebene Änderungen	393	60,6 %	0	0,0 %	2	0,3 %
Umstrukturierungen	56	8,6 %	4	0,6 %	8	1,2 %
nicht testgetr. Änderungen	–		178	27,4 %	8	1,2 %
Summe	449	69,2 %	182	28,0 %	18	2,8 %

Tabelle 5.6: Anteile der automatisch, manuell ohne Konflikt und manuell mit Konflikt klassifizierten Änderungen.

treffend, *eher nicht zutreffend* oder *nicht zutreffend* zu ergänzen. Wie wir in Tabelle 5.5 erkennen können, war die Zustimmung zur Beschreibung des ermittelten Temperaments groß.

Aufgrund des sogenannten Forer-Effekts ist aber dennoch nicht völlig ausgeschlossen, dass die Beschreibungen unzutreffend sein können und damit das Keirsey-Temperament als Maß für die Persönlichkeit ungeeignet ist. Der Forer-Effekt beschreibt die Neigung vieler Menschen, ungenauen und allgemeingültigen Beschreibungen der eigenen Persönlichkeit zuzustimmen. Forer [For49] hatte in einem Experiment 39 Studenten einen Persönlichkeitstest machen lassen, die Ergebnisse aber verworfen und allen Teilnehmern die gleiche aus einem Astrologiebuch entlehnte Kurzbeschreibung ihrer Persönlichkeit ausgehändigt. Danach bat er die Studenten, auf einer Likert-Skala von 0 (schlecht) bis 5 (perfekt) zu bewerten, wie gut die Kurzbeschreibung ihre Persönlichkeit wiedergibt. Der Durchschnitt der Antworten lag dabei über 4. Daher bleiben trotz der hohen Zustimmung unserer Teilnehmer zu ihrem Temperament Zweifel an der Gültigkeit des Keirsey-Temperaments als Maß für die Persönlichkeit der Teilnehmer.

Weiter gelten für diese Studie die gleichen Gefahren für die Validität des Konstrukts bezüglich der Zeiterfassung durch die Teilnehmer und der manuellen Klassifizierung der Änderungen wie schon für die Studie zuvor. Um eine hohe Genauigkeit der Zeiterfassung zu erreichen, haben wir während des Experiments immer wieder die Zeitprotokolle der Teilnehmer überprüft und die Teilnehmer in ihren Pausen angehalten, diese Pausen präzise zu erfassen. Damit wir Fehler bei der manuellen Klassifizierung möglichst ausschließen können, haben wir diese, wie bereits in Abschnitt 4.7.1 beschrieben, nach dem Vier-Augen-Prinzip durchgeführt. Wie in Tabelle 5.6 zu sehen ist, mussten wir insgesamt 30,8 Prozent der Änderungen manuell klassifizieren. Hierbei kam es bei lediglich 2,8 Prozent aller Änderungen bzw. 9 Prozent der von Hand zu klassifizierenden Änderungen zu Konflikten.

5.8.2 Statistische Validität

Zur Auswertung der Hypothesen wurde wie schon in der Studie zuvor der zweiseitige Wilcoxon-Test eingesetzt. Das Signifikanzniveau betrug 5 Prozent wodurch auch die Wahrscheinlichkeit eines Fehlers der 1. Art nach oben beschränkt ist.

Damit wir herausbekommen, wie hoch die Wahrscheinlichkeit für einen Fehler 2. Art maximal sein kann, müssen wir zunächst die minimale Güte des Wilcoxon-Tests abschätzen. Für die Abschätzung der Güte rechnen wir mit dem für die Studie festgelegten Signifikanzniveau von 5 Prozent und einer Gruppengröße von 7,467, was dem harmonischen Mittel der beiden tatsächlichen Gruppengrößen von 7 und 8 entspricht (vgl. [Coh88, S. 42]). Mit diesen Parametern lässt sich die in Abbildung 5.13 dargestellte Kur-

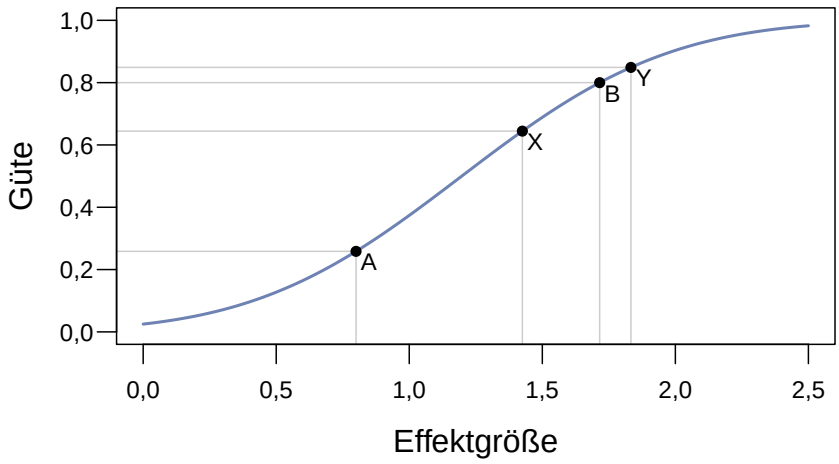


Abbildung 5.13: Zusammenhang zwischen Effektgröße und Güte für den Wilcoxon-Test bei einer Gruppengröße von 7,467, einem Signifikanzniveau von 0,05 und einer ARE von 0,864.

ve für den Zusammenhang von Effektgröße und Güte bestimmen. Wie durch Punkt A markiert, ergibt sich daraus für den Wilcoxon-Test bei einer großen Effektgröße von 0,8 für die Güte eine untere Schranke von 25,6 Prozent. Dies ist weit von der üblicherweise gewünschten Güte von 80 Prozent [Coh88, S. 531] entfernt und bedeutet, dass im ungünstigsten Fall eine Chance von 74,4 Prozent besteht, dass große Effekte nicht entdeckt werden. Eine Güte von 80 Prozent wird im ungünstigsten Fall, wie an Punkt B zu sehen, erst für eine Effektgröße von 1,716 erreicht.

Obwohl die Güte für diese Studie relativ gering ist, ist sie trotzdem ausreichend, um Effekte nachzuweisen, die so groß sind, wie sie Sfetsos und Kollegen [SSAD09] beobachten konnten. Wie bei der Diskussion der verwandten Arbeiten in 3.1.2 ausgeführt, lagen in deren Studie die beobachteten Effektgrößen zwischen 1,425 und 1,833. Die Güte beträgt für diese Effektgrößen mindestens zwischen 64,4 und 84,9 Prozent, wie die Punkte X und Y in Abbildung 5.13 veranschaulichen.

5.8.3 Interne Validität

Wie in der Studie zuvor stellt auch hier Sterblichkeit eine Bedrohung für die interne Validität dar. Wie bereits in Abschnitt 5.4 besprochen, mussten wir die Datenpunkte von zwei Paaren aus dem Datensatz herausnehmen. Ein Paar aus der Ungleich-Gruppe konnte den Akzeptanztest nicht bestehen und gab deswegen auf. Bei einem Paar aus der Gleich-Gruppe waren wegen technischer Probleme die Protokolldateien von TestPulse so stark beschädigt, dass sie sich nicht auswerten ließen. Da letzteres Paar den Akzeptanztest bestanden hatte, ist davon auszugehen, dass dessen Leistung besser war als die

Gruppe	*trifft zu*	*trifft eher zu*	*trifft eher nicht zu*	*trifft nicht zu*
Gleich	0	1	5	8
Ungleich	0	3	5	8
Summe	0	4	10	16

Tabelle 5.7: Verteilung der Bewertungen der Aussage „Ich war aufgeregt, weil ich an einem Experiment teilnahm und ich konnte deshalb nicht so programmieren, wie ich es gewohnt bin.“.

Gruppe	*trifft zu*	*trifft eher zu*	*trifft eher nicht zu*	*trifft nicht zu*
Gleich	13	1	0	0
Ungleich	10	6	0	0
Summe	23	7	0	0

Tabelle 5.8: Verteilung der Bewertungen der Aussage „Die Programmierung im Experiment hat mir gefallen.“.

des Paares, das vorzeitig abgebrochen hatte. Daher kann es durch die Eliminierung der zwei Datenpunkte aus dem Datensatz zumindest bei manchen der ausgewerteten Metriken zu einer Begünstigung der Ungleich-Gruppe gekommen sein.

Bereits in Tabelle 5.2 in Abschnitt 5.5 haben wir gezeigt, dass die Anteile der vier Basistemperamente in den beiden Gruppen nicht gleich ist. Es könnte daher sein, dass ein Teil der Beobachtungen nicht auf die unterschiedliche Zusammenstellung der Paare in den beiden Gruppen zurückzuführen ist, sondern auf durch die Basistemperamente der Teilnehmer direkt bedingte Unterschiede im Programmierverhalten. Dies hätte sich jedoch für unsere geringe Anzahl an Teilnehmern nur durch eine nicht zufällige Verteilung der Teilnehmer auf die Gruppen und Paare verhindern lassen, was aber ebenso negative Auswirkungen auf die Validität der Studie gehabt hätte.

5.8.4 Externe Validität

Für die externe Validität dieser Studie gelten zunächst einige Einschränkungen, die auch schon für die Studie zuvor gültig waren: Die Teilnehmer wurden nach ihrer Verfügbarkeit ausgewählt, deswegen sind die Ergebnisse nur auf Populationen übertragbar, die in ihren Eigenschaften denen unserer Teilnehmer ähnlich sind. Ferner lässt die geringe Dauer des Experiments keine zuverlässigen Aussagen über Langzeiteffekte zu. Auch sollten die Ergebnisse nur auf objektorientierte Sprachen übertragen werden, die mit Java vergleichbare Eigenschaften aufweisen und für die ein Testrahmenwerk wie JUnit zur Verfügung steht.

Eine weitere Einschränkung der externen Validität ergibt sich dadurch, dass, wie man schon in Tabelle 5.3 in Abschnitt 5.5 sehen konnte, in unserem Datensatz nicht alle möglichen paarweisen Kombinationen an Basistemperamenten vorkommen. Hiergegen hätte aber noch nicht einmal der Verzicht auf die zufällige Verteilung der Teilnehmer etwas gebracht, da es von einigen Basistemperamenten nicht genügend Teilnehmer gab, um überhaupt alle Kombinationen bilden zu können. Daher müssen wir diese Einschränkung der externen Validität als gegeben hinnehmen.

Zusätzlich dazu kann die externe Validität der Studie eingeschränkt sein, wenn die Teilnehmer nicht ihr übliches Arbeitsverhalten gezeigt haben, weil sie aufgeregt waren. Deswegen haben wir die Teilnehmer gebeten, die Aussage „Ich war aufgeregt, weil ich an einem Experiment teilnahm und ich konnte deshalb nicht so programmieren, wie ich es gewohnt bin.“ zu bewerten. Als Antworten waren dabei *trifft zu*, *trifft eher zu*, *trifft eher nicht zu* und *trifft nicht zu* vorgegeben. Tabelle 5.7 listet die Verteilung der Bewertungen zu dieser Aussage auf. Wie man dort sehen kann, stimmte die große Mehrheit der Teilnehmer für *trifft nicht zu* oder *trifft eher nicht zu*. Eine Einschränkung der externen Validität durch eine übermäßige Aufregung der Teilnehmer betrachten wir aus diesem Grund als unwahrscheinlich.

Schlussendlich könnte die externe Validität durch mangelnde Motivation der Teilnehmer gefährdet sein. Um die Motivation der Teilnehmer zu prüfen, haben wir sie aufgefordert die Aussage „Die Programmierung im Experiment hat mir gefallen.“ zu bewerten. Es standen dabei die in Absatz zuvor erwähnten vier Antworten zur Auswahl. Wie man in Tabelle 5.8 erkennen kann, stimmte keiner der Teilnehmer für *trifft eher nicht zu* oder *trifft nicht zu*. Daher lässt sich mangelnde Motivation als Bedrohung für die externe Validität mit hoher Wahrscheinlichkeit ausschließen.

5.9 Zusammenfassung und Diskussion der Ergebnisse

In diesem Kapitel haben wir ein Experiment vorgestellt, in dem wir der Frage nachgegangen sind, ob Programmierpaare, in denen die Programmiere unterschiedliche Persönlichkeitsmerkmale haben, anders arbeiten als Paare, in denen die Programmierer gleiche Persönlichkeitsmerkmale haben.

Bei diesem Experiment konnten wir zwei signifikante Unterschiede zwischen den Gruppen nachweisen: Zum einen hatten die Paare der Gleich-Gruppe weniger Akzeptanztestläufe benötigt als die Paare der Ungleich-Gruppe, zum anderen hatten die Paare der Ungleich-Gruppe die Anweisungsabdeckung stärker erhöht als die Paare der Gleich-Gruppe. Wie wir bereits bei der Vorstellung dieser Ergebnisse erwähnt haben, sind wir der Meinung, dass die stärke Erhöhung der Anweisungsabdeckung durch die Paare der Ungleich-Gruppe ein Seiteneffekt ihrer häufigeren Fehlversuche beim Akzeptanztest ist. Warum die Paare der Gleich-Gruppe weniger Versuche für den Akzeptanztest brauchten ist allerdings unklar.

Ansonsten ist festzustellen, dass wir keines der Ergebnisse aus der Studie von Sfetsos und Kollegen [SSAD09], die uns zu diesem Experiment inspiriert hatte, bestätigen konnten. Nach unseren Ergebnissen drängt sich vielmehr die Vermutung auf, dass die Zusammensetzung der Paare nach gleichen und ungleichen Keirsey-Temperamenten zumindest eine deutlich kleinere Rolle für deren Effektivität spielt, als dies die Ergebnisse aus der Studie von Sfetsos und Kollegen suggerieren. Unsere Ergebnisse deuten eher in die Richtung der von Hannay und Kollegen unter Verwendung des Big-Five-Modells als Maß für die Persönlichkeit [HAES10] erzielten Ergebnisse, welche die Autoren dazu veranlassten den Effekt der Persönlichkeit auf die Effektivität eines Paares als eher klein einzuschätzen.

Um das nach wie vor unklare Bild in nächster Zeit ein wenig aufhellen zu können, sollten zukünftige Studien noch einmal einen Schritt zurückgehen und sich zunächst einmal damit befassen, inwieweit Persönlichkeitsmerkmale überhaupt für die Leistungsfähigkeit eines Programmierers entscheidend sind oder ob ihr Einfluss nicht immer weit geringer ist als der von Faktoren wie der Erfahrung und der Ausbildung. Erst wenn diese Frage geklärt ist, lohnt es sich, erneut zu ergründen, welchen Einfluss die Persönlichkeitsmerkmale der Partner auf den Erfolg der Paarprogrammierung haben.

Kapitel 6

Vergleich von testgetriebener und konventioneller Entwicklung

In diesem Kapitel stellen wir eine Studie vor, welche die testgetriebene Entwicklung der konventionellen Entwicklung nach dem Wasserfallmodell bei Einzelprogrammierern gegenüberstellt. Motiviert wurde diese Studie durch die zahlreichen ohne Kontrolle der Prozesstreue durchgeführten Studien aus denen sich bisher kein klares Bild zu den Vor- und Nachteilen der testgetriebenen Entwicklung gegenüber der konventionellen Entwicklung nach dem Wasserfallmodell hervorgeht.

6.1 Verfeinerung der Forschungshypothese FH_3

Die schon in der Einleitung aufgestellte Forschungshypothese zu dieser Studie lautet:

FH_3 Der Einsatz der testgetriebenen Entwicklung führt gegenüber der Verwendung der konventionellen Entwicklung nach dem Wasserfallmodell zu Unterschieden bei der Produktivität der Entwickler und der Qualität der Programme.

Wie schon in den Studien zuvor geschehen, wollen wir auch aus der Forschungshypothese zu dieser Studie zunächst testbare Hypothesen ableiten, indem wir sie auf die in Abschnitt 2.4 zum testgetriebenen Entwicklungsprozess, der Produktivität und der Qualität vorgestellten Metriken übertragen. Zur besseren Nachvollziehbarkeit haben wir die Metriken in Tabelle 6.1 wiederholt aufgelistet. Wie auch schon in den beiden anderen Kapiteln handelt es sich um insgesamt elf Metriken. Aus diesem Grund werden wir auch hier nicht jede Alternativhypothese einzeln aufführen, sondern formulieren die folgende Schablone für die Alternativhypothesen:

$H_A^{[Abk.\ Metrik]}$ Der Einsatz der testgetriebenen Entwicklung führt gegenüber der Verwendung der konventionellen Entwicklung nach dem Wasserfallmodell zu Unterschieden bei *[der/dem/den] [Metrik]*.

Wiederum erhält man durch Einsetzen der Metriken die Alternativhypothesen, wie z. B.:

$\mathbf{H_A^{TGE}}$ Der Einsatz der testgetriebenen Entwicklung führt gegenüber der Verwendung der konventionellen Entwicklung nach dem Wasserfallmodell zu Unterschieden bei der TGE-Prozesstreue.

In der letzten Spalte von Tabelle 6.1 finden sich die Kurzschreibweisen aller Alternativhypothesen. Diese stehen dabei stets unter den Kurzschreibweisen der entsprechenden Nullhypothesen, welche sich durch logische Negation der Alternativhypothesen ergeben. Vor jeder der Kurzschreibweisen findet sich ein innerhalb dieses Kapitels eindeutiger Schlüssel über den wir die entsprechende Hypothese referenzieren werden. Innerhalb der Kurzschreibweisen ist der Name der Metrik durch einen korrespondierende Abkürzung ersetzt. Diese Abkürzung findet sich auch als hochgestellter Text im Schlüssel der Hypothese wieder. Die Indizes in der Kurschreibweise stehen für die in der Studie verglichenen Entwicklungsprozesse. Der Index KON steht dabei für die konventionelle Entwicklung nach dem Wasserfallmodell, der Index TGE für die testgetriebene Entwicklung.

6.2 Versuchsaufbau

Um die Hypothesen überprüfen zu können, wurde ein gegenbalanciertes Experiment entworfen (vgl. [SCC02, S. 268]). Bei einem gegenbalancierten Entwurf liefert jeder Teilnehmer zu jeder Ausprägung der unabhängigen Variable einen Datenpunkt. In diesem Experiment ist die unabhängige Variable der Entwicklungsprozess, die unabhängigen Variablen sind die Metriken aus Tabelle 6.1. Die unabhängige Variable hat in unserem Fall zwei Ausprägungen: Die testgetriebene und die konventionelle Entwicklung. Die eigentliche Gegenbalancierung bedeutet in unserem Fall, dass die Gruppe der Teilnehmer geteilt wird und die eine Hälfte zuerst testgetrieben entwickelt und die andere Hälfte zuerst konventionell. Auf diese Weise wird die Verfälschung der Ergebnisse durch Reihenfolgeeffekte beim gewählten Entwicklungsprozess vermieden.

