Karlsruhe Institute of Technology

Fallstudie zur Verbesserung halbautomatisierter Testerzeugung

Diplomarbeit
von

Steffen Börsig

An der Fakultät für Informatik
Institut für Programmstrukturen
und Datenorganisation (IPD)

Erstgutachter:	Prof. Dr. Walter F. Tichy
Zweigutachter:	Prof. Dr. Ralf H. Reussner
Betreuender Mitarbeiter:	Dipl. Inform.-Wirt Mathias Landhäußer

Bearbeitungszeit: 28.08.2013 – 31.03.2014

Ich versichere wahrheitsgemäß, die Arbeit selbstständig angefertigt, alle benutzten Hilfsmittel vollständig und genau angegeben und alles kenntlich gemacht zu haben, was aus Arbeiten anderer unverändert oder mit Abänderungen entnommen wurde.

Die Regeln zur Sicherung guter wissenschaftlicher Praxis im Karlsruher Institut für Technologie (KIT) habe ich befolgt.

Karlsruhe, 31.03.2014

. . . . . .

(**Steffen Börsig**)

Kurzfassung

Akzeptanztests bieten durch die Möglichkeit natürlichsprachliche Tests zu formulieren eine große Chance für Softwareentwickler, Anforderungen besser mit den Kunden abzusprechen.

Die Firma Agilent verwendet seit einigen Jahren Akzeptanztests um die Qualität ihrer Produkte sicherzustellen. Im Laufe der Zeit haben sich mehrere tausend einzelne Testschritte in der natürlichen Sprache angesammelt, deren Wiederverwendung eine sehr gute Werkzeugunterstützung erfordert. Zusammen mit zwei Expertenteams bestehend aus drei Chemikern und neun Entwicklern wurden im Zuge einer Fallstudie Probleme identifiziert und mögliche Lösungsansätze entwickelt. Das Ergebnis war die Weiterentwicklung des Visual Studio Plugin SpecFlow, um einen schnelleren und effizienteren Einsatz der bereits vorhandenen Autovervollständigung zu gewährleisten.

Die Auswertung der Ergebnisse erfolgte über Interviews und eine statistische Analyse des Versionsverwaltungssystems. Dabei konnte qualitativ festgestellt werden, dass kleinere und individuelle Änderungen von den Probanden leichter angenommen werden und eine gefühlte Verbesserung der Testerstellung auftritt. Die quantitative Analyse, die auf den durchschnittlichen Änderungen der Testskripte nach dem initialen Eintragen in das Versionsverwaltungssystems aufbaut, konnte leider kein signifikantes Ergebnis belegen. Eine schlichte Analyse aller benötigten Änderungen zeigt auf, dass fast 10% weniger Änderungen nach der Einführung des Plugins nötig sind. In einem entsprechenden Student's t-Test konnte jedoch nur mit einem $\alpha = 0,17$ die Nullhypothese verworfen werden. Die Datenbasis ist aufgrund verschiedener Probleme leider noch sehr klein.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Quelltextverzeichnis

1. Einleitung

1.1. Motivation

Spätestens seit der Veröffentlichung von Kent Beck's „Extreme Programming Explained: Embrace Change"[Bec10] und der Entwicklung von xUnit-Test, ist automatisiertes Testen zentraler Bestandteil der Softwareentwicklung. Beim Testen ist mittlerweile allerdings weniger wichtig, welche Technik genau eingesetzt wird und ob die Tests davor oder danach geschrieben werden. Der momentane Trend der meisten Programmierrichtungen geht dabei hin zu einer höheren Testabdeckung. Die Vorteile liegen dabei auf der Hand. Besser getestete Software beinhaltet allgemein weniger Fehler im Code und hat damit verbunden geringere Wartungskosten. Ebenso wird der Entwickler entlastet bei der Entscheidung, bestimmte Codestellen zu verändern, da er mit Hilfe der Tests eine Kontrollinstanz hat, die neu eingebaute Fehler schnell entdecken können. So gut eine hohe Testabdeckung ist, sie hilft nicht dabei sicherzustellen, dass tatsächlich das richtige getestet wird. Sie stellt lediglich fest, dass etwas getestet wird. Die Sicherstellung, dass der Code den Wünschen des Kunden entspricht muss weiterhin außerhalb der Tests geschehen.

Hier setzt die relativ junge Technik Behavior Driven Development (BDD) an. Dieser Ansatz treibt Test Driven Development (TDD) noch ein Stück höher in der Abstraktionsebene und verlangt vom Entwickler ein direktes Auseinandersetzen mit der Erfüllung der Anforderungen des Anwenders. Die Anforderungsanalyse spielt hierbei natürlich eine entscheidende Rolle. So werden die Akzeptanztests zumeist mit oder im Idealfall direkt von den Anwendern entwickelt. Dies ist dadurch möglich, dass die Akzeptanztests in der natürlichen Sprache formuliert werden.

Cucumber[WH12], eine Rahmenarchitektur zur Entwicklung mit BDD ist eine der Möglichkeiten diese natürlichsprachlichen Tests zu schreiben. Alle Tests bestehen aus mindestens drei Schritten: „Gegeben..." als Aufbau für den Test. „Wenn..." als Aktion die ausgeführt werden soll. Und dem letzten Schritt „Dann ...", der die Überprüfung des Resultats beschreibt. Diese simple Vorgabe ermöglicht es, dass auch weniger versierte Anwender die notwendigen Beispiele formulieren können, mit denen die Akzeptanz sichergestellt werden soll.

„Gegeben das ich ein falsches Passwort eingebe, Wenn ich auf den Knopf „Login" klicke, Dann bekomme ich eine Fehlermeldung". So oder ähnlich könnte dabei ein Test aussehen. Die drei Schritte werden dann im Test verarbeitet. Diese einfache Möglichkeit Testfälle zu beschreiben ist auf der einen Seite sehr mächtig, auf der anderen Seite ergeben sich in

der dauerhaften Anwendung des Konzepts einige Probleme. Die Ungenauigkeit der Testbeschreibung führt dazu, dass die syntaktische Formulierung sich in neueren Tests von den alten unterscheidet, obwohl kein semantischer Unterschied existiert. Als Beispiel könnte ein zweiter Testfall folgendermaßen aussehen: „Gegeben, dass ich ein richtiges Passwort eingebe, Wenn ich auf den Button „Login" klicke, Dann bekomme ich eine Willkommensmeldung". In diesem Fall müssten eigentlich nur der erste und der letzte Schritt von den Entwicklern neu implementiert werden. Der zweite Schritt, der die Aktion „Button/Knopf klicken" enthält, sollte wiederverwendbar sein. Durch die unglückliche Vertauschung von „Knopf" und „Button" wird der zweite Schritt bei einer standardmäßigen Mustererkennung nicht automatisch erkannt. Eine Nachbearbeitung durch einen Entwickler ist notwendig, um syntaktische Gleichheit herzustellen.

Mit dieser Problematik werden Unternehmen die mit BDD entwickeln früher oder später konfrontiert, da die Menge an Testschritten unüberschaubar für den Ersteller wird. Dies führt zu unnötiger Mehrarbeit und mindert damit den Nutzen der Akzeptanztests. Das Entwicklungsteam der Firma Agilent ist an diesem Punkt angekommen. Chemiker entwickeln hier eigenständig Akzeptanztests und die Entwickler müssen diese oftmals in hohem Maße anpassen, damit bereits vorhandene Schritte wiederverwendet werden. Ein vielversprechender Ansatz für die Lösung dieses Problems ist die Einführung eines Vorschlagsystems für die Testschreiber. Ähnlich der halbautomatisierten Codevervollständigung sollen die bereits existierenden Testschritte im System beim Erstellungszeitpunkt dem Anwender vorgeschlagen werden. Dies verspricht eine höhere Wiederverwendung der einzelnen Testschritte.

1.2. Industrie

1.2.1. Agilent Technologies GmbH

Die Fallstudie findet in Zusammenarbeit mit der Agilent Technologies GmbH statt. Agilent Technologies oder auch abgekürzt Agilent stellen elektronische und bioanalytische Messgeräte her. Dabei erfolgt im Haus sowohl der Entwurf, die Entwicklung, als auch die Herstellung der Hauptkomponenten. Der ursprünglich aus einigen Hewlett-Packard Abteilungen entstandene Konzern, vertreibt heutzutage eine große Produktpalette aus unter anderem Oszilloskopen, Signalgneratoren, Massenspektrometern und Chromatographiegeräten.

Das im Jahre 1999 gegründete Unternehmen hat seinen Hauptsitz in Santa Clara, Kalifornien (USA). Standorte von Agilent sind über die ganze Welt verteilt. Beispiele dafür sind die USA, Deutschland, Frankreich, China, Japan und Singapur. Mit 20.600 Mitarbeitern gehört Agilent zu einem der größeren Unternehmen im Silicon Valley.

In Deutschland beschäftigt die Agilent Technologies GmbH etwa 1.400 Menschen. Dabei sind die meisten Mitarbeiter auf die Standorte Böblingen und Waldbronn verteilt. In Böblingen werden dabei optische und hochfrequente Netzwerkelemente entwickelt und hergestellt. In Waldbronn, welches zur Divison „Life Sciences und Chemische Analysetechnik" gehört, werden hauptsächlich Flüssigkeitschromatographen entwickelt und produziert. Diese erlauben die Spaltung der Elemente in Nano-Mengen, um diese nachzuweisen und zu analysieren. Der Einsatz erfolgt vor allem in der pharmazeutischen und in der chemischen Industrie. Forschungseinrichtungen nutzen die Chromatographen ebenso zur Analyse.

1.2.2. Beteiligte Personen

Die Koordination und hauptsächliche Zusammenarbeit in dieser Fallstudie findet mit den Mitarbeitern am Standort Waldbronn statt. Um mehr Personen für die Studie zu erhalten

wird zusätzlich Personal aus dem Standort Grenoble in Frankreich mit einbezogen. Insgesamt sind über beide Teams verteilt drei Chemiker beteiligt. Bei den Softwareentwicklern sind es insgesamt neun Personen.

1.3. Ziel

Das Ziel der Diplomarbeit ist es, dass vorhandene Vorschlagsystem weiter zu entwickeln und bei der Firma Agilent einzuführen. Der Fokus liegt dabei auf der empirischen Auswertung des Einführungsprozesses und der darauf folgenden Nutzungsphase. Die zentralen Fragen lauten hierbei:

1. Wie wirken sich kleine Verbesserungen eines Vorschlagsystems zur halbautomatisierten Testerzeugung auf das Entwicklungsteam aus?
2. Wie viel Nachbearbeitung der Tests kann eingespart werden, wenn das System an die Bedürfnisse vor Ort zugeschnitten wird?
3. Wie gut werden die Anwender durch die Verbesserungen in der Testerstellung unterstützt?

Vorgehen

Um das zuvor gesteckte Ziel zu erreichen wird eine Fallstudie bei Agilent durchgeführt. Als Methodik der empirischen Forschung bietet sich die Fallstudie im Vergleich zu allen anderen Methodiken in dieser Situation am besten an. Die Begründung dieser Entscheidung wird nach der Einführung der empirischen Literatur gegeben.

Die Hauptschritte in der Studie sind:

- Die Durchführung der Interviews und die Analyse der bestehenden Tests vor der Einführung des Systems.
- Die erfolgreiche Implementierung und Einführung des Vorschlagsystems vor Ort.
- Die Auswertung der Nutzungsphase mit anschließenden Interviews und Analyse der Tests.

1.4. Gliederung

Im Folgenden wird zunächst Scrum erklärt, da die Teams von Agilent damit arbeiten. Danach wird auf die Theorie hinter BDD und die verwendeten Rahmenarchitekturen und Werkzeuge eingegangen. Im Anschluss daran werden die empirischen Grundlagen zum Verständnis einer Fallstudie aufbereitet.

Danach wird auf die eigentliche Fallstudie eingegangen bei dem die Planung und Durchführung genauer beschrieben wird. Vor der Auswertung wird noch die Implementierung der Verbesserungen beschrieben und die Arbeit endet mit der Zusammenfassung und gibt einen kleinen Ausblick für zukünftige Forschungsansätze.

2. Grundlagen

2.1. Scrum

Die Definition von Scrum laut scrum.org ist folgende.[SS13]

```
Scrum (n): A framework within which people can address complex
adaptive problems, while productively and creatively delivering products
of the highest possible value.
```

Scrum beschreibt demnach einen Rahmen in dem verschiedenste Techniken eingesetzt werden können, um Produkte für komplexe und adaptive Aufgaben zu entwickeln. Scrum versteht sich dabei selbst als leichtgewichtigen Prozess der einfach zu verstehen ist.

Scrum besteht im wesentlichen aus vier Komponenten:

- Das Scrum-Team
- Den Scrum-Ereignissen
- Den Scrum-Artefakten
- Den Scrum-Regeln

In den folgenden Unterkapiteln werden die englischen Begriffe von Scrum verwendet da die Verwendung deutscher Übersetzungen in diesem Umfeld nicht üblich ist.

2.1.1. Scrum-Team

Das komplette Scrum Team besteht aus zwei Einzelpersonen, dem Product-Owner und dem Scrum-Master und dem Develeopment-Team das üblicherweise aus mehreren Personen besteht.[SS13][Glo13, S. 61] Das gesamte Team sollte dabei so gewählt werden, dass es die Projekte ohne weitere Hilfe von außen abschließen kann. Das heißt, dass alle notwendigen Kompetenzen von einer oder mehreren Personen im Team abgedeckt werden.[Kni10, S. 27] In den meisten Fällen heißt dies auch, dass die Individuen im Team mehrere Aufgaben übernehmen können die nicht ihrer eigentlichen Ausbildung angehören. Diese Eigenschaft wird im agilen Umfeld auch als „cross-functional" bezeichnet. Scrum-Teams müssen zusätzlich selbst organisierend sein. Das heißt es wird innerhalb des Teams entschieden, welche Aufgaben wann und wie erledigt werden.

Wie in agilen Methoden üblich, wird auch in Scrum versucht möglichst viel Feedback in den Entwicklungsprozess einfließen zu lassen. Um dies sicherzustellen wird in Scrum iterativ entwickelt und in möglichst kurzen Abständen, das heißt wenigen Wochen, ein lauffähiges Produkt fertiggestellt. Um dies zu erreichen, müssen die Teammitglieder verschiedene Aufgaben erledigen die im Folgenden erklärt werden.

2.1.1.1. Product-Owner

Der Product-Owner ist hauptsächlich dafür verantwortlich die Prioritäten für das Development Team zu setzen. [SS13] Als Hilfsmittel dafür dient das Product-Backlog das in den Artefakten noch einmal genauer vorgestellt wird. Einfach ausgedrückt verwendet der Product-Owner das Product-Backlog um die gröberen Ziele für das Produkt aufzuschreiben. Sie sollten klar formuliert werden und so angeordnet werden, dass die wichtigsten Ziele zuerst erreicht werden, da die Ordnung die Priorität für das Development-Team bestimmt. Damit bestimmt das Product-Backlog zu jedem Zeitpunkt an welchen Aufgaben das Team im nächsten Schritt arbeiten wird und es ist damit von hohem Wert die einzelnen Punkte so klar wie möglich zu kommunizieren damit das Team die Ziele versteht. Dies ist die wichtigste Aufgabe des Product-Owners innerhalb des Scrum-Teams.

Zum Management und den restlichen Interessenvertretern hin nimmt er zusätzlich eine Schnittstellenfunktion ein. Alle Anfragen an das Projekt werden über den Product-Owner geleitet und die Verantwortung liegt bei ihm diese in den Projektplan als Punkte im Product-Backlog aufzunehmen. Die Priorität der Einträge legt auch in einem solchen Fall allein der Product-Owner fest.

2.1.1.2. Scrum-Master

Der Scrum-Master ist ähnlich wie der Product-Owner meist nicht an der eigentlichen Produktentwicklung beteiligt. Viel mehr nimmt der Scrum-Master eine beratende Position ein. Er ist dafür verantwortlich, dass sowohl das Development-Team als auch andere innerhalb der Organisation die Abläufe von Scrum verstehen. Er übernimmt dabei für die unterschiedlichen Beteiligten verschiedene Aufgaben.

Der Scrum-Master hilft dem Product-Owner dabei die Einträge im Product-Backlog so klar wie möglich zu kommunizieren und diese auch effektiv zu verwalten. Er vermittelt dabei sowohl die theoretischen Überlegungen hinter Scrum als auch die praktischen Arbeitsweisen mit dem Product-Backlog. Zusätzlich dazu übernimmt er eine organisatorische Rolle indem er Treffen und Scrum-Events plant und ermöglicht.

Das Development-Team wird durch den Scrum-Master in der praktischen Umsetzung von Scrum unterstützt. Innerhalb von Teams, die bisher wenig oder keine Erfahrung mit Scrum hatten, übernimmt er auch die Einweisung aller Beteiligten und hilft bei den ersten Schritten mit den neuen Praktiken. Im laufenden Betrieb stellt er sicher, dass das Development-Team sich auf die eigentlichen Aufgaben fokussieren kann ohne von außen abgelenkt zu werden. Er hilft außerdem dabei, dass sich das Team selbst besser organisiert und die bereits beschriebene cross-Funktionalität lebt.

Für den Rest der Organisation übernehmen Scrum-Master eine beratende Funktion. Sie sind dafür verantwortlich die notwendigen Änderungen herbeizuführen, damit Scrum von den Teams angenommen werden kann. Dazu zählt die Überzeugung der Verantwortlichen vom Konzept als auch die konkrete Planung der nächsten Schritte um Scrum innerhalb der Organisation so produktiv wie möglich umzusetzen.

2.1.1.3. Development-Team

Vom Development-Team wird die Entwicklungsarbeit getätigt. Mehrere Experten aus oft unterschiedlichen Fachrichtungen arbeiten gemeinsam daran, am Ende jeden Sprints ein fertiges Produkt zu liefern. Das Team sollte dabei so aufgestellt sein, dass die Entwickler allein die Fortschritte erreichen können, die zuvor gesetzt wurden.

Diese erzwungene cross-Funktionalität erhöht nicht nur die Möglichkeit tatsächlich selbstständig zu agieren, sondern fördert zusätzlich die Synergie innerhalb des Teams. Die angestrebten Ziele sind eine erhöhte Effizienz und Effektivität des Teams bei der Umsetzung des Produkts.

Hierbei spielt auch die Größe des Teams eine bedeutende Rolle. Für eine gute Kommunikation innerhalb des Development-Teams bietet es sich an die Anzahl an Entwicklern klein zu halten. Dies fördert den Kommunikationsfluss untereinander ohne einen hohen Verwaltungsaufwand zu benötigen. Zu kleine Teams sind jedoch ebenfalls problematisch, da die Selbstständigkeit des Development-Teams schwierig zu gewährleisten ist, wenn zu wenige Experten vorhanden sind um jeden Sprint ein lauffähiges Produkt zu liefern. Deshalb wird empfohlen eine Teamgröße von 3 - 9 Entwicklern zu wählen.

Unabhängig von der Größe sind die wichtigsten Eigenschaften des Development-Teams [Glo13, S. 70f]:

- Selbstorganisation
- Cross-Funktionalität
- Gleicher Rang für jeden (=Entwickler)
- Es gibt keine Aufspaltung in kleinere Einheiten innerhalb des Teams
- Das Team ist immer als Ganzes verantwortlich für den Erfolg oder Misserfolg eines Sprints

2.1.2. Scrum-Events

Nach der Erläuterung aller beteiligten Personen in Scrum wird nun auf die einzelnen Ereignisse eingegangen die regelmäßig vorkommen müssen. Um die Lesbarkeit in Verbindung mit anderen Texten innerhalb dieses Bereiches zu erhöhen werden auch hier weiterhin die englischen Begriffe verwendet.

Die Ereignisse in Scrum sind fest eingeplante Termine an denen sich die Mitglieder des Scrum-Teams treffen. Sie reduzieren die Notwendigkeit Treffen zu vereinbaren die außerhalb der Scrum-Events stattfinden, da sie regelmäßiges Feedback und Gesprächsbedarf aller Beteiligten abdecken sollten. Um die Ereignisse möglichst effizient abzuhalten sind alle zeitlich nach oben begrenzt.

Der Sprint selbst ist dabei wohl der wichtigste Punkt, da er alle anderen Ereignisse umschließt. Im Sprint wird die eigentliche Entwicklungsarbeit geleistet. Alle anderen dienen dazu den Sprint zu planen oder ihn in der Retroperspektive zu betrachten und zu bewerten. Dazu gehört die kritische Auseinandersetzung damit, welche Aspekte gut oder schlecht liefen. Ebenso schaffen sie Raum um Lösungsvorschläge zu entwickeln die den Prozess besser auf das Team anpassen.

2.1.2.1. Sprint

Der Hauptpunkt in Scrum ist der Sprint. Möglichst kurz gehalten wird vom Development-Team hier ein lauffähiges Produkt entwickelt. Die Dauer sollte nicht mehr als einen Monat

überschreiten jedoch auch nicht zu kurz sein um zumindest relevante Punkte des Product-Backlogs abarbeiten zu können. Dies ermöglicht eine klare Definition welche Arbeiten innerhalb des Sprints geleistet werden können und kann bei Misserfolg entsprechend schnell aufgefangen und angepasst werden. Längere Sprints würden lediglich das Risiko erhöhen.