Da bei unserem Experiment eine Wiederholung der gleichen Aufgaben wegen des Lerneffekts nicht möglich ist, benötigen wir zwei Aufgaben. Auf der Ebene der Aufgaben muss dann wiederum gegenbalanciert werden, um auch hier Reihenfolgeeffekte zu vermeiden. Daher muss jede der bisher zwei Gruppen erneut zweigeteilt werden, damit jede Kombination aus Aufgabe und Entwicklungsprozess an jeder Position im Versuchsaufbau vorkommt.

Tabelle 6.2 zeigt die so entstandene Gruppeneinteilung. TGE steht dabei für die testgetriebene Entwicklung, KON für die konventionelle Entwicklung nach dem Wasserfallmodell. Die Kürzel VVS und S532 kennzeichnen die in den Abschnitten 6.5.1 und 6.5.2

Metrik	*Null- & Alternativhypothese*	
	Schl.	*Kurzschreibweise*
Kategorie: TGE-Prozess		
TGE-Prozesstreue [%]	H_0^{TGE}	$TGE_{KON} = TGE_{TGE}$
	H_A^{TGE}	$TGE_{KON} \neq TGE_{TGE}$
Anzahl der automatischen Umstrukturierungen	H_0^{AU}	$AU_{KON} = AU_{TGE}$
	H_A^{AU}	$AU_{KON} \neq AU_{TGE}$
Mittlerer Abstand der Testläufe [min]	H_0^{MAT}	$MAT_{KON} = MAT_{TGE}$
	H_A^{MAT}	$MAT_{KON} \neq MAT_{TGE}$
Variationskoeffizient des Abstands der Testläufe [%]	H_0^{VAT}	$VAT_{KON} = VAT_{TGE}$
	H_A^{VAT}	$VAT_{KON} \neq VAT_{TGE}$
Kategorie: Produktivität		
Bearbeitungszeit [min]	H_0^{BZ}	$BZ_{KON} = BZ_{TGE}$
	H_A^{BZ}	$BZ_{KON} \neq BZ_{TGE}$
Netto-Änderungen [SLOC]	$H_0^{NÄ}$	$NÄ_{KON} = NÄ_{TGE}$
	$H_A^{NÄ}$	$NÄ_{KON} \neq NÄ_{TGE}$
Anteil der Netto-Änderungen im Testcode [%]	$H_0^{NÄ}$	$NÄ_{KON} = NÄ_{TGE}$
	$H_A^{NÄ}$	$NÄ_{KON} \neq NÄ_{TGE}$
Anzahl der geänderten Dateien	H_0^{GD}	$GD_{KON} = GD_{TGE}$
	H_A^{GD}	$GD_{KON} \neq GD_{TGE}$
Kategorie: Qualität		
Änderung der Anzahl der Testfälle	$H_0^{\Delta TF}$	$\Delta TF_{KON} = \Delta TF_{TGE}$
	$H_A^{\Delta TF}$	$\Delta TF_{KON} \neq \Delta TF_{TGE}$
Änderung der Anweisungs-abdeckung [%]	$H_0^{\Delta AÜ}$	$\Delta AÜ_{KON} = \Delta AÜ_{TGE}$
	$H_A^{\Delta AÜ}$	$\Delta AÜ_{KON} \neq \Delta AÜ_{TGE}$
Anzahl der Akzeptanz-testläufe	H_0^{AT}	$AT_{KON} = AT_{TGE}$
	H_A^{AT}	$AT_{KON} \neq AT_{TGE}$

Tabelle 6.1: Übersicht über die Hypothesen.

Gruppe	*Tag 1*		*Tag 2*	
	Prozess	*Aufgabe*	*Prozess*	*Aufgabe*
KV/TS	KON	VVS	TGE	S532
KS/TV	KON	S532	TGE	VVS
TV/KS	TGE	VVS	KON	S532
TS/KV	TGE	S532	KON	VVS

Tabelle 6.2: Gegenbalancierter Entwurf

Gruppe	Teilnehmer	
	initial	final
KV/TS	4	3
KS/TV	3	3
TV/KS	3	3
TS/KV	3	3

Tabelle 6.3: Verteilung der Teilnehmer auf die Gruppen.

besprochenen Aufgaben Videoverleihsystem und Soundex. Die Gruppennamen ergeben sich aus den Anfangsbuchstaben von Prozess und Aufgabe der beiden Durchgänge.

6.3 Versuchsdurchführung

Das Experiment wurde im Anschluss an das Extreme Programming-Praktikum 2010 an zwei aufeinanderfolgenden Tagen durchgeführt. Der erste Tag begann morgens mit der Gruppeneinteilung. Hierfür wurden die insgesamt 13 Teilnehmer zunächst von den drei Betreuern des Extreme Programming-Praktikums nach ihrer Programmierfähigkeit beurteilt und in drei Blöcke eingeteilt: Sehr gute, gute und mäßige Programmierer. In den ersten zwei Blöcken befanden sich jeweils vier im dritten fünf der Teilnehmer[1].

Im Anschluss fand die zufällige Verteilung der Teilnehmer auf die Gruppen statt, die das in Tabelle 6.3 dargestellte Ergebnis erbrachte. Für die zufällige Verteilung wurde für jede der vier Gruppen genau ein Los in ein Gefäß gegeben aus dem zunächst die sehr guten Programmierer ziehen durften. Im Anschluss wurden die Lose wieder in das Gefäß gegeben und die guten Programmierer durften ziehen. Für die mäßigen Programmierer

[1] Das Bilden von Blöcken, mit dem Ziel, eine möglichst gleichmäßige Verteilung einer Störvariablen auf die Gruppen zu erreichen, bezeichnet Christensen als *Matching* [Chr07, S. 270].

folgten zwei Losrunden nach dem gleichen Prinzip, wobei zuvor per Streichholzziehen ermittelt wurde, welcher der fünf mäßigen Programmierer in der letzten Runde alleine ein Los zu ziehen hatte.

Mit diesem Losverfahren konnten wir sicherstellen, dass die Gruppen bezogen auf die Programmierfähigkeiten der Teilnehmer recht ausgeglichen waren. Ein grobes Ungleichgewicht der Programmierfähigkeiten zwischen den Gruppen hätte ansonsten die Gegenbalancierung gefährdet. Theoretisch hätte man sich auch vorstellen können, statt der Einschätzung der Betreuer des Praktikums auch Daten wie die vorherige Programmiererfahrung in Jahren heranzuziehen. Wir sind jedoch der Meinung, dass die Einschätzung der Betreuer, welche die Teilnehmer in der Praktikumswoche intensiv bei der Arbeit beobachtet und unterstützt hatten, eine bessere Vorhersage für die tatsächliche Leistungsfähigkeit der Teilnehmer liefert.

Nach der Verteilung der Teilnehmer auf die Gruppen wurden ihnen Arbeitsplätze im Computerpoolraum des Instituts zugewiesen. Da dieser allerdings nur zwölf Arbeitsplätze hat, erhielt ein Teilnehmer einen Arbeitsplatz im wenige Meter entfernten Büro des Autors.

Nachdem alle Teilnehmer ihren Arbeitsplatz eingenommen hatten, wurden ihnen die Vertraulichkeitserklärungen ausgehändigt. Mit diesem Dokument wurde den Teilnehmern garantiert, dass ihre persönlichen Daten streng vertraulich behandelt und nur anonymisiert veröffentlicht werden. Zudem wurden die Teilnehmer informiert, dass sie zu jedem Zeitpunkt ihre Teilnahme beenden und eine Löschung der über sie gesammelten Daten verlangen können. Im Gegenzug verpflichteten sich die Teilnehmer mit ihrer Unterschrift, Stillschweigen über den Inhalt und Ziel der Studie zu wahren. Weiterhin enthielt das Dokument eine Angabe über die geschätzte Arbeitszeit von jeweils drei bis vier Stunden an zwei Tagen. Wie wir später in Abschnitt 6.6.2.1 sehen werden, wurde diese Arbeitszeit von einigen Teilnehmern deutlich überschritten. Wir werden in Abschnitt 6.8.4 diskutieren, ob dies möglicherweise negative Auswirkungen auf die Motivation der Teilnehmer hatte.

Im nächsten Schritt wurden die unterschriebenen Vertraulichkeitserklärungen eingesammelt und die Teilnehmerfragebögen ausgehändigt. Die Teilnehmerfragebögen dienten zur Erfassung der Vorkenntnisse der Teilnehmer. Die Daten dieser Fragebögen flossen in die Beschreibung der Teilnehmer in Abschnitt 6.4 ein.

Im Anschluss an das Ausfüllen der Teilnehmerfragebögen erhielt jeder der Teilnehmer die Aufgabenbeschreibung. Diese Aufgabenbeschreibung umfasste neben der eigentlichen Aufgabenstellung eine Zeiterfassungstabelle und Anweisungen, wie Arbeitszeiten und Pausen dort einzutragen sind. Zudem war in der Aufgabenbeschreibung eine Kurzbeschreibung des vom Teilnehmer erwarteten Entwicklungsprozess enthalten. Weiterhin wurden sie in diesem Dokument auf den Akzeptanztest hingewiesen und dass dieser

möglichst beim ersten Versuch bestanden werden sollte. Während die Teilnehmer mit dem Einlesen beschäftigt waren, wurde ihnen vom Experimentleiter ihre Entwicklungsumgebung für die Bearbeitung der Aufgabe eingerichtet, so dass sie nach dem Einlesen direkt zur Bearbeitung der Aufgabe übergehen konnten.

Sobald ein Teilnehmer der Überzeugung war, dass sein Programm alle Anforderungen aus der Aufgabenbeschreibung erfüllt, konnte er sich beim Experimentleiter zum Akzeptanztest melden. Bestand sein Programm den Akzeptanztest, erhielt er von Experimentleiter einen Fragebogen zur Aufgabe, in dem der Teilnehmer die Aufgabe aus seiner Sicht bewerten und von möglichen Problemen beim Lösen berichten konnte. Nach dem Ausfüllen dieses Fragebogens war der erste Tag des Experiments für den Teilnehmer beendet. Bestand sein Programm den Akzeptanztest dagegen nicht, wurden dem Teilnehmer die Testergebnisse übergeben und um eine Nachbesserung gebeten. Wenn der Teilnehmer sich sicher war, alle Fehler eliminiert zu haben, durfte er sein Programm erneut zum Akzeptanztest einreichen. Am ersten Tag des Experiments gelang es einem der Teilnehmer auch nach mehr als acht Stunden Arbeitszeit und fünf Anläufen nicht, den Akzeptanztest zu bestehen. Dieser Teilnehmer gab schließlich auf. Sein Datenpunkt wurde aus dem Datensatz entfernt. Daher beziehen sich alle Angaben in allen folgenden Abschnitten, soweit nicht anders erwähnt, auf die Datenpunkte der zwölf Teilnehmer, welche beide Aufgaben gelöst haben.

Am Morgen des zweiten Tags des Experiments erhielten die Teilnehmer, nachdem sie ihre Arbeitsplätze vom Vortag eingenommen hatten, die Aufgabenbeschreibung zur zweiten Aufgabe. Wiederum wurde ihnen die Entwicklungsumgebung von Experimentleiter eingerichtet, während sie mit dem Lesen der Aufgabenstellung beschäftigt waren, so dass sie auch am zweiten Tag direkt nach dem Einlesen zum Lösen der Aufgabe übergehen konnten. Nach dem Bestehen des Akzeptanztests erhielten sie, wie schon am Vortag, den Fragebogen zur Aufgabe und anschließend einen Fragebogen, indem sie gebeten wurden, ihre Eindrücke zu den von ihnen verwendeten Entwicklungsprozessen und den beiden bearbeiteten Aufgaben miteinander zu vergleichen. Nach dem Ausfüllen dieses Fragebogens war das Experiment für die Teilnehmer beendet.

6.4 Beschreibung der Teilnehmer

Anders als in den beiden vorhergehenden Studien lieferte in dieser Studie jeder Teilnehmer einen Datenpunkt zum TGE- und zum KON-Prozess ab. Daher macht sich die Erfahrung der Teilnehmer bei beiden Prozessen gleichermaßen bemerkbar. Deswegen sehen wir hier davon ab, die Vorkenntnisse nach Gruppen getrennt zu präsentieren und besprechen sie für alle zwölf Teilnehmer gemeinsam.

Metrik	*Codebereich*		
	Anwendung	*Test*	*Gesamt*
Anzahl der Klassen & Schnittstellen	46	38	84
Anzahl der Methoden	193	265	458
Länge [SLOC]	2316	1675	3991
Anweisungsabdeckung [%]	86,3	–	–
Anzahl der Testfälle	–	179	–

Tabelle 6.4: Kennzahlen der Aufgabe Videoverleihsystem

Bei den Teilnehmern an unserem Experiment handelte es sich ausschließlich um Studenten der Informatik. Vor ihrer Teilnahme am Extreme Programming-Praktikum hatten sie im Durchschnitt 4,3 Jahre studiert. Im privaten Umfeld und für den eigenen Gebrauch hatten sie durchschnittlich 6,4, in einer nebenberuflichen Tätigkeit 1,9 Jahre programmiert. Dabei hatten drei von zwölf Teilnehmern noch nie im Rahmen einer (neben-) beruflichen Tätigkeit programmiert. Mit der im Experiment verwendeten Programmiersprache Java konnten die Teilnehmer vor dem Experiment im Mittel 4,4 Jahre, mit Testautomatisierung 1,1 Jahre und mit JUnit 1,0 Jahre Erfahrung sammeln. Allerdings hatten fünf von zwölf Teilnehmern zuvor noch nie mit Testautomatisierung gearbeitet. Mit JUnit fehlte sogar der Hälfte der Teilnehmer jegliche Erfahrung. Am geringsten war jedoch die vorhergehende Programmiererfahrung mit der testgetriebenen Entwicklung: Nur drei der Teilnehmer hatten diese Programmiermethode vor dem Experiment bereits angewendet.

6.5 Aufgaben

Da jeder Teilnehmer des Experiments aufgrund des gegenbalancierten Entwurfs für jeden der zwei Prozesse jeweils einen Datenpunkt liefern soll, werden auch zwei Programmieraufgaben benötigt. Diese werden in den folgenden Abschnitten vorgestellt.

6.5.1 Aufgabe Videoverleihsystem

Die Aufgabe Videoverleihsystem, kurz VVS, wurde von Schneider [Sch10] im Rahmen seiner beim Autor dieser Arbeit durchgeführten Diplomarbeit entwickelt und für dieses Experiment mit kleinen Änderungen übernommen. Bei dieser Aufgabe muss die Verwaltungssoftware einer 24-Stunden-Videothek auf ein neues Preisschema, das die unterschiedlichen Preise von DVDs und BluRay-Disks berücksichtigt, umgestellt werden.

Metrik	*Codebereich*		
	Anwendung	*Test*	*Gesamt*
Anzahl der Klassen & Schnittstellen	19	17	36
Anzahl der Methoden	61	80	141
Länge [SLOC]	483	688	1171
Anweisungsabdeckung [%]	63,5	–	–
Anzahl der Testfälle	–	41	–

Tabelle 6.5: Kennzahlen der Aufgabe Soundex

Zudem soll die Verwaltungssoftware um die Erstellung von monatlichen Rechnungen für die Kunden der Videothek erweitert werden. Die Verwaltungssoftware verfügt über eine Kommandozeile, welche sämtliche Benutzerinteraktionen mit einfachen Befehlen im Stil von Unix-Kommandos abgewickelt. Dabei können sich unter anderem Kunden anmelden, Filme ausleihen beziehungsweise abgeben oder ihren aktuellen Kontostand abrufen.

Zur Datenhaltung verwendet die Verwaltungssoftware eine hauptspeicherbasierte Datenbank [HSQ], d. h. Änderungen in der Datenbank bleiben nur bis zum Beenden des Programms erhalten. Auf diese Weise ist sichergestellt, dass die Teilnehmer des Experiments den Datenbestand oder das Schema nicht beschädigen können, selbst wenn sie beim Testen Fehler machen. Das Verfassen von SQL-Abfragen ist zum Lösen der Aufgabe nicht nötig, da alle notwendigen Abfragen bereits in einer Datenbank-Klasse vorimplementiert sind.

Wie in Tabelle 6.4 zusammenfassend dargestellt, umfasst der Anwendungscode der Verwaltungssoftware in dem Zustand, indem sie an die Teilnehmer ausgehändigt wurde, 46 Klassen und Schnittstellen mit 193 Methoden und 2316 Zeilen nach dem in Abschnitt 2.4.1.2 beschriebenen Verfahren bereinigten Code. Der dazugehörige Testcode besteht aus 38 Klassen und Schnittstellen, hat 265 Methoden und 1675 Zeilen bereinigtem Code und definiert 179 Testfälle, welche 86,3 Prozent der Anweisungen des Anwendungscodes abdecken.

Mit der Verwaltungssoftware bekamen die Teilnehmer eine ausführliche Aufgabenbeschreibung ausgehändigt. Neben der eigentlichen Problemstellung sind dort die Paketstruktur des ausgelieferten Codes, die Befehle für die Kommandozeile sowie das Datenbankschema und die in der Datenbank vorhandenen Datensätze erläutert.

6.5.2 Aufgabe Soundex

Die Aufgabe Soundex verlangt die Implementierung des Soundex-Algorithmus. Dieser Algorithmus wurde 1918 von Robert C. Russell für die phonetische Suche von Familiennamen in den USA entwickelt. Der Algorithmus bildet Wörter auf stets vierstellige Soundex-Codes ab. Dabei werden ähnlich klingende Wörter auf identische Codes abgebildet. So ist das für diese Aufgabe verwendete Kürzel S532 der Soundex-Code des Wortes *Soundex*.

Der Soundex-Algorithmus muss bei dieser Aufgabe in ein existierendes Programm eingebettet werden, welches eine Kommandozeile zur Interaktion mit dem Benutzer zur Verfügung stellt. Dieses Programm kann über in der Kommandozeile eingegebene Befehle Textdateien einlesen. Um die Aufgabe zu lösen, muss ein weiterer, teilweise implementierter Befehl so ergänzt werden, dass in einer zuvor eingelesen Textdatei nach Wörtern gesucht wird, die den identischen Soundex-Code haben wie ein mit diesem Befehl übergebener Suchbegriff.

In der Form in der der Anwendungscode des Programms an die Teilnehmer ausgehändigt wurde hat er 19 Klassen und Schnittstellen mit 61 Methoden und 483 Zeilen bereinigtem Code. Der Testcode setzt sich aus 17 Klassen und Schnittstellen mit 80 Methoden und 688 Zeilen bereinigtem Code zusammen. Die 41 Testfälle decken 63,5 Prozent der Anweisungen des Anwendungscodes ab. Tabelle 6.5 fasst diese Werte noch einmal zusammen.