2.1.2.2. Sprint-Planning

Welche Punkte innerhalb eines Sprints abgearbeitet werden, wird vom kompletten Scrum-Team gemeinsam beim Sprint-Planning festgelegt. Der Zeitrahmen beträgt hierfür mehrere Stunden und sollte nie mehr als einen Arbeitstag überschreiten.

Es sollte an diesem Ereignis sowohl geklärt werden, welche Arbeiten innerhalb des nächsten Sprints erledigt werden können, als auch welche Ressourcen dafür zur Verfügung stehen. Hierfür diskutiert der Product-Owner zusammen mit dem Team die Ziele die er für besonders wichtig hält. Welche Arbeitspunkte gewählt werden definiert das Product-Backlog. Die jeweilige Priorität bestimmt, welche Punkte zuerst abgearbeitet werden. Wie viele der Einträge jedoch verwendet werden, bestimmt das Development-Team allein. Ausschlaggebend hierfür ist die Erfahrung der letzten Sprints, voraussichtliche Verfügbarkeit der Mitarbeiter und wie weit das Projekt als Ganzes bisher vorangeschritten ist.

Mit Hilfe der Einträge die aus dem Product-Backlog ausgewählt wurden, wird ein Sprint Ziel definiert. Dieses dient dazu, dem Development-Team eine übergeordnete Richtung zu geben auf das hingearbeitet werden kann. Das Ziel muss durch die Umsetzung der ausgewählten Einträge innerhalb des Sprint Zeitraums erreichbar sein. Nachdem das Ziel definiert wurde, werden die Punkte in den sogenannten Sprint-Backlog übernommen. Dieser zeigt für alle Mitglieder des Teams zu jedem Zeitpunkt an, welche Arbeit innerhalb des Sprints erledigt werden muss.

Nach der Zieldefinition und der Übernahme in das Sprint-Backlog erfasst das Development-Team einzelne Arbeitspakte für jedes Mitglied für die kommenden Tage. Die Planung der Arbeitsaufteilung ist wichtig um eventuelle Missverständnisse bei der Umsetzung der Punkte im Sprint möglichst frühzeitig zu entdecken. Umso konkreter diese ausgearbeitet werden, umso besser kann der Product-Owner aufkommende Fragen schon in der Planungsphase beantworten.

Es ist zu jedem Zeitpunkt innerhalb des Sprints möglich Punkte aus dem Sprint-Backlog zu löschen oder neue hinzuzufügen, falls es vom Team absehbar ist, dass dies notwendig ist. In Absprache mit dem Product-Owner werden die Gründe eruiert und entsprechende Anpassungen am Sprint gemacht. Diese Erfahrungen sollten im Sprint-Review aufgearbeitet und in der nächsten Planungsphase berücksichtigt werden.

2.1.2.3. Sprint-Review

Am Ende eines jeden Sprints wird allen Beteiligten das Ergebnis im Sprint Review vorgestellt. Dies dient dazu die Fortschritte im Product-Backlog festzustellen und eventuell eingeladenen Interessenvertretern den Teilerfolg vorzustellen. Das Treffen sollte möglichst viel Feedback darüber sammeln, welche Punkte innerhalb des letzten Sprints besonders gut abgelaufen sind und welche Punkte verbessert werden können. Aufgrund der Erfahrungen wird hier bereits vor dem nächsten Sprint-Planning besprochen, welche Arbeiten als nächstes angegangen werden sollten, um das Produkt maximal effizient voranzubringen. Ebenso finden aufgrund des vorgestellten Standes Anpassungen an Zeit- und Budgetplan statt, falls nötig.

Das Treffen sollte nicht mehr als vier Stunden dauern und sein Ergebnis wird in Form eines verbesserten Product-Backlogs festgehalten.

2.1.2.4. Daily-Scrum

In den Daily-Scrum-Meetings, die jeden Tag vor der Arbeit stattfinden sollten, werden die Ziele für den Arbeitstag definiert. Innerhalb von 15 Minuten trifft sich das Development-Team um gemeinsam zu besprechen, welche Probleme es am letzten Arbeitstag gab und wie weit der jeweilige Entwickler vorangekommen ist. Probleme, die das Erreichen des definierten Ziels gefährden, werden damit frühzeitig erkannt.

Durch das konstante Feedback von jedem Entwickler wird die Kommunikation innerhalb des Teams gestärkt und Lösungen für etwaige Probleme gemeinsam schnell und selbstständig im Team entwickelt. Diese dauerhafte Erhöhung der Wissensbasis aller Beteiligten mindert das Risiko, ein Sprint Ziel nicht zu erreichen.

2.1.3. Scrum-Artefakte

Die Scrum-Artefakte wurden bereits in den vorangegangenen Kapiteln teilweise beschrieben. Hier sollen sie noch einmal kurz aufgegriffen und erklärt werden.

2.1.3.1. Product-Backlog

Das Product-Backlog wird hauptsächlich vom Product-Owner verwaltet und enthält die Liste aller Anforderungen an das Produkt die zum jeweiligen Zeitpunkt bekannt sind. Damit ist das Product-Backlog ständigen Änderungen unterworfen, sobald neue Anforderungen vom Markt bekannt werden.

Um Kommunikation und Transparenz zu fördern, sollte das Product-Backlog jederzeit für das Development-Team zugänglich sein und möglichst gut beschriebene und eindeutige Punkte enthalten. Dabei müssen die oberen Einträge im Backlog deutlich detaillierter beschrieben sein als Punkte am Ende. Die obersten Anforderungen sollten so gut beschrieben sein, dass sie im nächsten Sprint-Planning vom Team so verstanden werden können, dass sie direkt mit der Arbeit beginnen können. Durch die geordnete Liste müssen die Einträge am Ende zunächst nur minimal erfasst werden und können nur grobe Ziele beschreiben. Mit der Zeit und weiterem Wissen können diese weiter verfeinert werden, um für die kommenden Sprints bereit zu sein.

Im Product-Backlog stehen die Beschreibungen der Anforderungen, der geschätzte Arbeitsaufwand und erhoffte Nutzen den die Umsetzung erbringt. Dies ist wichtig für die Anordnung und erlaubt eine bessere Auswahl der Einträge durch das Development-Team im Sprint-Planning.

Zu jedem Zeitpunkt sollte es mit Hilfe des Product-Backlogs möglich sein, alle verbleibende Arbeit innerhalb des Projektes auf einmal zu erfassen. Zusammen mit den im Sprint-Review erfassten Daten kann damit ständig der Fortschritt des Projektes abgelesen werden. Dies ermöglicht bessere Zeit- und Aufwandsschätzungen für das Projekt gegenüber den Interessenvertretern.

2.1.3.2. Sprint-Backlog

Im Sprint-Backlog werden alle Anforderungen festgehalten, die innerhalb des jeweiligen Sprints abgearbeitet werden sollen. Zusätzlich dazu wird sowohl das definierte Ziel aus der Planungsphase als auch das Konzept zur Umsetzung der jeweiligen Arbeitspakete festgehalten. Damit beschreibt es die Prognosen aller Teammitglieder für den nächsten Sprint.

Wie der Product-Backlog, ist auch der Sprint-Backlog ständig im Wandel. Sobald neue Erkenntnisse oder Probleme auftauchen, die zuvor nicht prognostiziert wurden, wird das Sprint-Backlog entsprechend angepasst. Ebenso sollte der Zugang für alle Mitglieder jederzeit möglich sein damit alle auf dem gleichen Stand sind. Änderungen am Sprint Backlog werden ausschließlich durch das Development Team vorgenommen.

2.1.3.3. Increment

Das Increment beschreibt den momentanen Stand des Produktes. Da die Anforderung von Scrum ist, dass nach jedem Sprint ein lauffähiges Produkt zur Verfügung steht, beschreibt das Increment immer den Stand nach dem letzten Sprint. Es wird damit automatisch ersetzt und sollte immer zu einem gewissen Grad funktionsfähig sein, egal ob es eine Releaseversion ist oder nicht.

Damit sind die wichtigsten Eigenschaften von Scrum erläutert. Im nächsten Kapitel folgt die Vorstellung von Behaviour Driven Development im Zusammenhang mit Cucumber bevor auf die empirischen Grundlagen eingegangen wird.

2.2. BDD

Die relative junge Technik der „Verhaltensgetriebenen Softwareentwicklung" (abgekürzt BDD = Behavior Driven Development) wurde erstmals 2006 von Dan North im „Better Software" Magazin vorgestellt.[Nor06] Der Grundgedanke entsprang einem Missstand den North im praktischen Einsatz von testgetriebener Entwicklung sah. Die Probleme die North dabei auffielen, waren unter anderem das Programmierer, die erstmals TDD verwendeten, oft nicht wussten an welchen Punkten sie tatsächlich Tests schreiben sollten und an welchen Stellen dies überflüssig war. Zusätzlich dazu kam eine teilweise erschwerte Lesbarkeit der Tests aufgrund des verwendeten Vokabulars das im TDD Raum üblich war. Ständig das Wort „Test" überall zu lesen verwirrte ihn zusehends mehr als das es ihm half den Testinhalt zu verstehen. Aus diesen Problemen heraus entwickelte er JBehave.

JBehave ist die erste BDD-Rahmenarchitektur die offiziell vorgestellt wurde. Es wurde 2003 entwickelt und war als Ersatz für JUnit gedacht. Mit JBehave ist es möglich komplett auf das Testvokabular von TDD zu verzichten und trotzdem oder gerade deswegen Tests zu schreiben die verständlich sind. Hierfür wird ein gewünschtes Verhalten zunächst grob erfasst. Dazu zählt die Person oder die Rolle, die ein Verhalten der Software wünscht. Darauf folgt sinnvollerweise das Verhalten selbst, dass erzielt werden soll. Um dem ganzen ein Ziel zu geben wird danach angegeben, welches Problem damit gelöst wird. In der Vorlage wird es folgendermaßen beschrieben.

- As a [X]
- I want [Y]
- so that [Z]

X beschreibt steht hier für die Rolle, Y für das gewünschte Verhalten und Z für das damit verbundene und hoffentlich gelöste Problem. Diese drei Punkte zu klären sollte immer der erste Schritt sein, wenn ein neuer Test entwickelt wird. Nach dieser vereinfachten Userstory müssen die Akzeptanzkriterien definiert werden, mit denen die Lösung des Problems sichergestellt werden kann. Um dies ebenfalls in einen Rahmen zu bringen wird von BDD auch hier eine Vorlage bereitgestellt.

1. Given [context]
2. When [event]
3. Then [outcome]

Der erste Schritt mit dem Stichwort „Given" beschreibt dabei einen Kontext der gegeben sein muss um das Verhalten zu ermöglichen. Hierzu zählen alle Schritte die als einmalige Konfiguration des Systems benötigt werden, damit das danach folgende „Event" im „When"-Schritt stattfinden kann. Das äquivalente Mittel in TDD ist der „setup()" Schritt

in JUnit-Tests. Hier werden die benötigten Objekte instanziiert und falls benötigt mit entsprechenden Daten gefüllt.

Im „When"-Schritt wird das Ereignis definiert das auftreten soll. Hierzu zählen alle denkbaren Tätigkeiten innerhalb des Kontextes von der jeweiligen Rolle, die zuvor definiert worden ist.

Im letzten Schritt wird „dann" das erwartete Ergebnis sichergestellt. Dies entspricht den aus TDD bekannten Zusicherungen.

Erst nachdem dieser Rahmen vollständig definiert wurde, wird mit der Implementierung der Funktionalität begonnen. Dies entspricht dem „Test-First"-Prinzip und spiegelt erneut wieder, wie stark BDD anfangs von TDD beeinflusst wurde. Der Rahmen aus Given, When und Then-Schritten, kann mit Hilfe von sogenanntem Verbindungscode mit dem Produktivcode verbunden werden, um die automatisierte Ausführung des Tests zu ermöglichen.

Seit der Vorstellung von BDD in seinen Anfangsphasen hat sich das System ständig weiterentwickelt. Es kann sich nun deutlich besser von seinem Vorgänger TDD abheben. Eine neuere Definition von BDD könnte folgendermaßen formuliert werden.[Che10, S. 138]

```
Bei verhaltensgetriebener Entwicklung wird eine Anwendung entwickelt, indem
ihr Verhalten aus der Sicht Ihrer Interessenvertreter beschrieben wird.
```

Bei dieser Definition werden die Veränderungen, die BDD in den letzten Jahren durchgemacht hat, sehr schnell deutlich. Es hat sich vom reinen syntaktischen Umformulieren der TDD Technik weiterentwickelt und definiert die Art und Weise wie Tests geschrieben werden neu. Hierzu muss man die Sichtweise eines Interessenvertreters verstehen, bevor die Software entwickelt werden kann. Diese Vorraussetzung ist zwar prinzipiell für jede Entwicklungstechnik gegeben, jedoch liegt ein stärkerer Fokus darauf die Domäne, die Probleme und die Herausforderungen zu verstehen, die ein Anwender hat, der die Software einsetzt. Die Entwickler müssen mit der Sprache und den Begriffen vertraut sein, welche die Anwender täglich verwenden um Ihre Tätigkeiten zu beschreiben.

Bei BDD wird immer angenommen, dass es mehrere Interessenvertreter gibt die alle unterschiedliche Sichtweisen haben. Es reicht nicht nur die Sichtweise des Auftraggebers zu erfassen welcher die Software bezahlt. Die Sichtweise eines jeden der ein Interesse an der Software hat, sollte erfasst werden, um eine zufriedenstellende Anwendung implementieren zu können. Als Beispiel für die unterschiedlichen Interessenvertreter können Anwender, Administratoren, Sicherheitsbeauftragte, Anwälte und Supportteams in Frage kommen. Diese Liste ist nicht erschöpfend und es können von Projekt zu Projekt unterschiedliche Gruppen mit dem entwickelten System in Berührung kommen.[Che10, S. 139] Wichtig ist, dass alle Gruppen die von der Software beeinflusst werden auch in die Planung zu einem gewissen Grad einbezogen werden.

2.2.1. Prinzipien

BDD hat mittlerweile auch eigene Prinzipien nach denen vorgegangen werden soll.[Che10, S. 138] So ist es wichtig, dass Planungs-, Analyse- und Aufbauphasen klein gehalten werden. Ähnlich wie in anderen agilen Techniken soll schnell ein Punkt gefunden werden mit dem begonnen werden kann. Ab diesem Zeitpunkt wird agil weiterentwickelt und die Phasen werden im kleinen Rahmen iterativ wiederholt.

Als zweites Prinzip sollte der Fokus bei mit BDD entwickelten Anwendungen immer auf den Interessenvertretern sein. Sobald irgendetwas gemacht wird, dass nicht den Wert für

eine Interessengruppe steigert, ist man auf dem falschen Weg und sollte seine Richtung sofort ändern. Die Interessenvertreter definieren immer die richtige Richtung.

Als letztes Prinzip sollte stets darauf geachtet werden, dass alles als Verhalten beschrieben wird. Dies gilt für alle Ebenen der Entwicklung, sei es die Codeebene oder die Applikationsebene. Die ausschließliche Ausrichtung auf Verhaltensweisen ist beliebig feingranular möglich und gibt der Entwicklung immer eine feste Richtung vor.

2.2.2. Entwicklungsprozess

Um die Entwicklung zu starten sollte sich natürlich nicht auf alle Interessenvertreter gleichermaßen intensiv fokussiert werden. Es müssen Ziele definiert werden, die mit der Software erreicht werden sollen. Dies wird in Zusammenarbeit mit den Kernanwendern der Software gemeinsam entwickelt. Diese Anwender beschreiben möglichst detailliert welche Aufgaben von der Software abgenommen werden sollen, die momentan noch nicht automatisch oder mit Hilfe von Software erledigt werden können. Sie definieren auch, woran erkannt wird, dass das Projekt ein Erfolg ist und die Ziele erreicht wurden. Innerhalb dieses Prozesses wird eine Zusammenstellung von Merkmalen entwickelt, die generell die Funktionalität des Produktes beschreiben. Daraus abgeleitet werden sogenannte Userstories entworfen. Diese sind der Ausgangspunkt für die tägliche Arbeit innerhalb des Entwicklungsprozesses. Sie beschreiben das Verhalten, das die Software abbilden soll.

Um die Userstories zu entwickeln setzen sich ein Analyst und ein Interessenvertreter zusammen.[Che10, S. 140] Die Aufgabe des Analysten ist hierbei, die Sprache des Vertreters aufzugreifen und so zu formulieren, dass sie einerseits die Userstory sehr genau beschreiben und andererseits vom Interessenvertreter verstanden werden. Die Story sollte dabei nur wenige Tage Implementierungsaufwand umfassen, um effektiv innerhalb des Entwicklungsprozesses abgearbeitet werden zu können. Zu lange Stories sind oft ein Zeichen dafür, dass die Anforderungen nicht klar herausgearbeitet worden sind.

Zusammen mit einem Tester werden nun die Userstories besprochen.[Che10, S. 141] Hier geht es darum, genau zu definieren, welche Eingaben zu welcher Ausgabe im System führen sollen. Die Anforderungen an die Software werden hier spezifiziert um zu wissen, wie der fertige Zustand der einzelnen Userstories aussieht. Der Unterschied zwischen dem Analysten und dem Tester liegt in der Abstraktionsebene. Bei der Analyse geht es darum die generellen Features zu benennen, welche die Software besitzen muss. Als Beispiel könnte der Analyst beschreiben, dass es eine Taschenrechnerfunktion geben muss. Der Tester denkt eher in konkreten Situationen und würde die einzelnen Szenarien auflisten, welche die Funktionalität bereitstellen. Für einen Taschenrechner könnten das beispielsweise konkrete Division („Wenn ich 4/2 eingebe...") sein. Einen guten Tester zeichnet aus, dass er die Randfälle im Blick hat („Wenn ich 2/0 eingebe...") und entsprechend ausgereifte Szenarien entwickeln kann, die diese Randfälle exakt überprüfen.

Durch die Verwendung von natürlicher Sprache beim Ausformulieren der Testszenarien kann der Interessenvertreter überprüfen, ob seine Anforderungen an das System tatsächlich erfüllt sind sobald die Tests bestanden werden. Für die Entwickler ist es ebenso hilfreich genaue Anforderungen in Form von einzelnen Testschritten zu haben, welche die Anwendung beschreiben, da sie ausschließlich sinnvolle Anforderungen implementieren und nichts darüber hinaus.

Nachdem die Szenarien definiert wurden müssen sie auf einen Stand gebracht werden, in dem sie automatisiert ausgeführt werden können.[Che10, S. 143] Dies wird in BDD mit Hilfe von Rahmenarchitektur wie „Cucumber" oder „RSpec" erreicht. Die finale Aufgabe liegt beim Entwickler den Verbindungscode zwischen Testszenario und der Anwendung bereitzustellen und danach die eigentliche Funktionalität in das System einzubauen, damit der

Test erfolgreich ausgeführt wird. Ähnlich wie in TDD wird hierbei zunächst nur das Minimum entwickelt um den Test durchführen zu können. Danach erfolgt eine Refaktorisierung bis die Funktionalität zufriedenstellend erfüllt wird.

Wird dieser komplette Prozess für eine Userstory erfolgreich abgeschlossen kann der Interessenvertreter sehr einfach nachvollziehen, ob seine Wünsche und Anforderungen erfüllt wurden. Dies gibt die Möglichkeit Feedback sehr effizient zu einem frühen Zeitpunkt zu bekommen.[Che10, S. 145] Dadurch können eventuell gewünschte Änderungen kostengünstig und schnell umgesetzt werden. Wird dieser Prozess für alle gewünschten Features iterativ wiederholt ist die Chance hoch, dass Software entwickelt wird, die tatsächlich die Bedürfnisse des Kunden befriedigt.

Bevor die praktischen Aspekte dieses Prozesses im nächsten Kapitel im Zusammenhang mit Cucumber vorgestellt werden, wird zunächst noch darauf eingegangen, was eine Userstory ausmacht.

2.2.3. Userstory

Eine Userstory enthält immer drei Komponenten:[Che10, S. 146]

- Einen Titel der die Story kurz beschreibt.
- Die Geschichte um die es in der Story geht.

  Die einfachste Form ist hierbei die zuvor erwähnte Vorlage zu verwenden, die auch als „Connextra"-Format bekannt ist: „As a [X], I want [Y], so that [Z]". Hiermit können die einzelnen Interessenvertreter, ihre Anforderungen und den erhofften Vorteil durch die Umsetzung dessen in kurzer Form festgehalten werden.
- Akzeptanzkriterien, um festzulegen, welche Anforderungen genau erfüllt sein müssen, um die Story erfolgreich abschließen zu können.

Die Userstories werden danach weiter verfeinert und mit Szenarien aufgefüllt um die Details zu klären. Dieser Schritt erfolgt in dem bereits erwähnten „Given, When, Then"-Schema. Die Szenarien werden im nächsten Kapitel im Zusammenhang mit Cucumber und einem kleinen Beispiel genauer erläutert.