Zusätzlich zum Programm erhielten die Teilnehmer eine genaue Aufgabenbeschreibung. In dieser sind die Umsetzungsvorschrift von Wörtern zu Soundex-Codes, die Paketstruktur des ausgelieferten Codes und die Befehle für die Kommandozeile erklärt.

6.6 Ergebnisse

In diesem Abschnitt stellen wir die Ergebnisse der Auswertung des Datensatzes vor. Die in Abschnitt 6.1 aufgelisteten haben wir mit einem zweiseitigen Wilcoxon-Test bei einem Signifikanzniveau von 5 Prozent getestet. Die dabei aufgetretenen p-Werte werden wir bei der Besprechung der Ergebnisse wie bei den anderen Studien auch in jedem Fall nennen. Lage- und Streuungsmaße zu den erhobenen Metriken werden wir aber nur angeben, wenn sie unserer Meinung nach erwähnenswert sind. Vollständige Übersichten über Lage- und Streuungsmaße finden sich aber, zusammen mit einer Übersicht über alle p-Werte, in Anhang C.

6.6.1 TGE-Prozess

Im Gegensatz zu den anderen Studien kommt den Metriken des testgetriebenen Entwicklungsprozesses diesmal eine Sonderrolle zu. Hier ist der von den Teilnehmern durchgeführte Entwicklungsprozess die unabhängige Variable, die wir manipuliert haben. In Folge dessen sind Unterschiede bei den Metriken, die wir dem TGE-Prozess zuschreiben, mindestens erwartet wenn nicht gar Voraussetzung für die Auswertung der weiteren Metriken.

6.6.1.1 TGE-Prozesstreue

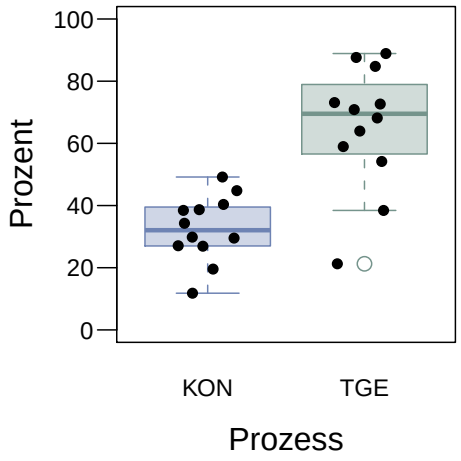


Abbildung 6.1: TGE-Prozesstreue.

Wir betrachten zunächst die TGE-Prozesstreue in Abhängigkeit des verwendeten Entwicklungsprozesses in Abbildung 6.1. Es zeigt sich, dass sich die Teilnehmer im Wesentlichen an die Anweisungen gehalten haben und sich folglich die TGE-Prozesstreue bei den Prozessen deutlich unterscheidet. Dementsprechend lässt sich auch die Nullhypothese H_0^{TGE} mit einem p-Wert kleiner 0,001 ablehnen. Die Prozesse unterscheiden sich bei der TGE-Prozesstreue signifikant.

Wenden wir uns für eine genauere Analyse der TGE-Prozesstreue Abbildung 6.2 zu. Die Datenpunkte sind in dieser Abbildung zunächst nach dem Prozess und im Anschluss nach ihrer TGE-Prozesstreue geordnet. Die TGE-Prozesstreue lässt sich in dieser Darstellung als Summe des blauen und des grünen Balkens ablesen. Beginnen wir unsere Betrachtung beim TGE-Prozess. Hier fallen zunächst zwei Datenpunkte am rechten Rand mit besonders niedrigen Werten der TGE-Prozesstreue von 38,4 bzw. 21,3 Prozent

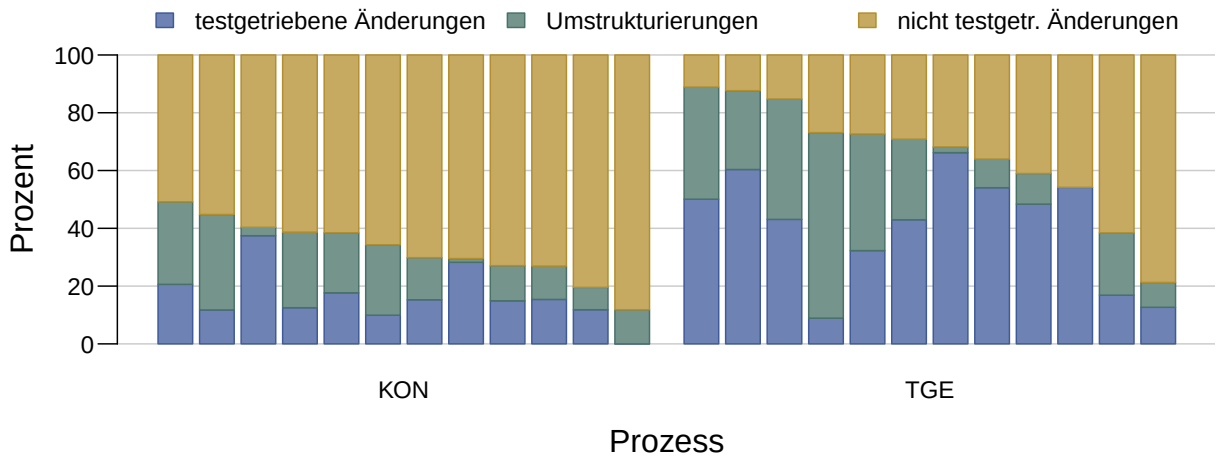


Abbildung 6.2: Anteile der testgetriebenen Änderungen, Umstrukturierungen und nicht testgetriebenen Änderungen an allen Änderungen.

auf. Wie es scheint, ist die Befürchtung, dass Anfänger möglicherweise Probleme mit der Umsetzung der testgetriebenen Entwicklung haben, zumindest in Einzelfällen nicht unbegründet.

Bevor wir uns den KON-Prozess in dieser Abbildung ansehen, müssen wir uns zunächst darüber klar werden, dass die TGE-Prozesstreue keine Metrik ist, welche die Prozesstreue zur konventionellen Entwicklung misst. Ein niedriger Wert der TGE-Prozesstreue ist nicht mit einer hohen Prozesstreue gegenüber der konventionellen Entwicklung gleichzusetzen, auch wenn eine gewisse Korrelation zu vermuten ist. Weiter gilt, dass ein hoher Wert in der TGE-Prozesstreue beim KON-Prozess nicht zwangsläufig problematisch sein muss, da Umstrukturierungen sowohl im testgetriebenen als auch im konventionellen Entwicklungsprozess prozesskonform sind.

Damit sind nicht die beiden linken Datenpunkte mit der höchsten TGE-Prozesstreue innerhalb beim KON-Prozess die auffälligsten Datenpunkte, sondern der dritte und der achte Datenpunkt von links, die mit 37,5 und 28,2 Prozent die höchsten Anteile an testgetriebenen Änderungen aufweisen. Der dritte Datenpunkt von links stammt vom gleichen Teilnehmer, der schon beim TGE-Prozess mit der niedrigste TGE-Prozesstreue aufgefallen war.

Nun stellt sich die Frage, was wir mit Datenpunkten machen, die darauf hindeuten, dass sich die dazugehörigen Teilnehmer im Experiment nicht an den vorgeschriebenen Entwicklungsprozess gehalten haben. Grundsätzlich haben wir zwei Optionen: Wir können die Datenpunkte entweder entfernen oder im Datensatz behalten. Für das Entfernen spricht, dass die Teilnehmer nicht fähig waren, sich an den vorgeschriebenen Entwicklungsprozess zu halten. Dagegen spricht, dass es auch in der Population immer Personen geben wird, die einen Entwicklungsprozess nicht richtig einhalten können und wir die

Ergebnisse unseres Experiments letztendlich auf die Population übertragen wollen. Daher entscheiden wir uns dafür, die fraglichen Datenpunkte im Datensatz zu belassen.

6.6.1.2 Anzahl der automatischen Umstrukturierungen

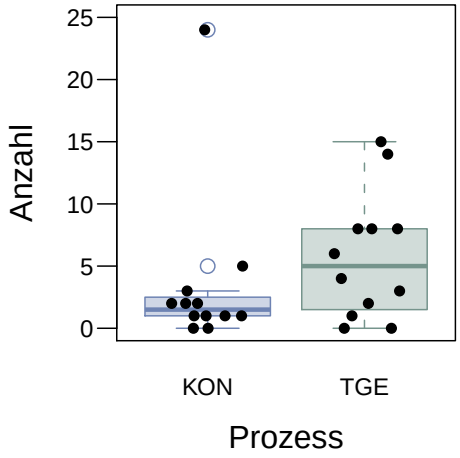


Abbildung 6.3: Anzahl der automatischen Umstrukturierungen.

Richten wir unseren Blick als nächstes auf die in Abbildung 6.3 dargestellte Anzahl der automatischen Umstrukturierungen. Wie wir dort sehen können, kam es im Großen und Ganzen bei Anwendung des TGE-Prozesses zu mehr automatischen Umstrukturierungen als bei Anwendung des KON-Prozesses. Dies zeigt sich auch bei der Lage der Mediane: Für den TGE-Prozess liegt er mit 5 automatischen Umstrukturierungen höher als für den KON-Prozess mit 1,5 automatischen Umstrukturierungen. Dieses Bild entspricht im Wesentlichen den Erwartungen, da häufiges Umstrukturieren ein zentraler Bestandteil des TGE-Prozesses ist, was sich, die entsprechenden Eclipse-Kenntnisse der Teilnehmer vorausgesetzt, auch bei den automatischen Umstrukturierungen wiederspiegeln sollte. Wäre da nicht der extreme Ausreißer beim KON-Prozess, dann wäre der optisch erkennbare Unterschied signifikant. Mit dem Ausreißer liefert der Wilcoxon-Tests jedoch einen p-Wert von 0,122, was bedeutet, dass der Unterschied nicht signifikant ist und wir folglich annehmen müssen, dass die Nullhypothese H_0^{AU} zutrifft.

6.6.1.3 Testläufe

Sehen wir uns als nächstes an, wie sich die unterschiedlichen Entwicklungsprozesse bei den Testläufen bemerkbar machen. Werfen wir hierzu einen Blick auf den mittleren

Abstand der Testläufe in Abbildung 6.4. Tendenziell scheint beim KON-Prozess der Abstand der Testläufe größer als beim TGE-Prozess zu sein. Die Lage der Mediane bei 4,5 Minuten beim KON- gegenüber nur 2,7 Minuten beim TGE-Prozess bestätigt diesen Eindruck. Allerdings ist der Unterschied mit einem p-Wert von 0,16 nicht signifikant, so dass die Nullhypothese H_0^{MAT} nicht abgelehnt werden kann und wir davon ausgehen müssen, dass die Beobachtung Zufall ist.

Da die regelmäßige Testausführung ein wesentlicher Bestandteil des testgetriebenen Entwicklungsprozesses ist, hatten wir erwartet, dass beim TGE-Prozess deutlich häufiger Testläufe stattfinden. Dass dies hier nicht nachgewiesen werden konnte, mag aber zumindest teilweise darin begründet sein, dass die Teilnehmer auch bei Einsatz des konventionellen Entwicklungsprozesses verpflichtet waren, ihre Programme ausführlich zu testen. Möglich ist es aber auch, dass der Unterschied beim mittleren Abstand der Testläufe zu klein ist, als dass wir ihn hier nachweisen können.

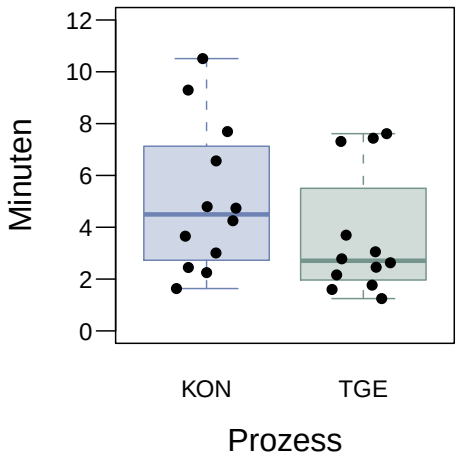


Abbildung 6.4: Mittlerer Abstand der Testläufe.

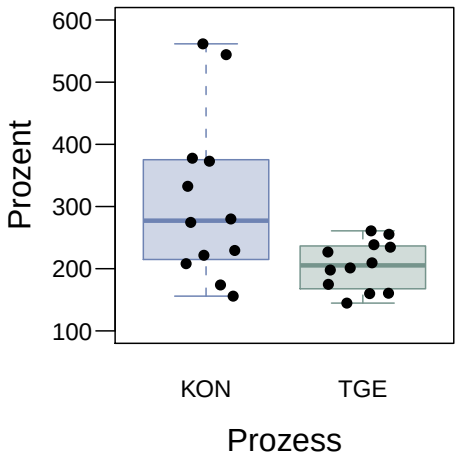


Abbildung 6.5: Variationskoeffizient des Abstands der Testläufe.

Wenn wir uns jedoch dem Variationskoeffizienten des Abstands der Testläufen in Abbildung 6.5 zuwenden, dann sehen wir, dass die Testläufe bei Einsatz des KON-Prozesses zwar nicht seltener aber dafür weitaus unregelmäßiger als bei Einsatz des TGE-Prozesses ausgeführt wurden. Hier liefert der Wilcoxon-Test einen p-Wert von 0,033 und wir können die Nullhypothese H_0^{VAT} zu Gunsten der Alternativhypothese H_A^{VAT} verwerfen[2]. Dies entspricht unseren Erwartungen, dass bei der testgetriebenen Entwicklung die

[2]Ebenso ergibt sich ein signifikanter Unterschied bei der Standardabweichung der Testläufe, die wir als Metrik für den TGE-Prozess jedoch ausgeklammert hatten. Die Ergebnisse hierzu können dennoch im Anhang C nachgelesen werden.

Testläufe regelmäßig stattfinden, was zu einem geringen Variationskoeffizienten führt, während sich bei der konventionellen Entwicklung Testphasen mit vielen Testläufen mit längeren Pausen abwechseln, was zu einem hohen Variationskoeffizienten führt.

6.6.2 Produktivität

Obgleich die Befürworter der testgetriebenen Entwicklung argumentieren, dass sie gegenüber der konventionellen Entwicklung zu einem Produktivitätszuwachs führt, konnte so ein Vorteil bisher erst ein Mal nachgewiesen werden [EMT05]. Bei weitaus mehr Studien zeigte sich ein Einbruch bei der Produktivität. Daher wird es interessant sein zu sehen, was die Auswertung unseres Datensatzes diesbezüglich ergeben hat.

6.6.2.1 Bearbeitungszeit

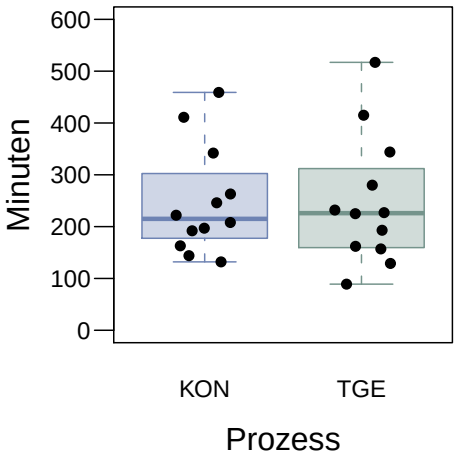


Abbildung 6.6: Bearbeitungszeit.

Sehen wir uns zuerst die Bearbeitungszeit in Abbildung 6.6 an. Hier ist praktisch kein Unterschied zwischen den Boxplots der beiden Prozessen erkennbar. Die Boxen überlappen sich fast vollständig und die Mediane liegen ebenfalls fast gleichauf. Dies spiegelt sich auch im Ergebnis des entsprechenden Hypothesentests mit einem p-Wert von 0,977 wieder. Die Nullhypothese H_0^{BZ} lässt sich nicht ablehnen und wir gehen davon aus, dass sich die Prozesse bezüglich der Bearbeitungszeit nicht unterscheiden.

6.6.2.2 Änderungen am Code

Kommen wir als nächstes zu den von den Teilnehmern durchgeführten Änderungen am Code. Die Netto-Änderungen an den Programmen sind in Abbildung 6.7 dargestellt. Hier sieht es so aus, als würde die testgetriebene gegenüber der konventionellen Entwicklung zu mehr Änderungen am Code führen. Der in der Abbildung sichtbare Unterschied ist allerdings nicht signifikant ($p = 0,178$) und wir können die Nullhypothese $H_0^{NÄ}$ nicht ablehnen.

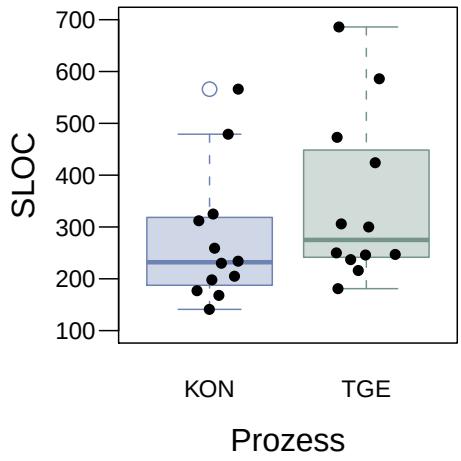


Abbildung 6.7: Netto-Änderungen.

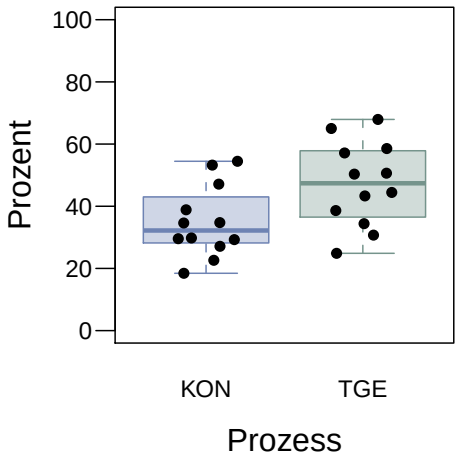


Abbildung 6.8: Anteil der Netto-Änderungen im Testcode.

Betrachten wir nun den Anteil der Netto-Änderungen im Testcode in Abbildung 6.8. Man sieht dort, dass der Anteil an Netto-Änderungen im Testcode beim KON-Prozess im Großen und Ganzen niedriger ausfällt als beim TGE-Prozess. So liegt auch der Median des KON-Prozesses bei 32,2 Prozent während er beim TGE-Prozess bei 47,4 Prozent liegt. Wie das Ergebnis des Wilcoxon-Tests mit einem p-Wert von 0,039 zeigt, ist dieser Unterschied sogar signifikant. Damit können wir die Nullhypothese H_0^{TC} ablehnen und die dazugehörige Alternativhypothese H_A^{TC} annehmen.

Zum Abschluss dieses Abschnitts wenden wir uns der Anzahl der geänderten Dateien zu. Diese ist in Abbildung 6.9 dargestellt. Ein klarer Trend, in bei welchem Entwicklungsprozess mehr Dateien geändert wurden, lässt sich aus der Abbildung nicht ablesen. Deswegen ist es nicht überraschend, dass es in diesem Punkt keinen signifikanten Unterschied gibt ($p = 0,577$) und sich die Nullhypothese H_0^{GD} folglich nicht ablehnen lässt.

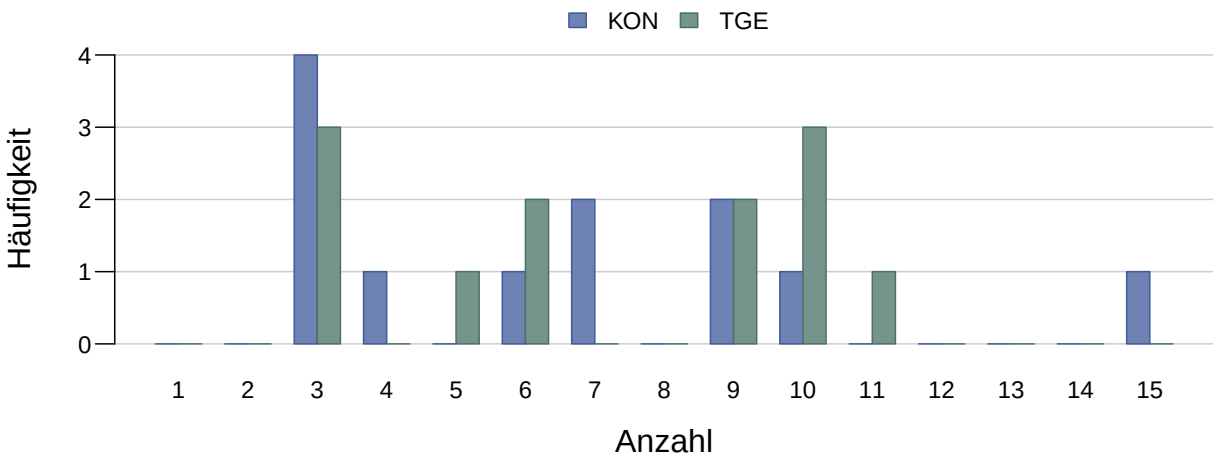


Abbildung 6.9: Anzahl der geänderten Dateien.

6.6.3 Qualität

Die Befürworter der testgetriebenen Entwicklung führen als einen der Vorteile dieser Entwicklungsmethode die regelmäßige Ausführung und Pflege der Testfälle an. Dadurch soll auch die Qualität des Anwendungscodes besser werden, da der Code stetig überprüft wird. Deswegen würden wir erwarten, dass beim TGE-Prozess die Testfälle besser gepflegt wurden.

6.6.3.1 Änderungen an den Testfällen

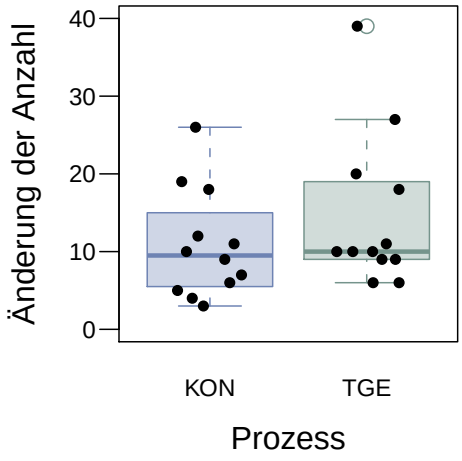


Abbildung 6.10: Änderung der Anzahl der Testfälle

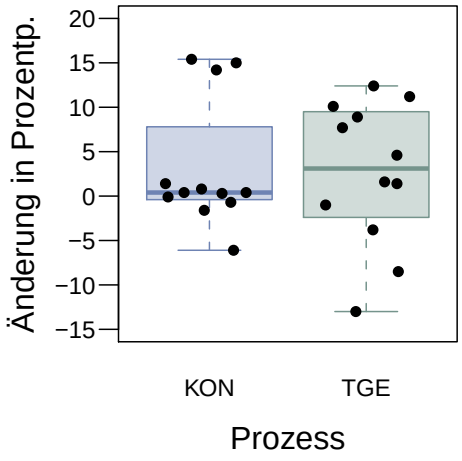


Abbildung 6.11: Änderung der Anweisungsabdeckung des Anwendungscodes.

Wenn man die Änderung der Anzahl der Testfälle in Abbildung 6.10 betrachtet, sieht man fast keinen Unterschied zwischen den beiden Gruppen. Wie der *p*-Wert des entsprechenden Wilcoxon-Tests von 0,324 zeigt, gibt es auch keinen signifikanten Unterschied. Die Nullhypothese $H_0^{\Delta TA}$ kann nicht abgelehnt werden. Es scheint, als würden bei der konventionellen Entwicklung in den Testphasen ungefähr gleich viele Testfälle erstellt, wie während des gesamten testgetriebenen Entwicklungsprozesses.

Sehen wir uns nun die in Abbildung 6.11 gezeigte Änderung der Anweisungsabdeckung des Anwendungscodes an. Hier fällt zunächst auf, dass acht der zwölf Datenpunkte beim KON-Prozess sehr nah am Nullpunkt liegen. Beim TGE-Prozess streuen die Datenpunkte weiter. Zudem fällt der Zuwachs an Anweisungsabdeckung beim TGE-Prozess mit 3,4 Prozentpunkten gegenüber 0,4 Prozentpunkten beim KON-Prozess leicht höher aus. Ein Vorteil des TGE-Prozesses lässt sich jedoch durch den Hypothesentest nicht nachweisen. Mit einem *p*-Wert von 0,908 können wir die Nullhypothese $H_0^{\Delta AÜ}$ nicht verwerfen.

6.6.3.2 Anzahl der Akzeptanztestläufe

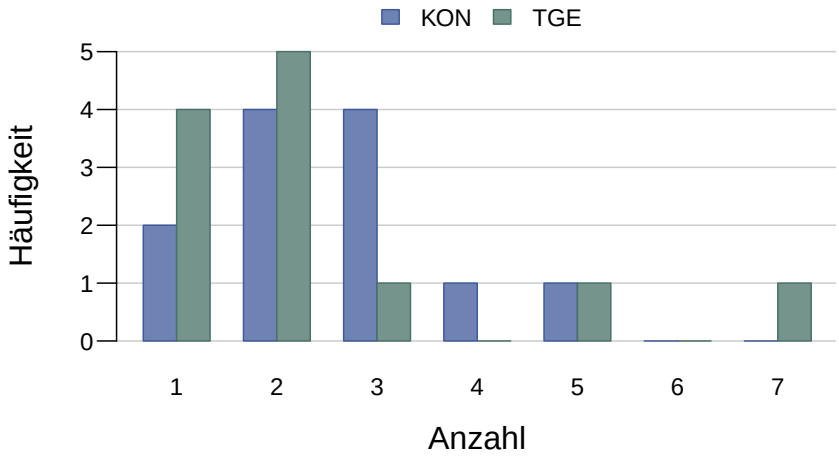


Abbildung 6.12: Anzahl der Akzeptanztestläufe.

Zum Abschluss betrachten wir die Anzahl der Akzeptanztestläufe, die von den Teilnehmern benötigt wurden, bis ihre Programme den Akzeptanztest bestanden. Zur Erinnerung: Die Teilnehmer hatten die Information, dass der Akzeptanztest beim ersten Versuch zu bestehen ist und sie daher die Programme vor Abgabe gründlich testen sollen. Dennoch haben dies, wie man in Abbildung 6.12 erkennen kann, die wenigsten geschafft. Weiter sieht man, dass die Anzahl der benötigten Akzeptanztests weitgehend

unabhängig vom eingesetzten Entwicklungsprozess zu sein scheint. Dies bestätigt sich auch beim Test der Nullhypothese H_0^{AT}. Das Ergebnis zeigt keinen signifikanten Unterschied zwischen den Prozessen ($p = 0{,}336$).

6.7 Alternative Auswertungen und weitere Ergebnisse

Die bisherigen Ergebnisse wurden durch die Auswertung des Datensatzes nach der unabhängigen Variablen dieser Studie, nämlich dem verwendeten Entwicklungsprozess, gewonnen. Wir haben den Datensatz zusätzlich nach der verwendeten Aufgabe und der Reihenfolge der Bearbeitung ausgewertet. Die Ergebnisse hierzu werden in diesem Abschnitt wiedergeben, wobei wir hier nur auf Unterschiede eingehen, die signifikant oder zumindest fast signifikant sind. Die vollständigen Ergebnisse zu den alternativen Auswertungen finden sich in tabellarischer Form im Anhang C.

6.7.1 Auswertung nach der Aufgabe

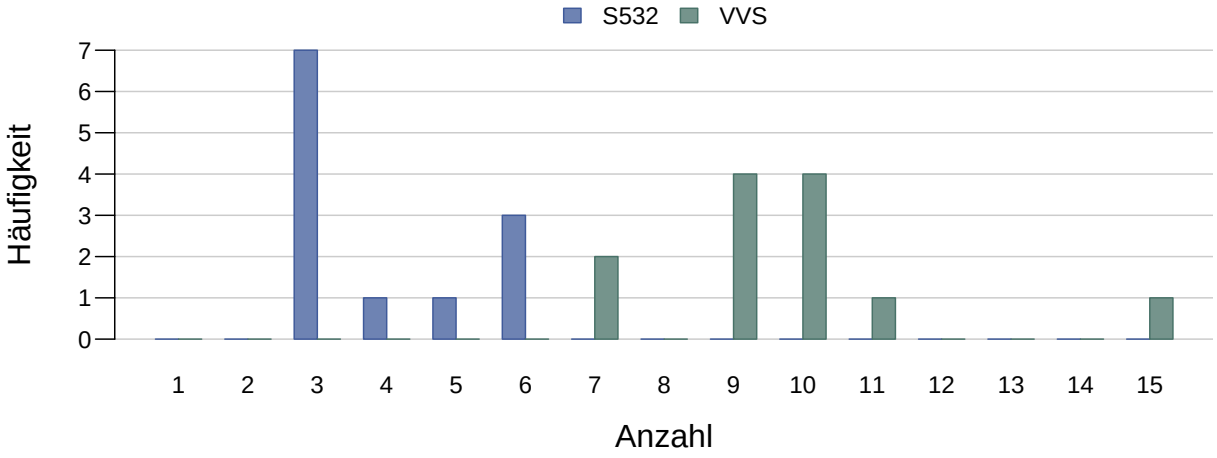


Abbildung 6.13: Anzahl der geänderten Dateien (nach Aufgabe).

Die ersten auffälligen Unterschiede zeigen sich bei der Auswertung nach der Aufgabe bei Metriken, die unter der Kategorie Produktivität eingeordnet sind. Das deutlichste Bild zeigt sich dabei, wenn man die Anzahl der geänderten Dateien in Abbildung 6.13 betrachtet. Wie man sehen kann, gibt es noch nicht einmal eine Überschneidung zwischen den beiden Aufgaben. Dementsprechend weist der dazugehörige Hypothesentest diese Unterschied mit einem p-Wert kleiner 0,001 als signifikant aus.

Diese Beobachtung lässt sich aus den Anforderungen der Aufgaben an die Teilnehmer erklären. Um die Aufgabe Videoverleihsystem lösen zu können, muss man mindestens vier Klassen ändern. Wenn man jedoch objektorientiert arbeitet und die dazugehörigen Testfälle anpasst und erweitert, werden es jedoch schnell sieben oder mehr Klassen, die zur Lösung der Aufgabe geändert oder sogar neu erstellt werden müssen. Zur Lösung der Aufgaben Soundex kommt man mit deutlich weniger geänderten Dateien aus. Minimal muss nur eine Klasse erstellt und ein weitere geändert werden. Schreibt man zur neu erstellten Klasse noch eine Testklasse, wie es erwünscht war, dann kommt man insgesamt auf drei geänderte Dateien.

Da sich besonders neu erstellte Klassen auch in geänderten Zeilen Code niederschlagen, ist es nicht verwunderlich, dass sich auch bei den Netto-Änderungen ein Trend in die gleiche Richtung beobachten lässt. Der Unterschied zwischen den beiden Aufgaben bezüglich der Netto-Änderungen ist sogar beinahe signifikant, wie der p-Wert von 0,06 zeigt.

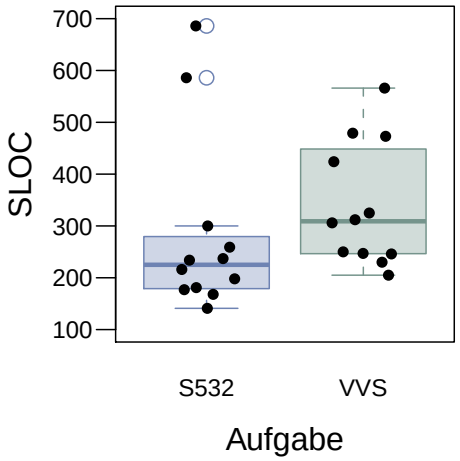


Abbildung 6.14: Netto-Änderungen (nach Aufgabe)

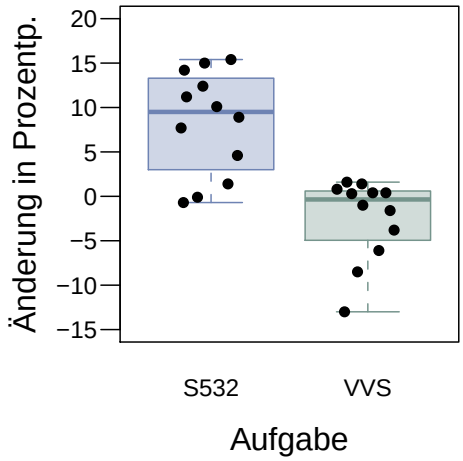


Abbildung 6.15: Änderung der Anweisungsabdeckung des Anweisungscodes (nach Aufgabe)

Ein weiterer signifikanter Unterschied ($p = 0{,}001$) zwischen den beiden Aufgaben zeigt sich bei der Veränderung der Anweisungsabdeckung, die in Abbildung 6.15 abgebildet ist. Auch dieser Unterschied kommt jedoch nicht überraschend. Vergleicht man die in den Abschnitten 6.5.1 und 6.5.2 präsentierten Kennzahlen des ausgelieferten Codes zu den Aufgaben, fällt zunächst auf, dass der Code der Aufgabe Videoverleihsystem

mit 3391 SLOC größer ist als der Code der Aufgabe Soundex mit 1171 SLOC. Weiterhin hat der Anwendungscode der Aufgabe Videoverleihsystem mit 86,3 Prozent eine höhere Anweisungsabdeckung als der Anwendungscode der Aufgabe Soundex mit 63,5 Prozent. Die Kombination dieser zwei Faktoren sorgt dafür, dass es bei der Aufgabe Videoverleihsystem schwerer ist, die Anweisungsabdeckung zu erhöhen, als dies bei der Aufgabe Soundex der Fall ist. Folglich liegt der Median der Veränderung der Anweisungsabdeckung bei der Aufgabe Videoverleihsystem mit $-0,35$ Prozentpunkten nahe dem Nullpunkt, wohingegen ein Median von $9,5$ Prozentpunkten bei der Aufgabe Soundex eine deutliche Erhöhung der Anweisungsabdeckung anzeigt.

6.7.2 Auswertung nach der Reihenfolge der Bearbeitung

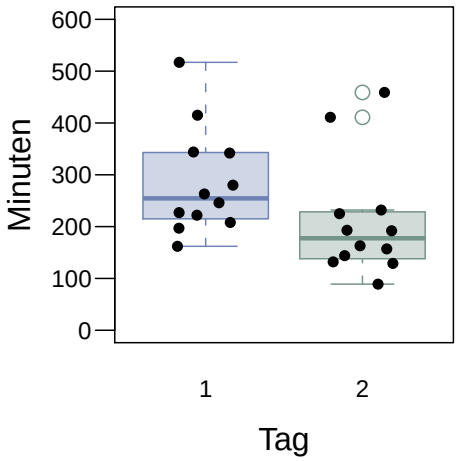


Abbildung 6.16: Bearbeitungszeit (nach Reihenfolge)

Bei der Auswertung nach der Reihenfolge der Bearbeitung stach lediglich die in Abbildung 6.16 dargestellte Bearbeitungszeit ins Auge. Wie man dort sehen kann, ist der Unterschied jedoch deutlich. Im Durchschnitt betrug die Bearbeitungszeit am zweiten Tag nur noch etwa 73 Prozent der Bearbeitungszeit des ersten Tages. Es ist also ein deutlicher Lerneffekt erkennbar. Der Unterschied ist, wie der p-Wert des entsprechenden Wilcoxon-Tests mit 0,024 zeigt, auch signifikant. Welche Auswirkungen der Lerneffekt auf die Validität der Studie hatte, wird im nächsten Abschnitt thematisiert.

6.8 Validität der Studie

Dieser Abschnitt befasst sich, ebenso wie die entsprechenden Abschnitte in den beiden vorhergehenden Kapiteln, mit den Einschränkungen, die für die Validität der Studie gelten. Wie in Abschnitt 4.7 eingeführt, unterscheiden wir dabei zwischen der Validität des Konstrukts, der statistischen sowie der internen und externen Validität.

6.8.1 Validität des Konstrukts

Bei der Validität des Konstrukts gelten für dieses Experiment die gleichen Einschränkungen bezüglich der Verwendung der Anzahl der Akzeptanztest als Maß für die Qualität der Programme wie für die anderen beiden in dieser Arbeit vorgestellten Studien. Deshalb sei an dieser Stelle auf die entsprechenden Ausführungen in Abschnitt 4.7.1 der ersten Studie in dieser Arbeit verwiesen.

Auch die kritischen Stellen, was die Möglichkeit von Messfehlern betrifft, sind aus den vorhergehenden Studien bereits bekannt: Dies sind zum einen die von den Teilnehmern ausgefüllten Zeitprotokolle und zum anderen die manuell klassifizierten Änderungen. Um die Zuverlässigkeit der Zeitprotokolle zu erhöhen, haben wir während des Experiments immer wieder die Zeitprotokolle kontrolliert und die Teilnehmer bei Pausen gebeten, diese korrekt einzutragen. Um Fehler bei der manuellen Klassifizierung zu vermeiden, haben wir diese, wie schon in Abschnitt 4.7.1 beschrieben, nach dem Vier-Augen-Prinzip durchgeführt.

Wie Tabelle 6.6 zeigt, mussten wir zur Auswertung dieses Experiments insgesamt 57,7 Prozent der Änderungen von Hand klassifizieren. Dieser hohe Anteil erklärt sich dadurch, dass bei den Datenpunkten mit konventioneller Entwicklung erwartungsgemäß nur sehr wenige Änderungen automatisch als *testgetrieben* klassifizierbar sind. Mit Ausnahme der erkannten automatischen Umstrukturierungen müssen dann alle verbleibenden Änderungen manuell klassifiziert werden. Trotz der umfangreichen manuellen Klassifikation entstanden bei nur 3,0 Prozent aller Änderungen Konflikte. Dies entspricht etwa einem Konflikt bei jeder zwanzigsten manuell zu klassifizierenden Änderung.