2.2.4. Cucumber

Cucumber ist eine Weiterentwicklung von RSpec's „Story Runner", welcher wiederum auf RBehave aufbaute, der JBehave als Vorgänger hatte. Das Cucumber-Projekt wurde 2008 von Aslak Hellesøy ins Leben gerufen und wird seitdem stets verbessert. Die Entwicklung hat dabei die grundlegende Herangehensweise nicht verändert, mit der ein Testszenario in BDD beschrieben wird. Der Fokus war vielmehr auf der Verbesserung der Benutzerfreundlichkeit der Werkzeuge.[Hel] Cucumber ist daher weiterhin eine Rahmenarchitektur um Akzeptanztests auszuführen.

Die ursprüngliche Version wurde für die Programmiersprache Ruby entwickelt. Mittlerweile gibt es Portierungen für weitere Sprachen wie etwa Java oder C++ und C#. Um die einzelnen Akzeptanztests zu beschreiben wird die Sprache „Gherkin" verwendet.

2.2.4.1. Gherkin

Gherkin ist im wesentlichen eine Weiterentwicklung des „Given, When, Then" Schemas. Ursprünglich von Dan North vorgestellt, wurden etliche weitere Elemente wie etwa Tabellen hinzugefügt, um die Mächtigkeit der Sprache zu verbessern. Obwohl Gherkin eine Programmiersprache ist, (nicht Turing-vollständig) sollte der Fokus nicht auf dem Programmierer liegen. Wie im BDD-Kapitel erwähnt, ist die Aufgabe eines Akzeptanztests

sicherzustellen, dass der damit verbundene Interessenvertreter den Testinhalt akzeptiert. Die Lesbarkeit des Tests für einen nicht technisch versierten Menschen muss immer das oberste Ziel sein.[WH12, S.27]

Um Gherkin im Einsatz zu sehen, werden die wichtigsten Syntaxelemente und Schlüsselwörter der Sprache anhand eines Beispiels erklärt. Alle Elemente von Gherkin sind lokalisiert und können somit in vielen gesprochenen Sprachen verwendet werden, im folgenden werden jedoch nur die englischen Elemente verwendet.

Quelltextausschnitt 2.1: Kreditkartenbeispiel

```
Feature: Feedback when entering invalid credit card details

        In user testing we've seen a lot of people who made mistakes
           entering their credit card. We need to be as helpful as
           possible here to avoid losing users at this crucial stage
           of the transaction.

        Scenario: Credit card number too short
                When I enter a card number that's only 15 digits long
                And all the other details are correct
                And I submit the form
                Then the form should be redisplayed
                And I should see a message advising me of the correct
                    number of digits

        Scenario: Expiry date invalid
                When I enter a card expiry date that's in the past
                And all the other details are correct
                And I submit the form
                Then the form should be redisplayed
                And I should see a message telling me the expiry date
                    must be wrong
```

Feature

Das erste Schlüsselwort in obigem Beispiel2.1 ist „Feature". Feature ist kein Steuerelement der Sprache, sondern sollte als Zusammenfassung genutzt werden, um die Szenarien innerhalb dieses Tests zu beschreiben. Bis zum nächsten Schlüsselwort von Gherkin werden alle folgenden Zeilen als Dokumentation angesehen. Der Text in der selben Zeile wie Feature gibt den Namen an. Alle anderen Zeilen sind die Beschreibung. Der Feature-Block endet sobald eines der folgenden Schlüsselwörter eingeben wurde:

- Background
- Scenario
- Scenario Outline

Background

Background gibt die Möglichkeit generelle Schritte auszuführen, die für alle Szenarien im Feature benötigt werden.[WH12, S.62] Dies verringert Redundanz und erhöht die Übersicht beim Lesen der einzelnen Szenarien, da diese deutlich kleiner werden und sich nur im Wesentlichen unterscheiden. Der Background-Teil ist nicht für jedes Feature nötig, kann jedoch nützlich sein. Der Background-Teil soll dem Leser des Tests helfen die einzelnen Szenarien besser zu verstehen. Technische Anforderungen, wie etwa das Starten von Servern für das Feature, gehören daher nicht in den Background-Teil. Diese tragen nicht

zum Verständnis bei und werden von einem Leser meist implizit angenommen und sollten deshalb über Tags, die später erklärt werden, erreicht werden. Generell gilt, dass der Background-Teil kurz gehalten werden sollte und damit eine Geschichte erzählt wird.

Scenario

Um das Verhalten tatsächlich zu beschreiben enthält jedes Feature einen oder mehrere Szenarien.[WH12, S.30] Jedes Szenario beschreibt dabei unterschiedliche Aspekte. Oft wird dabei eine Unterteilung der Randfälle und verschiedener Pfade im System gemacht. Alle Szenarien zusammen sollten das komplette Feature abdecken. Das heißt, dass nach der erfolgreichen Ausführung aller Szenarien innerhalb einer Feature-Datei die Anforderungen einer Userstory erfüllt sein müssen.

Alle Szenarien bauen auf dem „Given, When, Then"-Format auf. Zunächst wird mit „Given"-Schritten das System in einen bestimmten Zustand versetzt. Danach werden Aktionen auf dem System ausgeführt, die mit „When"-Schritten beschrieben werden. Im „Then" Teil wird der neue Zustand überprüft und mit dem erwarteten Ergebnis verglichen.

Jedes Szenario kann aus einem oder mehreren dieser Schritte bestehen. Weitere Schritte können entweder mit dem gleichen Schlüsselwort beginnen oder die Wörter „And" und „But" verwenden. Ein Beispiel2.2:

Quelltextausschnitt 2.2: Abgelaufene Sitzung Beispiel

```
Scenario: Submitting new post using expired session
        Given I am logged in the forum
        And I have written a new post
        But my session is expired
        When I click on "Submit"
        Then the system tells me that I have to log in again
        And my text is saved temporarily in the system
```

Für den Parser von Cucumber ist es dabei egal, ob die Schritte mit den Schlüsselwörtern „And" und „But" erweitert werden. Es besteht für die Ausführung kein Unterschied, ob mehrere Given-Schritte verwendet werden oder die Erweiterung mit „And" stattfindet. Daher hätte das gleiche Szenario auch folgendermaßen definiert werden können2.3:

Quelltextausschnitt 2.3: Abgelaufene Sitzung Beispiel ohne And/But

```
Scenario: Submitting new post using expired session
        Given I am logged in the forum
        Given I have written a new post
        Given my session is expired
        When I click on "Submit"
        Then the system tells me that I have to log in again
        Then my text is saved temporarily in the system
```

Der Unterschied liegt ausschließlich in der Lesbarkeit der einzelnen Szenarien. Alle Schlüsselwörter innerhalb eines Szenarios sollten daher so verwendet werden, dass das gewünschte Verhalten so lesbar wie möglich beschrieben wird. Tatsächlich wird in Cucumber nicht einmal der Unterschied zwischen Given, When und Then gemacht. Alle Schritte werden lediglich mit dem Code dahinter verglichen und nach einander ausgeführt. Deshalb könnte das Beispiel in Ausschnitt 2.4 auch mit dem universellen Schlüsselwort „*" beschrieben werden.

Quelltextausschnitt 2.4: Abgelaufene Sitzung Beispiel mit *

```
Scenario: Submitting new post using expired session
        * I am logged in the forum
```

```
        * I have written a new post
        * my session is expired
        * I click on "Submit"
        * the system tells me that I have to log in again
        * my text is saved temporarily in the system
```

Alle drei Formulierungen dieses Beispiels werden gleichermaßen ausgeführt. Welche Variante innerhalb des Teams eingesetzt wird muss jeweils individuell geklärt werden. Aufgrund der Tatsache das die Tests mit Menschen entwickelt wird die nicht immer einen technischen Hintergrund haben, ist es jedoch empfehlenswert die Formulierungen so zu wählen, dass eine komplette Geschichte erzählt wird die jeder nachvollziehen kann. Die Lesbarkeit der Szenarios für einen Menschen unterscheidet sich bei den verschiedenen Varianten erheblich.

Einer der wichtigsten Prinzipien für die Erstellung von Szenarien ist die zustandslose Formulierung. Jedes Szenario sollte einzeln ausführbar sein und nicht auf dem Zustand eines anderen Szenarios aufbauen.[WH12, S.32] Cucumber selbst kann nicht verhindern, dass diese Art von Szenarios entwickelt wird. Es ist Aufgabe des Testers unabhängige Szenarien zu formulieren, damit diese einzeln testbar sind. Deshalb sollte immer angenommen werden, dass sich das System im Ausgangszustand befindet und alle notwendigen Aktionen mit Hilfe von Given-Schritten ausgeführt werden müssen. Wenn diesem Prinzip nicht gefolgt wird, werden unnötige Abhängigkeiten mit in die Szenarien aufgenommen und die Wartung der Tests verschlechtert sich immens, da Änderungen innerhalb eines Szenarios nicht mehr isoliert gemacht werden können.

Scenario Outline

Das letzte verbleibende Schlüsselwort das einen Feature-Block beendet ist „Scenario Outline".

Bei der Erstellung von Szenarien kommt es oft vor, dass sich die unterschiedlichen Fälle innerhalb des Features nur in der Eingabe der Daten und der daraus resultierenden Ergebnisse unterscheiden. Die Reihenfolge der verwendeten Schritte bleibt dabei oft gleich. Ein Parkhaus könnte beispielsweise mehrere Tarife anbieten, die je nach Parkdauer eine andere Berechnung erfordern. Aus Kundensicht ist das Verhalten des Systems beim Bezahlen am Automaten jedoch immer gleich. Im Feature könnte das folgendermaßen aussehen2.5:

Quelltextausschnitt 2.5: Parkhausbeispiel ohne Tabelle

```
Scenario: Paying for parking 0.5 hour
        Given I have my parking ticket
        And the ticket has 0.5 hour as parking duration
        When I insert my ticket into the automat
        Then the system tells me I have to pay 0

Scenario: Paying for parking 1 hour
        Given I have my parking ticket
        And the ticket has 1 hour as parking duration
        When I insert my ticket into the automat
        Then the system tells me I have to pay 5

Scenario: Paying for parking 2 hour
        Given I have my parking ticket
        And the ticket has 2 hour as parking duration
        When I insert my ticket into the automat
        Then the system tells me I have to pay 10
```

Die Geschäftslogik unterscheidet hierbei, wie das System auf die unterschiedliche Dauer reagiert; der Ablauf ist jedoch in jedem Fall der gleiche. Die Lesbarkeit der Tests leidet

unter der Wiederholung und beim Überfliegen der Szenarien können Menschen die Unterschiede zwischen den verschiedenen Varianten nur mühsam erkennen. In Gherkin kann dieses Problem deshalb eleganter gelöst werden, indem variable Eingaben und Ausgaben eingesetzt werden. Dies geht über das Schlüsselwort „Scenario Outline".