6.8.2 Statistische Validität

Wie auch in den beiden anderen Studien wurde für die Hypothesentests der zweiseitige Wilcoxon-Test eingesetzt. Mit dem Signifikanzniveau von 5 Prozent ist gleichzeitig auch die maximale Wahrscheinlichkeit für einen Fehler 1. Art gegeben.

Um die obere Schranke für einen Fehler 2. Art zu berechnen, schätzen wir zunächst die Güte des Wilcoxon-Tests nach unten ab. Dazu berechnen wir die Güte des entsprechen-

Klasse	*automatisch*		*manuell*			
			ohne Konflikt		*mit Konflikt*	
testgetriebene Änderungen	905	32,1 %	8	0,3 %	7	0,2 %
Umstrukturierungen	287	10,2 %	38	1,3 %	48	1,7 %
nicht testgetr. Änderungen	–		1496	53,1 %	29	1,0 %
Summe	1192	42,3 %	1542	54,7 %	84	3,0 %

Tabelle 6.6: Anteile der automatisch, manuell ohne Konflikt und manuell mit Konflikt klassifizierten Änderungen.

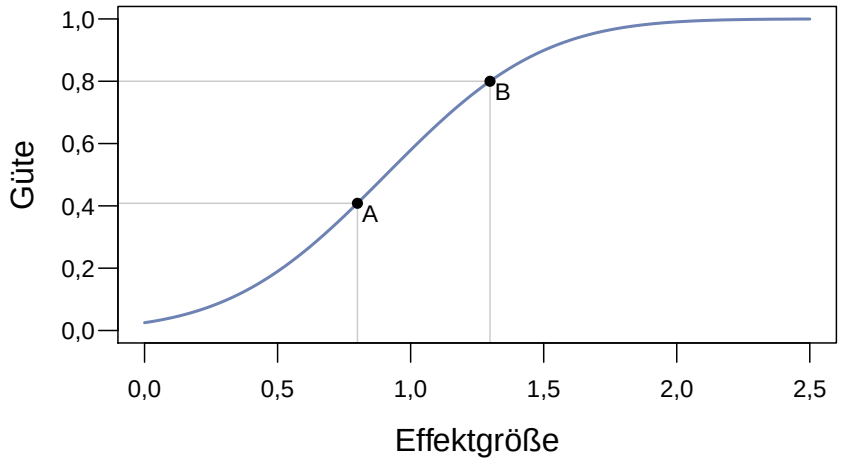


Abbildung 6.17: Zusammenhang zwischen Effektgröße und Güte für den Wilcoxon-Test bei einer Gruppengröße von 12 und einem Signifikanzniveau von 0,05 und einer ARE von 0,864.

den t-Tests für die um den Faktor der minimalen ARE von 0,864 reduzierte Gruppengröße von 12. Auf diese Weise lässt sich die dargestellte Kurve für den Zusammenhang von Effektgröße und Güte bestimmen. Wie durch Punkt A in Abbildung 6.17 markiert, ergibt sich daraus für den Wilcoxon-Test bei einer großen Effektgröße von 0,8 eine untere Schranke der Güte von 40,8 Prozent. Dies ist in etwa die Hälfte der üblicherweise gewünschten Güte von 80 Prozent (vgl. [Coh88, S. 531]) und bedeutet, dass eine Chance von etwa 60 Prozent besteht, dass große Effekte nicht aufgedeckt werden. Eine Güte von 80 Prozent wird, wie an Punkt B zu sehen, erst für eine Effektgröße von 1,299 erreicht.

Gruppe	*KON*	*TGE*	*keine Präferenz*
KV/TS	2	0	1
KS/TV	0	1	2
TV/KS	0	0	3
TS/KV	2	0	1
Summe	4	1	7

Tabelle 6.7: Verteilung der Antworten auf die Frage „Welchen Entwicklungsprozess bevorzugen Sie privat?“.

Prozess	*trifft voll zu*	*trifft eher zu*	*unentschieden*	*trifft eher nicht zu*	*trifft überhaupt nicht zu*
KON	1	3	7	0	1
TGE	3	8	0	1	0
Summe	4	11	7	1	1

Tabelle 6.8: Verteilung der Bewertungen der Aussage „Die Aufgabe unterstützt die Herangehensweise mit dem vorgegebenen Entwicklungsprozess.“ in Abhängigkeit des zum Lösen der Aufgabe verwendeten Entwicklungsprozesses.

1. Prozess	*bei d. Aufgabe mit KON*		*bei beiden gleich*	*bei d. Aufgabe mit TGE*	
	definitiv	*eher*		*eher*	*definitiv*
KON	0	3	2	1	0
TGE	0	1	2	2	1
Summe	0	4	4	3	1

Tabelle 6.9: Verteilung der Antworten auf die Frage „Bei welcher Aufgabe waren Sie motivierter?“.

1. Prozess	*bei d. Aufgabe mit KON*		*bei bei-*	*bei d. Aufgabe mit TGE*	
	definitiv	*eher*	*den gleich*	*eher*	*definitiv*
KON	1	2	3	0	0
TGE	1	0	2	2	1
Summe	2	2	5	2	1

Tabelle 6.10: Verteilung der Antworten auf die Frage „Bei welcher Aufgabe waren Sie konzentrierter?“.

6.8.3 Interne Validität

Die erste potentielle Bedrohung der internen Validität betrifft die Sterblichkeit. Auch in diesem Experiment musste ein Datenpunkt von der Auswertung ausgeschlossen werden. In diesem Fall stellt die Sterblichkeit jedoch eine Stärkung der internen Validität dar, da sie zufällig dazu geführt hat, dass die Gruppen exakt gleich groß sind und die Gegenbalancierung damit perfekt greifen kann.

Eine weitere mögliche Bedrohung für die interne Validität ist der in Abschnitt 6.7.2 diskutierte Lerneffekt. Allerdings stellt er wegen der Gegenbalancierung kein Problem dar, da er in allen der vier Gruppen des gegenbalancierten Entwurfs (vgl. Tabelle 6.2) in die gleiche Richtung ging und auch in allen Gruppen in etwa gleich groß war. Kritischer ist da schon die Frage, ob es Faktoren gab, die einen der beiden Entwicklungsprozesse bevorteilten. Ein solcher Faktor ergibt sich durch die Präferenzen der Teilnehmer für einen der Entwicklungsprozesse. Wir haben die Teilnehmer deswegen gebeten, die Frage „Welchen Entwicklungsprozess bevorzugen Sie privat?“ zu beantworten. Sie konnten dabei ihre Präferenz für die konventionelle und testgetriebene Entwicklung zum Ausdruck bringen oder angeben, dass sie keine Präferenz haben. Wie man in Tabelle 6.7 sehen kann, äußerten die meisten Teilnehmer, dass sie keinen der beiden Entwicklungsprozesse bevorzugen. Da aber vier Teilnehmer angaben, die konventionelle Entwicklung zu bevorzugen und nur ein Teilnehmer eine Präferenz für die testgetriebene Entwicklung äußerte, können wir nicht ausschließen, dass es hierdurch zu einer leichten Verfälschung der Ergebnisse kam.

Die interne Validität ist zudem eingeschränkt, falls die Aufgaben einen der Entwicklungsprozesse stärker unterstützt als den anderen. Tatsächlich könnte es durch die Menge an mitgelieferten Testfällen und die gute Testbarkeit der ausgeteilten Programme zu einer Begünstigung der testgetriebenen Entwicklung gekommen sein (vgl. Abschnitt 6.5). Diese Befürchtung wird durch die Wahrnehmung der Teilnehmer bestärkt. Sie waren von uns im Fragebogen zur Aufgabe aufgefordert worden, die Aussage „Die Aufgabe

Gruppe	*ja*	*etwas*	*nein*
KV/TS	1	1	1
KS/TV	0	0	3
TV/KS	0	1	1
TS/KV	1	0	2
Summe	2	2	7

Tabelle 6.11: Verteilung der Antworten auf die Frage „Hat Sie die Teilnahme am Experiment in Ihrer Arbeitsweise beeinträchtigt?“. (Ein Teilnehmer machte hier keine Angabe.)

unterstützt die Herangehensweise mit dem vorgegebenen Entwicklungsprozess.“ auf einer Likert-Skala von *trifft voll zu* bis *trifft überhaupt nicht zu* zu bewerten. Wie in Tabelle 6.8 zu sehen, lässt sich bei der Auswertung der Bewertungen in Abhängigkeit des zum Lösen der Aufgabe verwendeten Entwicklungsprozesses ein Verschiebung zu Gunsten des TGE-Prozesses feststellen. Umso erstaunlicher ist es, dass sich im Experiment keine Vorteile der testgetriebenen Entwicklung nachweisen ließen.

Weithin kann es für die interne Validität nachteilig sein, wenn die Teilnehmer bei einem der Entwicklungsprozesse motivierter oder konzentrierter gearbeitet haben. Aus diesem Grund haben wir die Teilnehmer die Aussagen „Bei welcher Aufgabe waren Sie motivierter?“ und „Bei welcher Aufgabe waren Sie konzentrierter?“ beurteilen lassen. Zur Beantwortung war eine Skala mit fünf Antwortmöglichkeiten vorgegeben, wobei *definitiv bei Aufgabe 1* bzw. *definitiv bei Aufgabe 2* die äußeren Extreme bildete und *bei beiden Aufgaben gleich* die Mitte markierte. Wir haben die auf die beiden Fragen gegebenen Antworten in Abhängigkeit des für die jeweilige Aufgabe verwendeten Entwicklungsprozesses ausgewertet. Die hieraus entstehenden Verteilungen sind in den Tabellen 6.9 und 6.10 dargestellt. Wie man dort sehen kann, sind beide Verteilungen fast perfekt ausbalanciert und es scheint dabei auch keinen Unterschied zu machen, mit welchem Entwicklungsprozess begonnen wurde. Eine Bedrohung der internen Validität durch unterschiedliche Motivation oder Konzentration bei den Entwicklungsprozessen liegt daher wahrscheinlich nicht vor.

6.8.4 Externe Validität

Für die externe Validität gilt, was auch schon für die anderen Studien galt: Durch die Auswahl der Teilnehmer nach ihrer Verfügbarkeit sind die Ergebnisse nur auf Populationen übertragbar, die in ihren Eigenschaften denen unserer Teilnehmer ähnlich sind.

Gruppe	*sehr gut*	*gut*	*mittel-mäßig*	*schlecht*	*sehr schlecht*
KV/TS	0	3	0	0	0
KS/TV	1	2	0	0	0
TV/KS	0	2	1	0	0
TS/KV	0	2	1	0	0
Summe	1	9	2	0	0

Tabelle 6.12: Verteilung der Antworten auf die Frage „Wie hat Ihnen die Programmierung im Experiment gefallen?“.

Dabei liefert der aufgetretene Lerneffekt ein starkes Indiz dafür, dass geringe Programmiererfahrung eine wesentliche Eigenschaft der Teilnehmer darstellt und bei der Übertragung der Ergebnisse unbedingt beachtet werden sollte.

Darüber hinaus lassen sich aufgrund der geringen Dauer des Experiments bestenfalls Vermutungen über Langzeiteffekte der Entwicklungsprozesse anstellen. Des Weiteren ist eine Übertragung der beobachteten Effekte sicherlich nur auf objektorientierte Sprachen möglich, die der in den Aufgaben verwendeten Programmiersprache Java ähneln und für die ein mit JUnit vergleichbares Testrahmenwerk existiert.

Abgesehen von den eben genannten Punkten kann die externe Validität beeinträchtigt sein, wenn die Teilnehmer wegen der ungewohnten Experimentsituation in ihrer üblichen Arbeitsweise beeinträchtigt waren. Um zu sehen, ob dies der Fall war, haben wir die Teilnehmer aufgefordert, die Frage „Hat Sie die Teilnahme am Experiment in Ihrer Arbeitsweise beeinträchtigt?“ mit *ja*, *etwas* oder *nein* zu beantworten. Wie in Tabelle 6.11 ersichtlich, verneinten die meisten Teilnehmer diese Frage, so dass wir davon ausgehen können, dass die Einschränkung der externen Validität durch die ungewohnte Experimentsituation gering ist.

Schließlich könnte die externe Validität durch mangelnde Motivation der Teilnehmer gefährdet sein. So könnten viele der Teilnehmer insbesondere demotiviert gewesen sein, da sie die in der Vertraulichkeitserklärung angegebene Arbeitszeit von drei bis vier Stunden pro Tag überschritten hatten. Deshalb haben wie die Teilnehmer gebeten, die Frage „Wie hat Ihnen die Programmierung im Experiment gefallen?“ zu beantworten. Vorgegeben war dabei eine Likert-Skala mit fünf Antwortmöglichkeiten von *sehr gut* bis *sehr schlecht*. Tabelle 6.12 zeigt die Verteilung der Antworten auf diese Frage. Abgesehen von zwei Teilnehmern, die mit *mittelmäßig* antworteten gefiel den Teilnehmern die Programmierung im Experiment *gut* oder *sehr gut*. Eine mangelnde Motivation der

Teilnehmer erscheint daher unwahrscheinlich.

6.9 Zusammenfassung und Diskussion der Ergebnisse

In diesem Kapitel wurde ein Experiment vorgestellt, das die testgetriebene und die konventionelle Entwicklung in Bezug auf die Produktivität der Entwickler und die Qualität der Programme untersucht. Signifikante Unterschiede zwischen den Entwicklungsprozessen zeigten sich fast ausschließlich bei den dem TGE-Prozess zugeordneten Metriken. So trat der erste signifikante Unterschied zwischen den Entwicklungsprozessen bei der TGE-Prozesstreue auf. Diese war, wie bei Einhaltung der Entwicklungsprozesse durch die Teilnehmer nicht anders zu erwarten, bei Verwendung des TGE-Prozess höher als bei Verwendung des KON-Prozess und bestätigt somit nur die Validität unserer Studie.

Auch der Unterschied beim Variationskoeffizient des Abstands der Testläufe zeigt nur, dass die Teilnehmer im Großen und Ganzen die von ihnen geforderten Entwicklungsprozesse eingehalten haben. Da beim TGE-Prozess eine regelmäßige Testausführung gefordert war während beim KON-Prozess die Testläufe idealerweise nur nach Abschluss der Implementierung in der Testphase laufen sollten, ist es keine Überraschung, dass der Variationskoeffizient beim KON-Prozess größer ausfällt als beim TGE-Prozess.

Das letzte Ergebnis dieser Studie bezüglich der Unterschiede zwischen den Entwicklungsprozessen ist ein signifikant höherer Anteil an Änderungen im Testcode bei Einsatz des TGE-Prozesses. Dieses Ergebnis ist besonders interessant, da es bei Einsatz des TGE-Prozess zudem einen Trend zu mehr Netto-Änderungen gab, der aber nicht zu einer höheren Bearbeitungszeit bei Verwendung des TGE-Prozess geführt hat. Dies ist erstaunlich, da der in die Testfälle investierte Aufwand nicht direkt zum erfolgreichen Erfüllen der in der Aufgabenstellung gestellten Anforderungen beiträgt.

Ansonsten konnten wir bezüglich der Produktivität keine signifikanten Unterschiede zwischen den Entwicklungsprozessen feststellen. Auch bei den der Qualität zugeordneten Metriken traten keine signifikanten Unterschiede auf. Damit stehen die Ergebnisse dieser Studie im Widerspruch zu den Studien von Edwards [Edw03] und George und Williams [GW04]. Beide hatten, ebenfalls mit Anfängern in der testgetriebenen Entwicklung, die Unterschiede zwischen testgetriebener und konventioneller Entwicklung untersucht und dabei deutliche Vorteile der testgetriebenen Entwicklung bezüglich der Qualität der Programme festgestellt. Bei [GW04] wurde zusätzlich eine längere Dauer für die Implementierung ermittelt. Neben den bereits bei der Besprechung der verwandten Arbeiten in Abschnitt 3.2.2 kritisierten Punkte könnte eine Ursache für die widersprüchlichen Ergebnisse darin liegen, dass sowohl Edwards als auch George und Wil-

liams bei der konventionellen Entwicklung keine Tests von den Teilnehmern gefordert hatten. Somit dokumentieren ihre Ergebnisse eher den Nutzen des Testens im allgemeinen als den der testgetriebenen Entwicklung.

Neben den Unterschieden zwischen den Entwicklungsprozessen hatten wir bei der Auswertung nach den bearbeiteten Aufgaben sowie bei der Auswertung nach der Reihenfolge der Bearbeitung weitere signifikante Unterschiede festgestellt. Dabei stellt die bei der Reihenfolge der Bearbeitung festgestellte Verkürzung der Bearbeitungszeit vom ersten auf den zweiten Tag ein starkes Indiz für die geringe Programmiererfahrung der Teilnehmer dar und sollte daher bei der Übertragung der Ergebnisse unbedingt berücksichtigt werden.

Die signifikanten Unterschiede bei der Anzahl der geänderten Dateien und der Änderung der Anweisungsabdeckung die durch die Auswertung des Datensatzes nach der bearbeiteten Aufgabe aufgedeckt wurden, zeigen, wie stark einige Metriken von den im Experiment verwendeten Aufgaben abhängen. Daher sollten in zukünftigen Studien die Aufgaben möglichst gut beschrieben werden, damit sich deren Einfluss auf die Ergebnisse möglichst gut einschätzen lassen. Weiterhin kann aus diesem Grund die Replikation von Studien unter Verwendung anderer Aufgaben einen wertvollen Beitrag zum empirischen Wissen über agile Softwareentwicklungsprozesse leisten.

Kapitel 7

Fazit und Ausblick

Wir haben in dieser Arbeit drei Studien zu Fragen aus dem Umfeld der agilen Softwareentwicklung vorgestellt. Die erste Studie in dieser Arbeit befasste sich dabei mit den Unterschieden bei testgetriebenen Entwicklungsprozessen zwischen erfahren und unerfahrenen Programmierpaaren. Wir konnten dabei deutliche Unterschiede zwischen erfahrenen und unerfahren Entwicklerpaaren bei den (automatischen) Umstrukturierungen der Bearbeitungszeit und den Netto-Änderungen nachweisen. Aufgrund dieser Ergebnisse sehen wir die Übertragbarkeit von mit Studenten oder vergleichbar unerfahrenen Entwicklern gewonnen Ergebnissen als nur in den Sonderfällen als gegeben an, in denen sie zuvor empirisch schlüssig nachgewiesen werden konnte. Des Weiteren ergibt sich aus unseren Ergebnissen der Bedarf nach mehr Studien mit professionellen und in agiler Softwareentwicklung erfahrenen Entwicklern. Aus eigner Erfahrung wissen wir jedoch auch, dass gerade diese Entwickler schwer für Studien zu gewinnen sind, daher werden vor allem bei Studien, die den Charakter von Pilotstudien haben, studentische Teilnehmer weiterhin eine Daseinsberechtigung haben.