Quelltextausschnitt 2.6: Parkhausbeispiel mit Tabelle

```
Scenario Outline: Paying for parking hours
        Given I have my parking ticket
        And the ticket has <duration> hour as parking duration
        When I insert my ticket into the automat
        Then the system tells me I have to pay <amount>

Examples:
        | duration      | amount        |
        | 0.5           | 0             |
        | 1             | 5             |
        | 2             | 10            |
        | 5             | 25            |
```

Die Formulierung der unterschiedlichen Fälle in Ausschnit 2.6 ist deutlich einfacher zu lesen und sehr effizient erweiterbar für weitere Fälle. Cucumber führt beim Outline das Szenario für jede Zeile einzeln nacheinander aus. Die Platzhalter, die mit den spitzen Klammern im Text gegeben sind, werden bei der Ausführung mit den Werten aus der Tabelle darunter befüllt. Es können beliebig viele Platzhalter verwendet werden. Die Tabelle ist schnell und einfach erstellt und weitere Platzhalter können mit zusätzlichen Trennzeichen „|" hinzugefügt werden. Innerhalb eines Scenario Outlines können mehrere Beispieltabellen verwendet werden, um eine bessere Gruppierung zu ermöglichen. Zusätzlich zur verbesserten Les- und Wartbarkeit können mit Hilfe der tabularen Darstellung die Randfälle oft einfacher identifiziert werden, da die Beispiele die minimalen und maximalen getesteten Werte sehr übersichtlich darstellen. Ebenso werden Lücken bei der Testabdeckung gut sichtbar. Der Tester kann dadurch sehr schnell sehen welche Eingaben fehlen oder eventuell redundant sind, da sie schon von anderen Zeilen abgedeckt werden. Aus der Sicht eines Programmierers sollte darauf geachtet werden, dass nicht alle Redundanz aus den Beispielen entfernt wird, damit die dahinter liegende Logik im Test schnell erkannt werden kann. Allerdings sollte ebenfalls darauf geachtet werden, dass nicht alle möglichen Kombinationen getestet werden, damit die Ausführungsdauer der Tests so gering wie möglich gehalten wird.

Damit sind alle Syntaxelemente die nach einem Feature-Block folgen können geklärt. Bevor die Ausführung der einzelnen Schritte erklärt wird, folgt zunächst noch eine kurze Erklärung von Datentabellen und Tags die in Cucumber oft genutzt werden.

Datentabellen

Bei Datentabelle handelt es sich in Cucumber um die Möglichkeit einzelne Testschritte mit variablem Inhalt zu wiederholen. Dies zielt ebenfalls wie die meisten anderen Syntaxelemente in Gherkin auf eine verbesserte Lesbarkeit ab.[WH12, S. 64] Die Formulierung von sich wiederholenden Schritten ist für den Leser anstrengend und Unterschiede werden wie schon bei den Beispieltabellen in Scenario Outlines schlechter erkannt. Das Beispiel in Ausschnitt 2.7 zeigt die Problematik die mit Datentabellen gelöst wird.

Quelltextausschnitt 2.7: Schauschpielersuche Beispiel

```
Scenario: Search by actor
        Given the movie "Terminator"
        And the movie "Total Recall"
```

```
        And the movie "True Lies"
        And the movie "Rocky"
        And the movie "Rambo"
        When I search for actor "Stallone"
        Then the list contains "Rocky" and "Rambo"
```

Die Lesbarkeit des Szenarios leidet unter der Wiederholung der Given-Schritte erheblich. Hier bietet sich wie schon zuvor die Darstellung der Daten als Tabelle oder Liste an. Die Tabellen werden ebenfalls mit dem Trennzeichen „|" unterteilt und jede Zeile ergibt eine neue Wiederholung des vorherigen Schrittes. Dies erhöht nicht nur die Lesbarkeit sondern auch die Wartbarkeit und Erweiterungsfähigkeit des Szenarios. Das obige Beispiel könnte demnach in Ausschnitt 2.8 umformuliert werden:

Quelltextausschnitt 2.8: Schauschpielersuche Beispiel mit Datentabelle

```
Scenario: Search by actor
        Given the movies:
        | Terminator     |
        | Total Recall   |
        | True Lies      |
        | Rocky          |
        | Rambo          |
        When I search for actor "Stallone"
        Then the list contains "Rocky" and "Rambo"
```

Tags

Tags werden in Gherkin dafür verwendet um Features und Szenarien in Gruppen zu organisieren. Damit ist es beispielsweise möglich die Features in schnelle und langsam laufende Tests zu unterscheiden, um die wiederholte Ausführung durch den Entwickler zu unterstützen. Ein Beispiel ist in Ausschnitt 2.9 zu sehen.

Quelltextausschnitt 2.9: Tags Beispiel

```
@Slow, @On3
Scenario: Slow test
        Given a mighty setup
        When I execute an O(n^3) operation...

@Fast, @On1
Scenario: Fast test
        Given a quick setup
        When I execute an O(n) operation...
```

Tags werden in Gherkin mit „@TagsString" definiert und können vor einem Feature oder einem einzelnen Szenario gesetzt werden. Es ist möglich mehrere Tags auf einmal zu verwenden, um ein Szenario mehreren Gruppen zu zuordnen. Alle Tags die auf Feature-Ebene definiert wurden, gelten auch für alle Szenarien innerhalb des Features. Wichtig ist auch hierbei, Tags nicht so zu verwenden, dass sie weiterhin kontextlose Formulierungen der Szenarien ermöglichen und nicht zu künstlichen Einschränkungen führen.

Damit wurden die wichtigsten Eigenschaften von Gherkin vorgestellt. Die Ausführung der Feature-Dateien übernimmt Cucumber mit den sogenannten „Stepdefinitions" oder auch Code-Behind-Anweisungen genannt. Diese werden im nächsten Kapitel genauer erläutert.

2.2.4.2. Stepdefinitions

Die Ausführung der einzelnen Schritte in den Szenarien wird von Cucumber übernommen. Damit der Zusammenhang zwischen den Begriffen deutlicher wird und Stepdefinitions ein

spezieller Begriff ist, wird im folgenden das englische Wort „Steps" verwendet, wenn sich auf die Given/When/Then-Schritte innerhalb eines Features bezogen wird. Bisher wurde mit den einzelnen Szenarien lediglich festgehalten, was getestet werden soll. Wie die Tests ablaufen wurde bisher nicht erklärt. Diesen Teil übernehmen die Stepdefinitions.[WH12, S. 40ff]

Stepdefinitions können in verschiedenen Programmiersprachen geschrieben werden. Das Prinzip ist dabei jedoch immer das Gleiche. Da die eingesetzte Programmiersprache des untersuchten Teams innerhalb dieser Fallstudie mit C# arbeitet werden die verwendeten Beispiele hier auch in C# gegeben. Als Beispiel wird der Login in ein eine beliebige Anwendung genommen.

```
Given I am logged in as Steffen
```

Der Step beschreibt dabei zwei Dinge:

- Ich bin ein registrierter Anwender mit dem Accountnamen "Steffen"
- Ich bin im System angemeldet

Wie diese beiden Ziele erreicht werden wird im Step nicht geklärt. In den meisten Fällen werden Akzeptanztests so geschrieben, dass sie Nutzerverhalten simulieren. Dies könnte in diesem Fall das Klicken des Anmeldeknopfs in einem Browser bedeuten oder direkt mit der Anmeldung zu kommunizieren ohne die Nutzeroberfläche zu bedienen, indem direkt Datenbanken oder Dateien angesprochen werden. Man sollte dabei zwischen dem Automatisierungscode wie zum Beispiel „Prowser" und der StepDefinition unterscheiden. Bibliotheken wie Prowser ermöglichen es einen Anwender beim navigieren von Websiten zu simulieren und sollten strikt vom semantischen Teil getrennt werden der in Steps beschrieben ist. Abbildung2.1 veranschaulicht dies.

Sobald ein Feature ausgeführt wird sucht Cucumber nach dem ersten Szenario und dort nach dem ersten Step der definiert wurde. Die Zuordnung der Steps zu den Stepdefinitions erfolgt durch einfache reguläre Ausdrücke. Im obigen Beispiel also

```
[Given(@"I am logged in as Steffen")]
```

Wird dieses innerhalb der Projektstruktur gefunden, wird der Code ausgeführt. Im C# Fall werden alle Klassen in einem bestimmten Ordner durchsucht. Die Stepdefintions sind dort als Methoden implementiert und die regulären Ausdrücke sind Annotationen der Methode. Der Name der Methode ist dabei nicht entscheidend und kann prinzipiell beliebig gewählt werden. Aus Gründen der Wartbarkeit empfiehlt es sich jedoch auch hier einen sinnvollen Namen zu verwenden, der das Verhalten prägnant ausdrückt. In der Stepdefinition selbst wird mit normalem C#-Code gearbeitet. Hier werden die entsprechenden Bibliotheken angesprochen, die wiederum das System direkt ansprechen. Werden keine speziellen Bibliotheken benötigt, kann natürlich auch das System direkt angesprochen werden. Allgemein sind hier keine Grenzen durch Cucumber gesetzt und alle Funktionalität der Programmiersprache kann verwendet werden um das beschriebene Verhalten umzusetzen.

Argumente innerhalb der Steps bei Scenario Outlines zum Beispiel müssen im regulären Ausdruck berücksichtigt werden. Allgemein gelten alle Regeln die auch bei herkömmlichen regulären Ausdrücken zum Tragen kommen. Wird eine Gruppe innerhalb des regulären Ausdrucks erkannt (gekennzeichnet mit runden Klammern) wird dies als Argument an die Methode weitergereicht. Die Wertigkeit der Gruppe bestimmt dabei die Reihenfolge der Argumente in der Methode.

Da reguläre Ausdrücke verwendet werden muss darauf geachtet werden, dass es durchaus Steps gibt die semantisch unterschiedlich sind aber wegen zu grob gefasster Ausdrücke

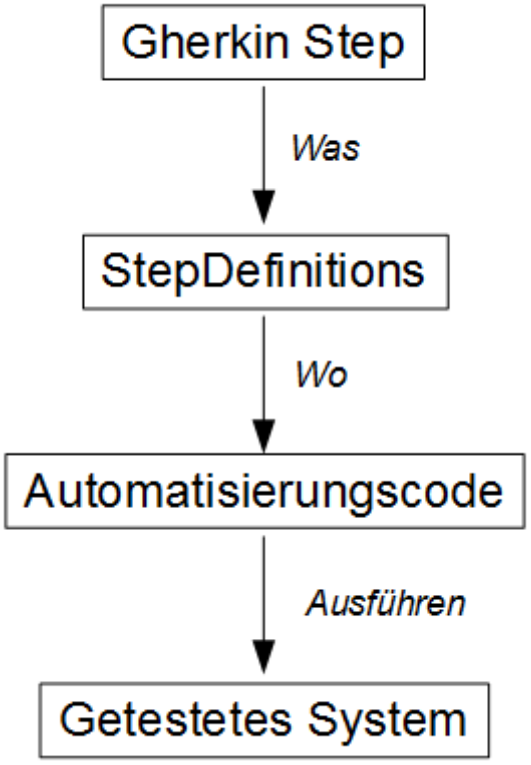

Abbildung 2.1.: Codeebenen eines Features

eine gemeinsame StepDefinition haben. Deshalb sollte bei der Erstellung der regulären Ausdrücke darauf geachtet werden, welche StepDefinitions hier angesprochen werden. Generell gilt, dass die regulären Ausdrücke so restriktiv wie möglich beschrieben werden sollten, damit es keine unabsichtliche Überschneidung der Steps gibt, die einen gemeinsame StepDefinition haben.

Sollte ein Step keine StepDefinition haben die dazu passt, wird dies bei der Ausführung angezeigt und ein entsprechender Vorschlag für einen möglichen regulären Ausdruck samt Methodenkopf gegeben die den Step repräsentiert. Bleibt dieser Vorschlag aus so bedeutet das, dass es bereits eine StepDefinition gibt die zum Step passt.

Bei der Ausführung von Steps kann es unterschiedliche Rückgaben geben:

- Undefiniert
- Unerledigt
- Fehlgeschlagen
- Bestanden

Undefinierte Steps sind diejenigen die noch keine entsprechende StepDefinition besitzen die zum Step passen. Es wird dementsprechend kein passender regulärer Ausdruck in den C#-Klassen gefunden. Der Vorschlag in der Cucumber-Fehlermeldung kann dann genutzt werden um einen einfachen regulären Ausdruck zu bekommen der exakt den String im Step entspricht. Die Ausführung der darauf folgenden Steps im Szenario wird beim antreffen eines undefinierten Steps übersprungen oder ebenfalls als undefiniert angegeben.

Unerledigte Steps sind Schritte, die zwar eine Methode haben die mit einem passenden regulären Ausdruck als Annotation versehen ist, jedoch noch nicht vollständig implementiert

wurde. Der Status wird dadurch definiert das eine „Pending-Exception" von der Methode ausgeworfen wird. Diese Ausnahme wird beim Aufruf der Funktion „Pending()" geworfen die standardmäßig beim ersten Vorschlag von unerledigten Steps von Cucumber gegeben wird. Der nachfolgende Ausschnitt 2.10 veranschaulicht dies in C#-Code.

Quelltextausschnitt 2.10: StepDefinition Beispiel

```
[Given(@"I am logged in as Steffen")]
public void GivenIAmLoggedInAsSteffen()
{
    ScenarioContext.Current.Pending("Need to implement login");
}
```

Pending kann ein String-Argument entgegen nehmen um dem Entwickler eine Mitteilung zu geben, anhand derer er sich orientieren kann.

Ein fehlgeschlagener Step ist offensichtlich ein Test bei dem ein Fehler im Programmablauf festgestellt wurde. Dies können Laufzeitfehler oder Ausnahmen sein die nicht abgefangen wurden. Diese können sich sowohl im System als auch im Code der StepDefinition befinden. Bei den typischen Then-Steps handelt es sich meist um fehlgeschlagene Vergleiche mit den erwarteten Werten. Die Zusicherungen müssen mit Mitteln der Sprache oder weiteren Bibliotheken vorgenommen werden. Cucumber selbst stellt keine solchen Methoden zur Verfügung.

Bestandene Steps sind selbst erklärend und führen lediglich zur weiteren Ausführung des Szenarios.

2.2.5. SpecFlow

SpecFlow ist eine Erweiterung von Microsofts Visual Studio um BDD einfach in .NET-Projekte zu integrieren. Im wesentlichen ist es Cucumber für die .NET Umgebung. SpecFlow selbst besteht aus drei wesentlichen Komponenten:

- IDE Integration
- Generator
- Runtime

Die IDE-Integration ermöglicht das einfache Navigieren zwischen Steps und StepDefinitions in Visual Studio (VS). Autovervollständigung und Syntaxhervorhebung werden ebenfalls von der IDE-Einheit übernommen. Zusätlich dazu werden die anderen Komponenten von SpecFlow bei der IDE-Integration miteinander verbunden. Das Kontextmenü von VS wird sowohl um die Generierung als auch um die Ausführung von Stepdefinitions erweitert.

Der Generator stellt Vorlagen für Feature-Dateien und Stepdefinition Klassen bereit.

Die Laufzeit von SpecFlow unterstützt verschiedene Zielplattformen (.NET, Silverlight, Windows Phone) und ist in Verbindung mit anderen Testrahmenarchitekturen wie NUnit oder MsTest für die Ausführung der Szenarien verantwortlich.

2.2.5.1. Sprachen

Als Sprache für die Steps wird in SpecFlow ebenfalls Gherkin eingesetzt, mit allen bereits zuvor erwähnten Funktionen. Die StepDefinitions werden in C#-Code geschrieben. Die Definitionen in anderen .NET-Sprachen zu schreiben ist nicht möglich. Allerdings ist es durch die .NET-Runtime möglich C#-Klassen in anderen .NET-Sprachen zu verwenden. Das heißt, dass es prinzipiell möglich ist auch Anwendungen in anderen .NET-Sprachen mit Hilfe von SpecFlow zu testen. Der Aufwand dafür steigt im Vergleich zu reinen C#-Projekten nur zu einem geringen Grad.

2.2.5.2. Bindings

Um in SpecFlow mit dem System-Under-Test zu kommunizieren, gibt es in SpecFlow vier verschiedene Varianten die „Bindings" genannt werden. Die meist genutzt Binding-Technik ist die simple Ausführung von StepDefinitions die mit einem Step in Verbindung gebracht werden können. Diese geschieht wie in Cucumber über die regulären Ausdrücke die als Annotation an den Methoden angegeben werden.

Zusätzlich hierzu gibt es sogenannte „hooks", die die Ausführung von zusätzlichem Code zu einem bestimmten Zeitpunkt ermöglichen. Dies kann beispielsweise vor einem Szenario der Fall sein oder direkt danach.

Die „step argument transformation" ist die im Cucumber-Kapitel erwähnte Ausführung von Beispieltabellen, bei denen die Argumente der Tabelle in die Steps eingesetzt werden und jeweils einzeln ausgeführt werden.

„Scoped Bindings" ermöglichen es in SpecFlow Steps nur dann auszuführen, wenn sie in einem bestimmten Geltungsbereich sind. Das heißt, dass StepDefinitions, Szenarien oder ganze Feature nur ausgeführt werden, wenn die Ausführungseinheit sich im gleichen Geltungsbereich befindet. Geltungsbereiche werden als String angegeben und können nur für einzelne Methoden gelten oder für eine komplette Klasse. Der Vorteil von Scoped Bindings ist, dass syntaktisch gleiche Steps innerhalb eines Projektes voneinander unterschieden werden können. Ein Beispiel hierfür wäre eine Suche die jeweils anders reagiert je nachdem in welchem Kontext die Anwendung sich befindet. Ausschnitt 2.11 veranschaulicht dies in C#-Code.

Quelltextausschnitt 2.11: Scope Beispiel

```
[When(@"I perform a simple search on '(.*)'", Scope(Tag = "controller")
    )]
public void WhenIPerformASimpleSearchOn(string searchTerm)
{
    var controller = new CatalogController();
    actionResult = controller.Search(searchTerm);
}

[When(@"I perform a simple search on '(.*)'"), Scope(Tag = "web")]
public void PerformSimpleSearch(string title)
{
    selenium.GoToThePage("Home");
    selenium.Type("searchTerm", title);
    selenium.Click("searchButton");
}
```

Der Step („When I perform a simple search on <something>") ist in beiden Fällen syntaktisch gleich, wie die Annotation der Methoden zeigt. Der Methodenrumpf zeigt allerdings, dass semantische völlig unterschiedliche Testschritte durchgeführt werden. In der ersten Methode wird in ein Katalog mit Hilfe des entsprechenden Controllers durchsucht, im zweiten wird eine WebSite angesteuert um die Suche dort abzuschicken. Scopes geben hier die Möglichkeit, dass jeweils die richtige Methode bei der Ausführung der Steps gewählt wird.

Zu beachten ist jedoch, dass Steps so formuliert werden sollten, dass sie unabhängig von bestimmten Geltungsbereichen ausgeführt werden können. Dies gilt nicht nur zwischen zwei Steps sondern auch zwischen Steps und einem Feature. Dementsprechend sollte nur in Notfällen auf diese Art des Bindings zurückgegriffen werden.

2.2.5.3. Integration

Um den Entwicklungsprozess mit Hilfe von VS weiter zu unterstützen, wurde SpecFlow mit einem mächtigen Werkzeug zur Protokollerzeugung ausgestattet um verschiedenste Berichte für die Tests zu generieren. Ebenso werden verschiedene automatisierte Integrationsserver wie etwa der Team Foundation Server unterstützt der die Ausführung der Tests automatisiert übernehmen kann.

Dies schließt die Vorstellung von SpecFlow ab. Im folgenden wird kurz der Unterschied von Akzeptanz- und Unittests beschrieben. Danach wird der empirische Teil der Arbeit vorgestellt.

2.3. Akzeptanztests vs. Unittests

Im Grunde liegt der Unterschied zwischen Akzeptanz- und Unittests darin, an wen die Tests gerichtet sind. Unittests ermöglichen es dem Entwickler sicherzustellen, dass er die Anwendung richtig implementiert. Akzeptanztests sichern ab, dass die Anwendung das richtige leistet.[WH12, S.4]

```
It's sometimes said that unit tests ensure you build the thing right,
while acceptance tests ensure you build the right thing.
```

Um dies besser zu veranschaulichen wird kurz auf ein Beispiel eingegangen:

Wird ein Taschenrechner entwickelt so gibt es verschiedene Betrachtungsweisen. Für den Anwender des Taschenrechners ist es lediglich von Bedeutung, dass er Zahlen eingeben und die verschiedenen Operationen ausführen kann und natürlich das Ergebnis stimmt. Ein möglicher Test für die Addition wäre:

```
Given no previous insertions
When I enter 5+5
And I hit "="
Then the display shows me 10
```

Damit wird ausschließlich die Funktionalität getestet. Es entspricht einem BlackBox Testverhalten da lediglich von außen darauf zugegriffen wird und die internen Vorgänge nicht von Belang sind. Lediglich die Erfüllung des Testinhalts ist von Bedeutung. Ebenso ist es wichtig, dass der Anwender den Vorgang versteht und nachvollziehen kann ohne technische Aspekte des Taschenrechners zu verstehen. Ob die Operanden auf einen Stapel gelegt werden müssen um die Ausführung zu ermöglichen sollte kein erforderliches Wissen des Anwenders sein.