Die zweite in dieser Arbeit präsentierte Studie konnte ihr Ziel, nämlich die Frage zu klären, ob Programmierpaare, in denen die Programmiere unterschiedliche Persönlichkeitsmerkmale haben, anders arbeiten als Paare, in denen die Programmierer gleiche Persönlichkeitsmerkmale haben nicht erfüllen. Hierzu waren die Ergebnisse der Studie schlichtweg nicht deutlich genug und standen in einem nicht eindeutig erklärbaren Widerspruch zu der Studie, die uns ursprünglich zur Durchführung unserer Studie animiert hatte. Die Lehre, die wir hieraus ziehen konnten ist, dass es in Zukunft nötig sein wird, sich zunächst einmal damit befassen, inwieweit Persönlichkeitsmerkmale überhaupt einen Einfluss auf die Effektivität eines Programmierers haben.

Aus der dritten Studie dieser Arbeit, in der wir die testgetriebene mit der konventionellen Entwicklung verglichen hatten, geht zum einen die ermutigende Erkenntnis hervor, dass auch in der testgetriebenen Entwicklung unerfahrene Entwickler nach kurzer aber intensiver Ausbildung in der Lage sind, die geforderten testgetriebenen Entwicklungsprozess einzuhalten. Zum anderen weist der höhere Anteil der Netto-Änderungen im

Testcode bei Verwendung des TGE-Prozess auf einen bisherigen Studien nicht beachteten Unterschied zwischen testgetriebener und konventioneller Entwicklung hin, den wir aber in unserer Studie nicht vollständig aufklären konnten und der daher weiteren Forschungsbedarf nach sich zieht.

Weiterer Forschungsbedarf ergibt auch aus der Tatsache, dass diese letzte Studie wie auch die anderen beiden Studien dieser Arbeit mit einer relativ kleinen Anzahl an Teilnehmern durchgeführt wurde. Dadurch lassen sich kleine Unterschiede praktisch nicht aufdecken. Um eine schlüssige Antwort über Vor- und Nachteile der testgetriebenen Entwicklung gegenüber ihrem konventionellen Pendant zu erhalten, wäre ein Vergleich der beiden Entwicklungsprozesse in einem Experiment mit großen Gruppen wünschenswert. Damit die Übertragbarkeit der Ergebnisse eines solchen Experiments hoch ist, sollte es sich bei den Teilnehmern idealerweise um in den Entwicklungsprozessen erfahrene, professionelle Softwareentwickler handeln. Zudem sollte die Einhaltung der Entwicklungsprozesse während des Experiments überprüft werden, damit sich mangelnde Prozesstreue als Gefahr für die Gültigkeit einer solchen Studie ausschließen lässt. Zur Prüfung der Prozesstreue bietet sich ein Werkzeug wie TestPulse an, das zu allen Studien dieser Arbeit einen unverzichtbaren Beitrag geleistet hat.

TestPulse oder vergleichbare Werkzeuge ließen sich ebenfalls einsetzen, um zu klären, ob die Paarprogrammierung einen positiven Effekt auf die Anwendung der testgetriebenen Entwicklung hat, wie es von Anhängern des Extreme Programmings oft behauptet wird. Ein weiteres Einsatzszenario für TestPulse oder ähnliche Werkzeuge wäre die Ermittlung der Prozesstreue in realen Projekten in der Industrie, bestenfalls über einen längeren Zeitraum hinweg. Dies würde einerseits zeigen wie genau die Regeln der testgetriebenen Entwicklung in der Praxis tatsächlich befolgt werden und würde andererseits helfen, die in den Experimenten ermittelten Werte für die Prozesstreue einzuordnen.

Wie man an den eben genannten Vorschlägen für mögliche Folgestudien sehen kann, bestehen im Umfeld der agilen Entwicklungsprozesse und Methoden noch genug unbeantwortete Forschungsfragen. Dies gilt insbesondere, wenn man bedenkt, dass andere Praktiken des Extreme Programmings wie beispielsweise die kontinuierliche Integration und die häufige Veröffentlichung der Software bisher kaum empirisch untersucht wurden. Es besteht also weiterhin Bedarf an empirischen Studien zum Thema agile Softwareentwicklung und man darf gespannt sein, welche Ergebnisse diese Studien in Zukunft bringen werden.

Anhang A

Ergänzende Grafiken und Tabellen zu Kapitel 4

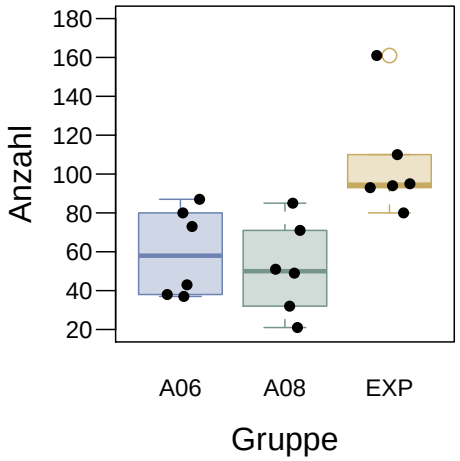


Abbildung A.1: Anzahl der Testläufe.

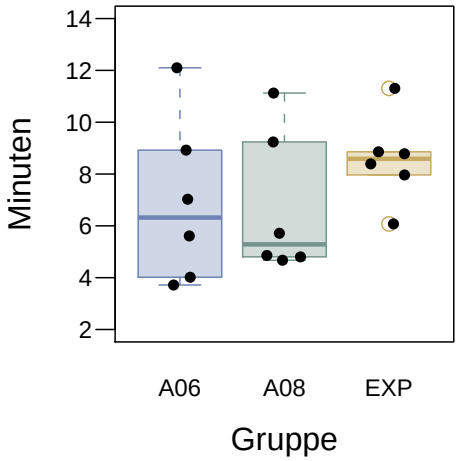


Abbildung A.2: Standardabweichung des Abstands der Testläufe.

Metrik	*p-Wert (nach Bonferroni-Holm korrigiert)*		
	A06 vs. A08	*A06 vs. EXP*	*A08 vs. EXP*
Studiendauer	0,014*	< 0,001*	< 0,001*
Programmiererfahrung ...			
allgemein [Jahre]	0,239	–	–
mit Paarprogrammierung [Jahre]	0,527	–	–
mit Testautomatisierung [Jahre]	0,078	–	–
mit testgetriebener Entwicklung [Jahre]	0,037*	–	–
mit JUnit	0,373	–	–
mit Java	0,013*	–	–
Industrielle Programmiererfahrung ...			
mit Paarprogrammierung [Jahre]	0,079	< 0,001*	< 0,001*
mit Testautomatisierung [Jahre]	0,359	< 0,001*	< 0,001*
mit testgetriebener Entwicklung [Jahre]	1	< 0,001*	< 0,001*
mit JUnit [Jahre]	1	< 0,001*	< 0,001*
mit Java [Jahre]	0,655	< 0,001*	< 0,001*

* signifikant auf dem 5 Prozent-Niveau

Tabelle A.1: Nach Bonferroni-Holm korrigierte *p*-Werte der zum Vergleich der in den Gruppen vorhandenen Vorkenntnisse durchgeführten Wilcoxon-Tests.

Metrik	*Gruppe*	*Min.*	*1. Qrt.*	*Median*	*3. Qrt.*	*Max.*	*Mittel.*	*Std. abw.*
Studiendauer [Jahre]	A06	3	3	3	3	5	3,25	0,622
	A08	3	3,75	4	4	5	3,833	0,577
	EXP	4	5	6,5	8	10	6,65	2,161
Programmiererfahrung ...								
allgemein [Jahre]	A06	1,5	3	4	5	12	4,792	3,086
	A08	3	3,75	7	7,5	10	6,083	2,575
	EXP	–	–	–	–	–	–	–
mit Paarprogrammierung [Jahre]	A06	0	0	0	0,625	2	0,4	0,634
	A08	0	0	0	0,2	1	0,175	0,322
	EXP	–	–	–	–	–	–	–
mit Testautomatisierung [Jahre]	A06	0	0	0	0	0	0	0
	A08	0	0	0	0,25	1	0,25	0,452
	EXP	–	–	–	–	–	–	–
mit testgetriebener Entwicklung [Jahre]	A06	0	0	0	0	0	0	0
	A08	0	0	0	0,2	1	0,175	0,322
	EXP	–	–	–	–	–	–	–
mit JUnit [Jahre]	A06	0	0	0	0	3	0,25	0,866
	A08	0	0	0	0,25	1	0,25	0,452
	EXP	–	–	–	–	–	–	–
mit Java [Jahre]	A06	0,5	1	1,5	2,5	5	1,958	1,514
	A08	0,5	3	4	6	8	4,208	1,994
	EXP	–	–	–	–	–	–	–
Industrielle Programmiererfahrung ...								
mit Paarprogrammierung [Jahre]	A06	0	0	0	0,075	1	0,15	0,312
	A08	0	0	0	0	0	0	0
	EXP	2	3,688	5,5	8	8	5,521	2,506
mit Testautomatisierung [Jahre]	A06	0	0	0	0	0	0	0
	A08	0	0	0	0	1	0,083	0,289
	EXP	2	3,75	5,5	7,25	8	5,333	2,219
mit testgetriebener Entwicklung [Jahre]	A06	0	0	0	0	0	0	0
	A08	0	0	0	0	0	0	0
	EXP	1	1,75	2,25	4,25	6,5	3	1,942
mit JUnit [Jahre]	A06	0	0	0	0	3	0,25	0,866
	A08	0	0	0	0	1	0,083	0,289
	EXP	2	3,75	5,75	8	8	5,583	2,42
mit Java [Jahre]	A06	0	0	0	0,075	5	0,608	1,497
	A08	0	0	0	0	3	0,333	0,888
	EXP	2	5	7,75	10	11	7,167	3,193

Tabelle A.2: Lage- und Streuungsmaße der zu den Vorkenntnissen der Teilnehmer erhobenen Metriken.

Metrik	*p-Wert (nach Bonferroni-Holm korrigiert)*		
	A06 vs. EXP	*A08 vs. EXP*	*A06 vs. A08*
Kategorie: TGE-Prozess			
TGE-Prozesstreue [%]	0,721	0,788	0,937
Anzahl der automatischen Umstrukturierungen	0,024*	0,014*	0,248
Anzahl der Testläufe	0,021*	0,013*	0,589
Mittlerer Abstand der Testläufe [min]	1	0,721	1
Standardabweichung des Abstands der Testläufe [min]	0,788	0,721	0,937
Variationskoeffizient des Abstands der Testläufe [%]	0,13	0,006*	0,394
Kategorie: Produktivität			
Bearbeitungszeit [min]	0,13	0,045*	0,937
Netto-Änderungen [SLOC]	0,155	0,045*	0,937
Anteil der Netto-Änderungen am Testcode [%]	0,929	0,929	0,929
Anzahl der geänderten Dateien	0,183	0,577	0,672
Kategorie: Qualität			
Änderung der Anzahl der Testfälle	0,373	0,373	0,87
Änderung der Anweisungsüberdeckung [%]	0,941	0,326	0,941
Anzahl der Akzeptanztestläufe	1	1	1

* signifikant auf dem 5 Prozent-Niveau

Tabelle A.3: Nach Bonferroni-Holm korrigierte *p*-Werte der zum Vergleich der Metriken zu TGE-Prozess, Produktivität und Qualität durchgeführten Wilcoxon-Tests.

Metrik	*Gruppe*	*Min.*	*1. Qrt.*	*Median*	*3. Qrt.*	*Max.*	*Mittel.*	*Std. abw.*
Kategorie: TGE-Prozess								
TGE-Prozesstreue [%]	A06	47,5	50,577	61,144	76,762	82,53	63,557	15,388
	A08	25,346	42,747	64,12	77,926	88,571	60,051	24,886
	EXP	61,446	67,259	72,614	82,006	96,421	75,618	12,943
Anzahl der automatischen	A06	0	0	0,5	1	2	0,667	0,816
Umstrukturierungen	A08	0	0	0	0	1	0,167	0,408
	EXP	1	6,75	9,5	10,75	21	9,667	6,623
Anzahl der Testläufe	A06	37	39,25	58	78,25	87	59,667	22,801
	A08	21	36,25	50	66	85	51,5	23,747
	EXP	80	93,25	94,5	106,25	161	105,5	28,808
Mittlerer Abstand der	A06	1,325	2,361	3,422	3,888	6,45	3,445	1,775
Testläufe [min]	A08	2,431	2,973	3,719	3,997	4,728	3,569	0,846
	EXP	2,12	2,713	3,273	3,423	3,745	3,068	0,61
Standardabweichung des	A06	3,717	4,416	6,32	8,45	12,1	6,9	3,202
Abstands der Testläufe [min]	A08	4,674	4,817	5,285	8,358	11,126	6,736	2,761
	EXP	6,076	8,072	8,588	8,84	11,308	8,564	1,688
Variationskoeffizient des	A06	176,514	178,498	183,57	226,868	303,257	210,836	51,294
Abstands der Testläufe [%]	A08	116,283	146,163	204,377	235,066	235,3	188,732	54,059
	EXP	237,418	260,637	282,129	298,11	328,664	281,208	32,649
Kategorie: Produktivität								
Bearbeitungszeit [min]	A06	139	157	174	242,75	262	194,333	54,357
	A08	97	150,5	173	198,5	275	178	59,823
	EXP	219	242,5	266,5	294,25	305	265,833	34,862
Netto-Änderungen [SLOC]	A06	84	112,75	147,5	174,75	312	162,667	81,377
	A08	106	120,5	145,5	155,5	223	148,5	41,707
	EXP	176	185	220,5	234,25	304	222,167	47,545
Anteil der Netto-Änd.	A06	59,524	64,294	65,007	72,624	77,841	67,766	7,072
im Testcode [%]	A08	44,872	56,883	62,117	73,906	78,302	63,397	12,831
	EXP	48,864	67,44	73,821	79,453	82,297	70,908	12,338
Anzahl der geänderten	A06	2	2	2,5	3	7	3,167	1,941
Dateien	A08	2	2,25	3	5,25	6	3,667	1,862
	EXP	4	4	4,5	5	6	4,667	0,816
Kategorie: Qualität								
Änderung der Anzahl	A06	5	6,25	7	9,25	18	8,833	4,792
der Testfälle	A08	5	7	7,5	8	15	8,333	3,445
	EXP	7	10	11	13,5	18	11,833	3,817
Änderung der Anweisungs-	A06	−7,3	−3,975	−1	0,55	0,7	−2,083	3,28
überdeckung [%]	A08	−10,1	−2,725	−1,7	−0,675	0,5	−2,733	3,813
	EXP	−9,6	−0,125	0,45	1,1	7	−0,117	5,353
Anzahl der Akzeptanz-	A06	1	1	1	1,75	4	1,667	1,211
testläufe	A08	1	1	1	1	3	1,333	0,816
	EXP	1	1	1	1,75	2	1,333	0,516

Tabelle A.4: Lage- und Streuungsmaße der zu Produktivität, Qualität und TGE-Prozess erhobenen Metriken.

Anhang B

Ergänzende Grafiken und Tabellen zu Kapitel 5

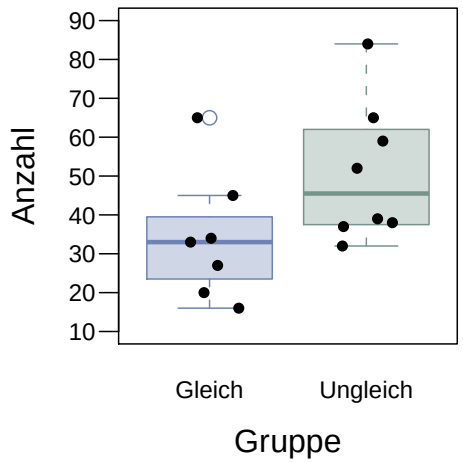


Abbildung B.1: Anzahl der Testläufe.

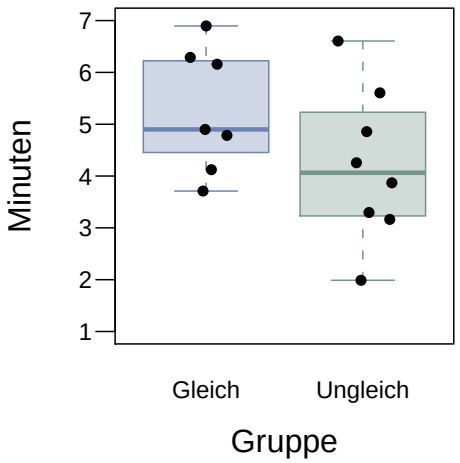


Abbildung B.2: Standardabweichung des Abstands der Testläufe

Metrik	*p-Wert*
Studiendauer [Jahre]	0,057
Programmiererfahrung ...	
allgemein [Jahre]	0,559
mit Paarprogrammierung [Jahre]	0,631
mit Testautomatisierung [Jahre]	0,372
mit testgetriebener Entwicklung [Jahre]	0,936
mit JUnit [Jahre]	0,831
mit Java [Jahre]	0,769
Industrielle Programmiererfahrung ...	
mit Paarprogrammierung [Jahre]	1
mit Testautomatisierung [Jahre]	0,497
mit testgetriebener Entwicklung [Jahre]	0,316
mit JUnit [Jahre]	0,136
mit Java [Jahre]	0,457

Tabelle B.1: *p*-Werte der zum Vergleich der in den Gruppen vorhandenen Vorkenntnisse durchgeführten Wilcoxon-Tests.

Metrik	*Gruppe*	*Min.*	*1. Qrt.*	*Median*	*3. Qrt.*	*Max.*	*Mittel.*	*Std. abw.*
Studiendauer [Jahre]	Gleich	2,5	3	3,5	4	9,5	3,846	1,796
	Ungleich	3	3,875	4	5,25	10	4,594	1,763
Programmiererfahrung ...								
allgemein [Jahre]	Gleich	1	5	7	10,5	16	7,893	4,456
	Ungleich	0	4	6	8,5	15	7	4,427
mit Paarprogrammierung [Jahre]	Gleich	0	0	0	0	2	0,286	0,726
	Ungleich	0	0	0	0	4	0,625	1,36
mit Testautomatisierung [Jahre]	Gleich	0	0	0	1	3	0,679	0,953
	Ungleich	0	0	0	0,125	6	0,719	1,732
mit testgetriebener Entwicklung [Jahre]	Gleich	0	0	0	0,75	1	0,286	0,469
	Ungleich	0	0	0	0,125	4	0,594	1,357
mit JUnit [Jahre]	Gleich	0	0	0	0,75	5	0,643	1,393
	Ungleich	0	0	0	0,125	5	0,656	1,535
mit Java [Jahre]	Gleich	0	1	3	4,75	9	3,393	2,83
	Ungleich	0	2	3	5	8	3,562	2,19
Industrielle Programmiererfahrung ...								
mit Paarprogrammierung [Jahre]	Gleich	0	0	0	0	0	0	0
	Ungleich	0	0	0	0	0	0	0
mit Testautomatisierung [Jahre]	Gleich	0	0	0	0	3	0,321	0,868
	Ungleich	0	0	0	0	2	0,125	0,5
mit testgetriebener Entwicklung [Jahre]	Gleich	0	0	0	0	1	0,071	0,267
	Ungleich	0	0	0	0	0	0	0
mit JUnit [Jahre]	Gleich	0	0	0	0	2	0,214	0,579
	Ungleich	0	0	0	0	0	0	0
mit Java [Jahre]	Gleich	0	0	0	0,438	2	0,342	0,6
	Ungleich	0	0	0	0,125	4	0,406	1,02

Tabelle B.2: Lage- und Streuungsmaße der zu den Vorkenntnissen der Teilnehmer erhobenen Metriken.

Metrik	*p-Wert*
Kategorie: Produktivität	
Bearbeitungszeit [min]	0,524
Netto-Änderungen [SLOC]	0,536
Anteil der Netto-Änderungen im Testcode [%]	0,463
Anzahl der geänderten Dateien	0,334
Kategorie: Qualität	
Änderung der Anzahl der Testfälle	0,953
Änderung der Anweisungsüberdeckung [%]	0,012*
Anzahl der Akzeptanztestläufe	0,027*
Kategorie: TGE-Prozess	
TGE-Prozesstreue [%]	0,281
Anzahl der automatischen Umstrukturierungen	0,906
Anzahl der Testläufe	0,082
Mittlerer Abstand der Testläufe [min]	0,232
Standardabweichung des Abstands der Testläufe [min]	0,189
Variationskoeffizient des Abstands der Testläufe [%]	0,867

* signifikant auf dem 5 Prozent-Niveau

Tabelle B.3: *p*-Werte der zum Vergleich der Metriken zu TGE-Prozess, Produktivität und Qualität durchgeführten Wilcoxon-Tests

Metrik	*Gruppe*	*Min.*	*1. Qrt.*	*Median*	*3. Qrt.*	*Max.*	*Mittel.*	*Std. abw.*
Kategorie: Produktivität								
Bearbeitungszeit [min]	Gleich	90	103,5	116	139,5	162	122	26,752
	Ungleich	86	117,25	119	155,5	214	135,625	39,326
Netto-Änderungen [SLOC]	Gleich	365	501	547	576,5	768	547,857	123,014
	Ungleich	440	459,5	492,5	543,25	787	534,375	121,01
Anteil der Netto-Änd.	Gleich	61,091	62,741	65,814	70,634	74,609	66,895	5,196
im Testcode [%]	Ungleich	51,467	57,989	62,908	69,427	79,924	63,491	9,605
Anzahl der geänderten	Gleich	6	6	6	7,5	8	6,714	0,951
Dateien	Ungleich	6	6	6	6	14	7	2,828
Kategorie: Qualität								
Änderung der Anzahl	Gleich	−7	−4,5	0	4,5	11	0,571	6,451
der Testfälle	Ungleich	−4	−4	−2,5	1,5	11	0	5,632
Änderung der Anweisungs-	Gleich	−1,2	−0,65	−0,3	0,05	0,1	−0,371	0,489
überdeckung [%]	Ungleich	−0,4	0,1	0,2	0,625	1,6	0,437	0,67
Anzahl der Akzeptanz-	Gleich	1	1	1	1	2	1,143	0,378
testläufe	Ungleich	1	1,75	2	2	2	1,75	0,463
Kategorie: TGE-Prozess								
TGE-Prozesstreue [%]	Gleich	36,81	60,238	71,284	73,432	91,729	66,737	17,88
	Ungleich	34,307	65,935	86,842	92,379	99,415	76,487	24,821
Anzahl der automatischen	Gleich	0	1	3	3,5	6	2,571	2,07
Umstrukturierungen	Ungleich	0	0,75	2	4	6	2,375	2,2
Anzahl der Testläufe	Gleich	16	23,5	33	39,5	65	34,286	16,59
	Ungleich	32	37,75	45,5	60,5	84	50,75	17,806
Mittlerer Abstand der	Gleich	2,247	2,984	3,473	4,344	5,316	3,67	1,174
Testläufe [min]	Ungleich	1,604	2,238	2,866	3,139	3,985	2,776	0,79
Standardabweichung des	Gleich	3,711	4,454	4,9	6,224	6,896	5,266	1,197
Abstands der Testläufe [min]	Ungleich	1,988	3,263	4,063	5,044	6,606	4,205	1,468
Variationskoeffizient des	Gleich	118,325	134,898	137,763	162,107	188,561	148,38	26,753
Abstands der Testläufe [%]	Ungleich	105,658	120,022	153,921	177,93	213,497	152,98	39,476

Tabelle B.4: Lage- und Streuungsmaße der zu Produktivität, Qualität und TGE-Prozess erhobenen Metriken.

Anhang C

Ergänzende Grafiken und Tabellen zu Kapitel 6

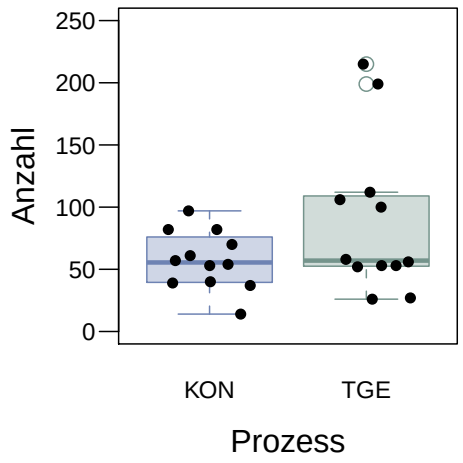


Abbildung C.1: Anzahl der Testläufe

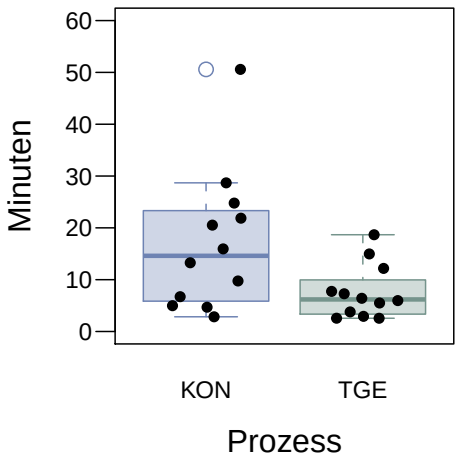


Abbildung C.2: Standardabweichung des Abstands der Testläufe

Metrik	*Min.*	*1. Qrt.*	*Median*	*3. Qrt.*	*Max.*	*Mittel.*	*Std. abw.*
Studiendauer [Jahre]	3	3	4,5	5	6	4,333	1,155
Programmiererfahrung ...							
privat [Jahre]	4	5	6	8	10	6,417	1,929
nebenberuflich [Jahre]	0	0,75	2,25	3	4	1,875	1,416
hauptberuflich [Jahre]	0	0	0	0	0	0	0
mit Java [Jahre]	1	3	4,5	5,25	8	4,375	2,247
mit Testautomatisierung [Jahre]	0	0	1	2,125	3	1,125	1,208
mit JUnit [Jahre]	0	0	0,5	2,125	3	1,042	1,252
mit testgetriebener Entwicklung [Jahre]	0	0	0	0,5	3	0,583	1,084
mit konventioneller Entwicklung [Jahre]	0	1,75	4	5,25	8	3,667	2,387

Tabelle C.1: Lage- und Streuungsmaße der zu den Vorkenntnissen der Teilnehmer erhobenen Metriken.

Metrik	*p-Wert*		
	KON vs. TGE	*S532 vs. VVS*	*Tag 1 vs. Tag 2*
Kategorie: TGE-Prozess			
TGE-Prozesstreue [%]	$< 0{,}001^*$	0,242	0,63
Anzahl der automatischen Umstrukturierungen	0,122	0,954	0,884
Anzahl der Testläufe	0,37	0,133	0,26
Mittlerer Abstand der Testläufe [min]	0,16	0,114	0,843
Standardabweichung des Abstands der Testläufe [min]	$0{,}045^*$	0,291	0,755
Variationskoeffizient des Abstands der Testläufe [%]	$0{,}033^*$	0,671	0,198
Kategorie: Produktivität			
Bearbeitungszeit [min]	0,977	1	$0{,}024^*$
Netto-Änderungen [SLOC]	0,178	0,06	0,41
Anteil der Netto-Änderungen im Testcode [%]	$0{,}039^*$	0,319	0,551
Anzahl der geänderten Dateien	0,557	$< 0{,}001^*$	0,557
Kategorie: Qualität			
Änderung der Anzahl der Testfälle	0,324	1	0,417
Änderung der Anweisungsüberdeckung [%]	0,908	$0{,}001^*$	0,488
Anzahl der Akzeptanztestläufe	0,336	0,904	0,976

* signifikant auf dem 5 Prozent-Niveau

Tabelle C.2: *p*-Werte der zum Vergleich der Metriken zu TGE-Prozess, Produktivität und Qualität durchgeführten Wilcoxon-Tests.

Metrik	*Proz.*	*Min.*	*1. Qrt.*	*Median*	*3. Qrt.*	*Max.*	*Mittel.*	*Std. abw.*
Kategorie: TGE-Prozess								
TGE-Prozesstreue [%]	KON	11,803	27,035	32,069	39,096	49,18	32,54	10,616
	TGE	21,267	57,77	69,514	76,032	88,889	65,24	20,033
Anzahl der automatischen	KON	0	1	1,5	2,25	24	3,5	6,599
Umstrukturierungen	TGE	0	1,75	5	8	15	5,75	5,065
Anzahl der Testläufe	KON	14	39,75	55,5	73	97	57,167	23,202
	TGE	26	52,75	57	107,5	215	88,083	62,325
Mittlerer Abstand der	KON	1,634	2,867	4,495	6,844	10,509	5,069	2,865
Testläufe [min]	TGE	1,246	2,063	2,706	4,6	7,613	3,646	2,389
Standardabweichung des	KON	2,841	6,284	14,603	22,596	50,586	17,053	13,589
Abstands der Testläufe [min]	TGE	2,556	3,566	6,189	8,85	18,671	7,546	5,166
Variationskoeffizient des	KON	155,877	218,219	277,248	374,091	561,625	311,019	133,611
Abstands der Testläufe [%]	TGE	144,708	171,288	205,358	235,609	260,906	205,42	39,046
Kategorie: Produktivität								
Bearbeitungszeit [min]	KON	132	184,75	215	282,75	459	248,25	104,3
	TGE	89	160,75	226	296	517	247,5	124,264
Netto-Änderungen [SLOC]	KON	141	192,75	232	315,25	566	274,5	129,321
	TGE	181	243,75	275	436,25	686	346	160,723
Anteil der Netto-Änd.	KON	18,44	28,731	32,207	40,931	54,462	34,988	11,491
im Testcode [%]	TGE	24,862	37,563	47,389	57,498	67,925	47,165	13,576
Anzahl der geänderten	KON	3	3	6,5	9	15	6,583	3,728
Dateien	TGE	3	4,5	7,5	10	11	7,083	3,088
Kategorie: Qualität								
Änderung der Anzahl	KON	3	5,75	9,5	13,5	26	10,833	6,965
der Testfälle	TGE	6	9	10	18,5	39	14,583	9,895
Änderung der Anweisungs-	KON	−6,1	−0,25	0,4	4,6	15,4	3,283	7,247
überdeckung [%]	TGE	−13	−1,7	3,1	9,2	12,4	2,633	8,084
Anzahl der Akzeptanz-	KON	1	2	2,5	3	5	2,583	1,165
testläufe	TGE	1	1	2	2,25	7	2,417	1,832

Tabelle C.3: Lage- und Streuungsmaße der zu Produktivität, Qualität und TGE-Prozess erhobenen Metriken.

Metrik	*Aufg.*	*Min.*	*1. Qrt.*	*Median*	*3. Qrt.*	*Max.*	*Mittel.*	*Std. abw.*
Kategorie: TGE-Prozess								
TGE-Prozesstreue [%]	S532	11,803	38,616	54,073	68,842	88,889	53,872	23,397
	VVS	19,57	27,035	36,362	58,8	84,762	43,908	22,263
Anzahl der automatischen	S532	0	1,75	2,5	4,5	14	3,75	3,98
Umstrukturierungen	VVS	0	1	1,5	8	24	5,5	7,379
Anzahl der Testläufe	S532	27	53,75	59,5	100,75	215	87,917	60,407
	VVS	14	38,5	53	73	106	57,333	28,017
Mittlerer Abstand der	S532	1,246	2,03	2,544	5,193	9,295	3,757	2,618
Testläufe [min]	VVS	1,766	2,986	3,974	7,481	10,509	4,958	2,717
Standardabweichung des	S532	2,568	3,566	6,189	14,615	50,586	11,933	14,046
Abstands der Testläufe [min]	VVS	2,556	6,841	10,965	17,089	28,687	12,666	7,941
Variationskoeffizient des	S532	155,877	174,618	244,972	275,879	544,251	259,945	109,717
Abstands der Testläufe [%]	VVS	144,708	200,415	215,523	262,09	561,625	256,494	115,822
Kategorie: Produktivität								
Bearbeitungszeit [min]	S532	89	153,75	217,5	359,25	459	254,833	124,849
	VVS	132	184,75	223,5	250,25	517	240,917	103,09
Netto-Änderungen [SLOC]	S532	141	180	225	269,25	686	281,917	172,239
	VVS	205	246,75	309	436,25	566	338,583	118,011
Anteil der Netto-Änd.	S532	18,44	26,554	34,682	50,408	65,017	38,314	14,918
im Testcode [%]	VVS	29,268	33,48	41,095	53,542	67,925	43,838	12,571
Anzahl der geänderten	S532	3	3	3	5,25	6	4	1,348
Dateien	VVS	7	9	9,5	10	15	9,667	2,06
Kategorie: Qualität								
Änderung der Anzahl	S532	3	8,5	10	12,75	39	13,25	10,244
der Testfälle	VVS	4	6	10	18,25	26	12,167	6,965
Änderung der Anweisungs-	S532	−0,7	3,8	9,5	12,85	15,4	8,342	5,816
überdeckung	VVS	−13	−4,375	−0,35	0,5	1,6	−2,425	4,587
Anzahl der Akzeptanz-	S532	1	1	2	3	7	2,583	1,832
testläufe	VVS	1	2	2	3	5	2,417	1,165

Tabelle C.4: Lage- und Streuungsmaße der zu Produktivität, Qualität und TGE-Prozess erhobenen Metriken (nach Aufgabe).

Metrik	*Tag*	*Min.*	*1. Qrt.*	*Median*	*3. Qrt.*	*Max.*	*Mittel.*	*Std. abw.*
Kategorie: TGE-Prozess								
TGE-Prozesstreue [%]	1	11,803	25,509	46,983	61,266	88,889	46,275	23,766
	2	27,072	33,183	39,518	71,306	87,626	51,505	22,748
Anzahl der automatischen	1	0	1	2	6,5	24	5,333	7,278
Umstrukturierungen	2	0	1	2,5	5,75	14	3,917	4,231
Anzahl der Testläufe	1	39	52,75	59	107,5	215	88,083	60,163
	2	14	34,5	55	82	100	57,167	28,342
Mittlerer Abstand der	1	1,246	2,587	3,03	4,83	7,693	3,935	2,127
Testläufe [min]	2	1,599	2,063	4,216	7,387	10,509	4,78	3,179
Standardabweichung des	1	2,924	5,852	7,002	16,358	28,687	11,518	8,642
Abstands der Testläufe [min]	2	2,556	3,545	9,96	16,624	50,586	13,081	13,588
Variationskoeffizient des	1	155,877	220,469	236,611	299,107	561,625	277,67	110,917
Abstands der Testläufe [%]	2	144,708	170,554	208,77	261,499	544,251	238,769	111,045
Kategorie: Produktivität								
Bearbeitungszeit [min]	1	162	218,5	254,5	342,5	517	285,25	102,522
	2	89	141	177,5	226,75	459	210,5	112,969
Netto-Änderungen [SLOC]	1	198	233	248,5	459,5	686	343,5	174,215
	2	141	180	273	315,25	479	277	112,166
Anteil der Netto-Änd.	1	29,268	33,249	36,809	52,26	67,925	42,942	13,938
im Testcode [%]	2	18,44	26,554	41,531	51,059	58,562	39,211	13,976
Anzahl der geänderten	1	3	5,5	6,5	9,25	15	7,333	3,473
Dateien	2	3	3	6	9,25	11	6,333	3,312
Kategorie: Qualität								
Änderung der Anzahl	1	4	6,75	10,5	21,5	39	15,25	10,864
der Testfälle	2	3	8,25	10	11,25	19	10,167	4,687
Änderung der Anweisungs-	1	−13	−2,725	0,15	11,5	15	2,292	9,274
überdeckung [%]	2	−3,8	0,375	1,4	8	15,4	3,625	5,578
Anzahl der Akzeptanz-	1	1	2	2	3	7	2,5	1,567
testläufe	2	1	1	2	3,25	5	2,5	1,508

Tabelle C.5: Lage- und Streuungsmaße der zu Produktivität, Qualität und TGE-Prozess erhobenen Metriken (nach Reihenfolge).

Anhang D

Verfügbarkeit der Studienmaterialien im Internet

Alle Komponenten von TestPulse sowie die Unterlagen und Daten zu den Studien dieser Arbeit stehen unter http://www.ipd.uka.de/~ahoefer/diss.zip zum Herunterladen bereit. Tabelle D.1 gibt einen Überblick über den Inhalt der Archivdatei *diss.zip*. Die Archivdatei enthält u. a. einen Ordner namens *TestPulse*. In diesem Ordner liegen die Quelltexte der Logger und des PostProzessors von TestPulse in mehreren Eclipse-Projekten vor. Die Eclipse-Projekte mit dem gemeinsamen Präfix *edu.kit.ipd.actionlogger* gehören dabei zu den Loggern. Das Eclipse-Projekt mit dem Namen *edu.kit.ipd.p2* enthält den in Java geschriebenen PostProzessor, das Eclipse-Projekt *edu.kit.ipd.p2.analysis* enthält die in R geschriebenen Analyseskripte, welche für die Berechnung der Prozessstatistiken und der Generierung der Grafiken zuständig sind (vgl. Abbildung 2.5 in Abschnitt 2.3.1).

Die Eclipse-Projekte mit dem gemeinsamen Präfix *edu.kit.ipd.actionlogger* und das Eclipse-Projekt mit dem Namen *edu.kit.ipd.p2* wurden zuletzt mit Eclipse in der Version 3.7.2 (Indigo) und dem JDK in der Version 1.6.0_30-12b erfolgreich übersetzt. Die Logger ließen sich anschließend auch in der zuvor genannten Eclipse-Version betreiben. Die R-Skripte im Eclipse-Projekt *edu.kit.ipd.p2.analysis* wurden mit R 2.15.1 auf Windows 7 professional 64 Bit erfolgreich ausgeführt. Zur Bearbeitung der R-Skripte innerhalb von Eclipse wurde zusätzlich das Plugin StatET[1] installiert.

Neben den Quelltexten von TestPulse finden sich in der Archivdatei drei Ordner namens *Chapter 4*, *5* und *6*, welche Material zu den Studien enthalten, die im entsprechend nummerierten Kapitel dieser Arbeit beschrieben sind. Dieses Material ist in jedem dieser Ordner auf drei weitere Unterordner verteilt. Der Unterordner *Code* enthält dabei Eclipse-Projekte mit den Quelltexten der Programmieraufgabe samt dazugehöriger Musterlösung.

Der Unterordner *Data* enthält eine SQL-Datei mit allen durch den PostProzessor von

[1] http://www.walware.de/goto/statet

diss.zip

- ▶ Chapter 4
  - ▶ Code
    - ▷ Elevator
    - ▷ Elevator_solution
  - ▶ Data
    - ▷ Analysis
    - ☐ NonP2Data.xlsx
    - ☐ Questionnaires.xlsx
    - ☐ xperiments_20110828.sql
  - ▶ Documents
    - ▷ Aufgabenbeschreibung
    - ▷ Information
    - ▷ NachtestFragebogen
    - ▷ TeilnehmerFragebogen
- ▷ Chapter 5
- ▷ Chapter 6
- ▶ TestPulse
  - ▷ edu.kit.ipd.actionlogger
  - ▷ edu.kit.ipd.actionlogger.core
  - ▷ edu.kit.ipd.actionlogger.junit
  - ▷ edu.kit.ipd.actionlogger.parser
  - ▷ edu.kit.ipd.actionlogger.refactoring
  - ▷ edu.kit.ipd.actionlogger.updatesite
  - ▷ edu.kit.ipd.actionlogger.useractions
  - ▷ edu.kit.ipd.actionlogger.xmlreencoder
  - ▷ edu.kit.ipd.p2
  - ▷ edu.kit.ipd.p2.analysis

▶ Ordner, geöffnet ▷ Ordner, geschlossen ☐ Datei

Tabelle D.1: Ordnerstruktur der Studienmaterialien.

TestPulse ausgewerteten Datensätzen. Diese Datei kann mit Unterstützung der MySQL-Workbench (Version 5.7.2 CE) in eine MySQL-Datenbank (Version 5.1.65 CE) importiert werden. Der im PostProzessor von TestPulse enthaltene Datenpunktbetrachter kann sich dann mit der Datenbank verbinden und erlaubt eine Begutachtung der Datensätze. Weiter enthält der Unterordner *Data* zwei Excel-Dateien, welche die manuell erfassten Daten zu den Experimentdatensätzen enthalten. Die Daten in der Datei *NonP2Data.xlsx* lassen sich, nachdem sie als CSV-Datei abspeichert wurden, mit den in *edu.kit.ipd.p2.analysis* vorhandenen R-Skripten auswerten. Zur Auswertung der Daten in *Questionnaire.xlsx* finden sich in dem daneben befindlichen Ordner *Analysis* spezielle R-Skripte.

Im letzten der drei Unterordner *Documents* finden sich sämtliche zu einer Studie gehörigen Schriftstücke als LaTeX-Quelltexte wieder.

Abbildungsverzeichnis

Tabellenverzeichnis

Quellen- und Literaturverzeichnis

[AGDS07] ARISHOLM, Erik ; GALLIS, Hans ; DYBÅ, Tore ; SJØBERG, Dag I. K.: Evaluating Pair Programming with Respect to System Complexity and Programmer Expertise. In: *IEEE Transactions on Software Engineering* Jhg. 33 (2007), Februar, Nr. 2, S. 65–86. – DOI 10.1109/TSE.2007.17 – ISSN 0098–5589

[AS04] ARISHOLM, Eric ; SJØBERG, Dag I. K.: Evaluating the Effect of a Delegated versus Centralized Control Style on the Maintainability of Object-Oriented Software. In: *IEEE Transactions on Software Engineering* Jhg. 30 (2004), August, Nr. 8, S. 521–534. – DOI 10.1109/TSE.2004.43 – ISSN 0098–5589

[Bec00] BECK, Kent: *Extreme Programming Explained: Embrace Change*. 1. Ausg. Reading, Massachusetts, USA : Addison-Wesley, 2000. – ISBN 0–201–61641–6

[Bec02] BECK, Kent: *Test Driven Development: By Example*. Amsterdam : Addison-Wesley, Longman, 2002. – ISBN 978–0321146533

[Bec05] BECK, Kent: *Extreme Programming Explained: Embrace Change*. 2. Ausg. Reading, Massachusetts, USA : Addison-Wesley, 2005. – ISBN 0–321–27865–8

[BN06] BHAT, Thirumalesh ; NAGAPPAN, Nachiappan: Evaluating the Efficacy of Test-Driven Development: Industrial Case Studies. In: *ISESE '06: Proceedings of the 2006 ACM/IEEE international symposium on empirical software engineering*. New York, NY, USA : ACM Press, 2006. – ISBN 1–59593–218–6, S. 356–363

[BN07] BEGEL, Andrew ; NAGAPPAN, Nachiappan: Usage and Perceptions of Agile Software Development in an Industrial Context: An Exploratory Study. In: *ESEM '07: Proceedings of the First International Symposium on Empirical Software Engineering and Measurement*. Washington, DC, USA : IEEE Computer Society, 2007. – ISBN 0–7695–2886–4, S. 255–264

[Bry04] BRYANT, Sallyann: Double Trouble: Mixing Qualitative and Quantitative Methods in the Study of eXtreme Programmers. In: *2004 IEEE Symposium on Visual Languages and Human Centric Computing*, 2004, S. 55–61

[BT94] BÜNING, Herbert ; TRENKLER, Götz: *Nichtparametrische statistische Methoden*. 2. Ausg. de Gruyter, 1994. – ISBN 3–11–014105–1

[CCG+06] CANFORA, Gerardo ; CIMITILE, Aniello ; GARCIA, Felix ; PIATTINI, Mario ; VISAGGIO, Corrado A.: Evaluating Advantages of Test Driven Development: a Controlled Experiment with Professionals. In: *ISESE '06: Proceedings of the 2006 ACM/IEEE International Symposium on Empirical Software Engineering*. New York, NY, USA : ACM Press, 2006. – ISBN 1–59593–218–6, S. 364–371

[CH07] CHONG, Jan ; HURLBUTT, Tom: The Social Dynamics of Pair Programming. In: *ICSE '07: Proceedings of the 29th International Conference on Software Engineering*. Washington, DC, USA : IEEE Computer Society, 2007. – ISBN 0–7695–2828–7, S. 354–363

[Chr07] CHRISTENSEN, Larry B.: *Experimental methodology*. 10. Ausg. Pearson/Allyn and Bacon, 2007. – ISBN 0–205–48473–5

[Coh88] COHEN, Jacob: *Statistical Power Analysis for the Behavioral Sciences*. 2. Ausg. Lawrence Erlbaum Associates, Inc., 1988. – ISBN 0–8058–0283–5

[DAS+07] DYBÅ, Tore ; ARISHOLM, Erik ; SJØBERG, Dag I. ; HANNAY, Jo E. ; SHULL, Forrest: Are Two Heads Better than One? On the Effectiveness of Pair Programming. In: *IEEE Software* Jhg. 24 (2007), November/Dezember, Nr. 6, S. 12–15. – DOI 10.1109/MS.2007.158 – ISSN 0740–7459. – List of studies: http://www2.computer.org/cms/Computer.org/dl/mags/so/2007/06/extras/mso2007060012x1.html

[Ecl] ECLEMMA: *Mountainminds GmbH & Co. KG*. http://www.eclemma.org

[Edw03] EDWARDS, Stephen H.: Using Test-Driven Development in the Classroom: Providing Students with Automatic, Concrete Feedback on Performance. In: *Proceedings of the International Conference on Education and Information Systems: Technologies and Applications (EISTA'03), International Institute of Informatics and Systemics*, 2003, S. 421–426

[EMT05] ERDOGMUS, Hakan ; MORISIO, Maurizio ; TORCHIANO, Marco: On the Effectiveness of the Test-First Approach to Programming. In: *IEEE Transactions on Software Engineering* Jhg. 31 (2005), März, Nr. 3, S. 226–237. – DOI 10.1109/TSE.2005.37

[For49] FORER, Bertram R.: The Fallacy of Personal Validation: A Classroom Demonstration of Gullibility. In: *Journal of Abnormal Psychology* Jhg. 44 (1949), S. 118–121

[Fow99] FOWLER, Martin: *Refactoring – Improving the Design of Existing Code*. Reading, Massachusetts, USA : Addison-Wesley, 1999. – ISBN 0–201–48567–2

[Gol92] GOLDBERG, Lewis R.: The Development of Markers for the Big-Five Factor Structure. In: *Psychological Assessment* Jhg. 4 (1992), Nr. 1, S. 26–42

[GSM04] GERAS, A. ; SMITH, M. ; MILLER, J.: A Prototype Empirical Evaluation of Test Driven Development. In: *METRICS '04: Proceedings of the 10th International Symposium on Software Metrics*. Washington, DC, USA : IEEE Computer Society, 2004. – ISBN 0–7695–2129–0, S. 405–416

[GW04] GEORGE, Boby ; WILLIAMS, Laurie A.: A Structured Experiment of Test-Driven Development. In: *Information and Software Technology* Jhg. 46 (2004), Nr. 5, S. 337–342. http://www.cse.dmu.ac.uk/~ieb/Test%20driven%20development.pdf

[Hac] HACKYSTAT: http://www.hackystat.org

[HAES10] HANNAY, Jo E. ; ARISHOLM, Erik ; ENGVIK, Harald ; SJØBERG, Dag I. K.: Effects of Personality on Pair Programming. In: *IEEE Transactions on Software Engineering* Jhg. 36 (2010), Januar/Februar, Nr. 1, S. 61–80. – DOI 10.1109/TSE.2009.41 – ISSN 0098–5589

[HDC88] HOLLAND, Burt S. ; DIPONZIO COPENHAVER, Margaret: Improved Bonferroni-Type Multiple Testing Procedure. In: *Psychological Bulletin* Jhg. 104 (1988), Nr. 1, S. 145–149. http://www.usq.edu.au/users/patrick/PAPERS/bonferroni%201.pdf. – ISSN 0033–290

[Höf08] HÖFER, Andreas: Video Analysis of Pair Programming. In: *APSO '08: Proceedings of the 2008 International Workshop on Scrutinizing Agile Practices or Shoot-out at the Agile Corral*. New York, NY, USA : ACM, 2008. – ISBN 978–1–60558–021–0, S. 37–41

[Höf10] HÖFER, Andreas: Exploratory Comparison of Expert and Novice Pair Programmers. In: *Computing and Informatics* Jhg. 29 (2010), Nr. 1, S. 73–91

[How99] HOWELL, David C.: *Fundamental Statistics for the Behavioral Sciences*. 4. Ausg. Brooks/Cole Publishing Company, 1999. – ISBN 0–534–35821–7

[HP09] HÖFER, Andreas ; PHILIPP, Marc: An Empirical Study on the TDD Conformance of Novice and Expert Pair Programmers. In: *Agile Processes in Software Engineering and Extreme Programming* Bd. 31, Springer Berlin Heidelberg, Mai 2009 (Lecture Notes in Business Information Processing). – ISBN 978–3–642–01852–7, S. 33–42

[HSQ] HYPERSQL DATABASE: http://hsqldb.org

[HW99] HOLLANDER, Myles ; WOLFE, Douglas A.: *Nonparametric Statistical Methods*. 2. Ausg. Wiley Interscience, 1999. – ISBN 0–471–19045–4

[Jal] JALOPY: *kommerzielle Version*. http://www.triemax.com/products/jalopy/

[JK07] JOHNSON, Philip M. ; KOU, Hongbing: Automated Recognition of Test-Driven Development with Zorro. In: *AGILE '07: Proceedings of the AGILE 2007*, IEEE Computer Society, August 2007. – ISBN 0–7695–2872–4, S. 15–25

[JUn] JUNIT: http://www.junit.org

[KB90] KEIRSEY, David ; BATES, Marilyn: *Versteh mich bitte*. 1. Ausg. Prometheus Nemesis Book Co., 1990. – ISBN 0–960–69544–3

[Kei98] KEIRSEY, David: *Please understand me II*. 2. Ausg. Prometheus Nemesis Book Co., 1998. – ISBN 1–885705–02–6

[KJ06] KOU, Hongbing ; JOHNSON, Philip M.: Automated Recognition of Low-Level Process: A Pilot Validation Study of Zorro for Test-Driven Development. In: *Software Process Change* Bd. 3966/2006. Berlin/Heidelberg : Springer, Juli 2006 (Lecture Notes in Computer Science). – ISBN 978–3–540–34199–4, S. 322–333

[KJE10] KOU, Hongbing ; JOHNSON, Philip M. ; ERDOGMUS, Hakan: Operational Definition and Automated Inference of Test-Driven Development with Zorro. In: *Automated Software Engineering* Jhg. 17 (2010), März, Nr. 1, S. 57–85. – DOI 10.1007/s10515–009–0058–8

[Kou07] KOU, Hongbing: *Automated Inference of Software Development Behaviors: Design, Implementation and Validation of Zorro for Test-Driven Development*, University of Hawaii, Dissertation, Dezember 2007. http://csdl.ics.hawaii.edu/techreports/07-04/07-04.pdf

[Leo07] LEONHARDT, Dominik: Entwicklung und Evaluierung eines Refactoring-Plugins für Eclipse / Universität Karlsruhe (TH), Institut für Programmstrukturen und Datenorganisation (IPD). 2007. – Forschungsbericht

[Lin05] LINK, Johannes: *Softwaretests mit JUnit*. 2. Ausg. dpunkt.verlag, 2005 http://stmj.developertests.de/portal. – ISBN 3–89864–325–5

[Mar07] MARMÉ, Florian: Entwicklung einer Aufgabe zur empirischen Untersuchung der testgetriebenen Entwicklung / Universität Karlsruhe (TH), Institut für Programmstrukturen und Datenorganisation (IPD). 2007. – Forschungsbericht

[MBM95] MYERS-BRIGGS, Isabel ; MYERS, Peter B.: *Gifts Differing: Understanding Personality Type*. 2. Ausg. Davies-Black Publishing, 1995. – 248 S. – ISBN 978–0891060741

[MFC01] *Kapitel* Endo-testing: Unit Testing With Mock Objects. In: MACKINNON, Tim ; FREEMAN, Steve ; CRAIG, Philip: *Extreme Programming Examined*. Boston, MA, USA : Addison-Wesley Longman Publishing Co., Inc., 2001. – ISBN 0–201–71040–4, 287–301

[MH02] MÜLLER, Matthias M. ; HAGNER, Oliver: Experiment about Test-First Programming. In: *IEE Proceedings – Software* Bd. 149, 2002, S. 131–136

[MH07] MÜLLER, Matthias M. ; HÖFER, Andreas: The Effect of Experience on the Test-Driven Development Process. In: *Empirical Software Engineering* Jhg. 12 (2007), Dezember, Nr. 6, S. 593–615. – DOI 10.1007/s10664–007–9048–2

[MW03] MAXIMILIEN, E. M. ; WILLIAMS, Laurie A.: Assessing Test-Driven Development at IBM. In: *ICSE '03: Proceedings of the 25th International Conference on Software Engineering*. Washington, DC, USA : IEEE Computer Society, 2003. – ISBN 0–7695–1877–X, S. 564–569

[PCTV03] PANČUR, Matjaž ; CIGLARIČ, Mojca ; TRAMPUŠ, Matej ; VIDMAR, Tone: Towards Empirical Evaluation of Test-Driven Development in a University

Environment. In: *EUROCON 2003. Computer as a Tool. The IEEE Region 8.* Bd. 2, 2003, S. 83–86

[Phi08] PHILIPP, Marc: *Comparison of the Test-Driven Development Processes of Novice and Expert Programmer Pairs*, Universität Karlsruhe (TH), Institut für Programmstrukturen und Datenorganisation (IPD), Diplomarbeit, September 2008

[R] R: *The R Project for Statistical Computing*. http://www.r-project.org. – Eingesetzte Version: 2.11.1

[Sau94] SAUCIER, Gerard: Mini Markers: A Brief Version of Goldberg's Unipolar Big-Five Markers. In: *Journal of Personality Assessment* Jhg. 63 (1994), Nr. 3, S. 506–516. http://pages.uoregon.edu/prsnlty/SAUCIER/Saucier.Minimarkers.Full.pdf

[SCC02] SHADISH, William R. ; COOK, Thomas D. ; CAMPBELL, Donald T.: *Experimental and Quasi-Experimental Designs for Generalized Causal Inference*. 1. Ausg. Houghton Mifflin, 2002. – ISBN 0–395–61556–9

[Sch09] SCHNEIDER, Marcel: Vorbereitung der Eclipse-Plug-ins zur Protokollierung der testgetriebenen Entwicklung auf den Einsatz in Feldstudien / Karlsruher Institut für Technologie (KIT), Institut für Programmstrukturen und Datenorganisation (IPD). 2009. – Forschungsbericht

[Sch10] SCHNEIDER, Marcel: *Vergleich der testgetriebenen Entwicklung mit der Entwicklung nach dem Wasserfallmodell*, Karlsruher Institut für Technologie (KIT), Institut für Programmstrukturen und Datenorganisation (IPD), Diplomarbeit, August 2010

[SSAD09] SFETSOS, Panagiotis ; STAMELOS, Ioannis ; ANGELIS, Lefteris ; DELIGIANNIS, Ignatios S.: An Experimental investigation of Personality Types Impact on Pair Effectiveness in Pair Programming. In: *Empirical Software Engineering* Jhg. 14 (2009), April, Nr. 2, S. 187–226. DOI 10.1007/s10664–008–9093–5. – ISSN 1382–3256

[WE04] WANG, Yihong ; ERDOGMUS, Hakan: The Role of Process Measurement in Test-Driven Development. In: *Extreme Programming and Agile Methods – XP/Agile Universe 2004, Proceedings on the 4th Conference on Extreme Programming and Agile Methods* Bd. 3134, Springer, August 2004 (Lecture Notes in Computer Science), S. 32–42

[Weg04] WEGE, Christian: *Automated Support for Process Assessment in Test-Driven Development*, Eberhard-Karls-Universität Tübingen, Dissertation, 2004. http://w210.ub.uni-tuebingen.de/dbt/volltexte/2004/1392/pdf/thesis.pdf