Für einen Entwickler des Taschenrechners ist natürlich auch das korrekte Ergebnis von Interesse, allerdings steht der technische Aspekt im Vordergrund. Unittests ermöglichen es dem Entwickler die Interna seiner Anwendung zu testen. Damit ist das Ziel von Unittests, dass Fehler im Code und Fehlentscheidungen im Design entdeckt werden. Die Tests überprüfen damit im wesentlichen den richtigen Ablauf des Programms und weniger welche Funktionalität für den Endanwender dabei herauskommt. Ob die Anwender die gewünschten Funktionen bekommen, kann damit nicht sichergestellt werden, lediglich, dass die gelieferten Funktionen keine Fehler enthalten.

2.4. Empirie

Um die eingangs gestellten Fragen zu beantworten, muss eine empirische Untersuchung stattfinden. Empirie beschreibt dabei die auf wissenschaftlichem und methodischem Wege

Tabelle 2.1. Unterscheidung empirischer Studienarten

Studienart	Studienfrage	Kontrolle benötigt?	Randomisierung nötig?
Feldexperiment	wie, warum?	ja	ja
Kontr. Experiment	wie, warum?	ja	ja
Umfrage	wer,was,wo..?	nein	ja
Metastudie	wie viele?	nein	nein
Fallstudie	wie, warum?	nein	nein

gewonnene Erfahrung. Dieser erkenntnistheoretische Ansatz steht oftmals im Gegensatz zu einem Großteil der restlichen Ausbildung eines Informatikers, welche mathematisch und analytisch fokussiert ist.

Trotzdem ist die Informatik keine rein theoretische Wissenschaft. Die Überschneidungen mit wirtschaftlichen Interessen bei der Umsetzung von Informationssystemen, erzwingt die Auseinandersetzung mit Themen, die nicht mit rein analytischen Verfahren bewältigt werden können. Da der Großteil der Arbeit eins Informatikers einen direkten oder indirekten Einfluss auf Menschen hat, ist beispielsweise eine Kosten-/Nutzenrechnung bei jedem Projekt von Bedeutung. Vor- und Nachteile verschiedener Verfahren müssen nicht nur theoretisch belegt sein, damit Projektleiter diese akzeptieren. Empirische Untersuchungen, ob die Vor- und Nachteile auch in der Realität erreichbar sind, müssen unternommen werden.

2.4.1. Studienarten

In der empirischen Softwaretechnik gibt es eine Reihe von Studienarten die genutzt werden um ein Phänomen zu erkunden. Hierbei werden Methoden aus der klassischen empirischen Sozialforschung genutzt um die Theorien zur Erklärung der Phänomene zu testen und anschließend zu bewerten. Mögliche Studienarten für diese Arbeit sind:

1. Feldexperiment
2. Kontrolliertes Experiment
3. Umfrage
4. Metastudie
5. Fallstudie

Um die Entscheidung zu begründen warum eine Fallstudie gewählt wurde um das Thema der Arbeit zu untersuchen, folgt hier eine kurze Übersicht über die anderen möglichen Arten und ihre Vor- und Nachteile. Darauf folgt eine kurze Bewertung aller Arten im Bezug auf die hier behandelte Situation und die Begründung, warum sie nicht gewählt wurden. Anschließend daran wird die Fallstudie vorgestellt.

Als Kurzreferenz bietet Tabelle 2.1 eine Übersicht.

2.4.1.1. Feldexperiment

Das Feldexperiment wird wie die Fallstudie in einer realen Umgebung durchgeführt. Dabei werden die unabhängigen Variablen, zumeist nur eine, im Experiment variiert, wobei alle anderen Eigenschaften, die sogenannten Störvariablen, möglichst nicht berührt werden um sie konstant zu halten. Darauf folgend wir der Einfluss dieser veränderten unabhängigen Variablen auf die abhängigen Variablen beobachtet.

Die wichtigsten Anwendungsgebiete sind dadurch beschrieben, dass die Situation nicht im Labor realistisch nachgestellt werden kann und die notwendigen Daten nur in der Praxis

erhoben werden können. Die beteiligten Versuchspersonen sollten kein Wissen über ein stattfindendes Experiment haben, um eine natürliche Reaktion auf die Veränderungen der unabhängigen Variablen sicherzustellen. Ebenso werden im Feldexperiment die Versuchspersonen randomisiert gewählt.

Der große Vorteil des Feldexperiment sind die realistischen Ergebnisse die beim Vergleich mit einer Kontrollstudie produziert werden können. Die Nachteile umfassen einen langen Beobachtungszeitraum und die unter anderem damit verbundenen hohe Kosten. Ebenso ist die Unterstützung des Managements für den Erfolg unerlässlich.

2.4.1.2. Kontrolliertes Experiment

Im wissenschaftlichen Experiment wird ähnlich wie im Feldexperiment planmäßig eine Variation der unabhängigen Variablen durchgeführt. Dabei werden auch hier die abhängigen Variablen möglichst objektiv beobachtet. Im Gegensatz zum Feldexperiment allerdings können hier die Störvariablen, die auch einen Einfluss auf die abhängigen Variablen haben könnten, genauestens kontrolliert werden. Durch ein Konstanthalten dieser Variablen kann deren Einfluss auf ein absolutes Minimum beschränkt werden.

Da das Experiment in einem Labor stattfindet, können die Abläufe wiederholt werden und in weiteren Experimenten bestätigt werden. Bei gleichzeitigem verwenden von Kontrollgruppen können somit genauestens Ursache-Wirkung-Beziehungen beobachtet werden, was die Qualität der Ergebnisse immens erhöht.

2.4.1.3. Umfrage

Bei der Umfrage werden einzelne Mitglieder einer bestimmten Zielgruppe befragt um Informationen zu einem Sachverhalt zu erhalten. Dies gibt einen Einblick in den momentanen Zustand der Zielgruppe. Ziel ist es hierbei möglichst viele Datenpunkte aus der untersuchten Gruppe zu erhalten. Dabei werden standardisierte Fragen an alle beteiligten gestellt, die mit einer Skala bewertet werden müssen. Hierbei können die Fragen sowohl objektiv als auch subjektive Sachverhalte betreffen. Die Antworten darauf sind allerdings immer subjektiv und damit nur eingeschränkt überprüfbar.

Der größte Vorteil dieser Methode sind die geringen Kosten. Vor allem im schriftlichen Fall können die Fragen online gestellt und beantwortet werden. Aber auch im mündlichen Fall ist der Zeitaufwand im Vergleich zu allen anderen Methoden sehr gering. Der große Nachteil ist, dass die Ergebnisse durch die vielen möglichen Störvariablen und subjektiven Einschätzungen meist nicht sehr verlässlich sind.

Diese Methode wird meist als Vorstudie genutzt um erste Ergebnisse für ein folgendes Experiment oder eine Fallstudie zu bekommen, die wiederum die Qualität der folgenden Studien verbessert.

2.4.1.4. Metastudie

Bei der Metastudie werden bereits vorhandene Studien zum untersuchten Thema analysiert und bewertet. Dadurch erhebt die Metastudie selbst keine Primärdaten. Es handelt sich hierbei nicht um eine schlichte Zusammenfassung der vorhandenen Ergebnisse sondern tatsächlich um eine Analyse und einen Vergleich zwischen den vorhandenen Studien. Die Informationen werden verdichtet und bieten dadurch wertvolle Orientierungen und zusammengefasstes Wissen. Dabei können noch nicht erforschte Aspekte aufgedeckt werden oder widersprüchliche Ergebnisse in den Fokus gerückt werden um weitere Forschung anzustoßen.

Durch die fehlende Erhebung von Primärdaten ist der Aufwand geringer als bei den meisten anderen Studienarten. Allerdings können keine Mängel in den vorhandenen Studien

ausgeglichen werden und sind damit automatisch auch Teil der Ergebnisse der Metastudie. Ebenso kann die Technik nur bei genügend bereits vorhandener empirischer Studien verwendet werden.

2.4.1.5. Ausschluss der Studienarten

Nach diesem kurzen Überblick über die möglichen Arten wird nun der Ausschluss der Methoden erklärt. Die Verwendung eines Feldexperiments fällt dadurch weg, dass in diesem Fall keine zufallsgenerierte Gruppe von Versuchspersonen möglich ist. Die beteiligten Softwareentwickler und Chemiker sind von Agilent vorgegeben und können aufgrund der Größe nicht in Gruppen relevanter Untersuchungsgröße unterteilt werden. Ein wissenschaftliches Experiment kommt deshalb ebenso nicht in Frage. Ohne vorhandene Kontrollgruppe liefert das kontrollierte Experiment keine verlässlichen Daten und die Wiederholbarkeit ist aufgrund der besonderen Vertrautheit mit den Akzeptanztests innerhalb der Gruppe nicht gegeben. Eine Umfrage liefert ebenso keine verlässlichen Ergebnisse, da hier eine nur sehr kleine Gruppe untersucht wird. Um relevante Ergebnisse zu liefern müsste der Personenkreis deutlich größer sein um überhaupt Repräsentanten der Gruppe auswählen zu können, die an der Umfrage teilnehmen. Die Metastudie ist aufgrund fehlender Studien zum untersuchten Thema nicht möglich. Eine Erhebung von Primärdaten ist deshalb unerlässlich.

Die letzte verbleibende Methodik ist damit die Fallstudie. Im Nachfolgenden wird zunächst die Methode selbst vorgestellt. Anschließend daran wird geklärt wie die Fallstudie zur gegebenen Situation passt.

2.4.2. Fallstudie

Yin definiert die Fallstudie wie folgt[Yin14, S. 18]:

```
A case study is an empirical phenomenen inquiry that investigates a
contemporary phenomenen in depth and within it's real-life context,
especially when the boundaries between phenomenon and context are not
clearely evident.
```

Demzufolge wird die Fallstudie verwendet, wenn der Kontext nicht vom untersuchten Phänomen getrennt werden kann, da er ein wichtiger Bestandteil der Untersuchung darstellt. Dies steht im starken Gegensatz zum Experiment das die Trennung von Kontext und Phänomen benötigt.

Fallstudien sind das Mittel der Wahl, wenn folgende Bedingungen gegeben sind[Yin14, S. 2][RHRR12, S.19]:

1. "Wie" oder "warum"-Fragen werden gestellt.
2. Der Experimentator hat wenig Kontrolle über die Ereignisse.
3. das Forschungsthema ist nur im echten Leben untersuchbar und findet aktuell statt.

Wichtig ist hier der Fokus auf die Ereignisse im echten Leben, die nicht in einem Labor nachgestellt werden können. Damit erlaubt die Fallstudie eine gesamtheitliche Betrachtung der Ereignisse und damit verbunden liefert sie aussagekräftige Ergebnisse. [Yin14, S. 4]

2.4.2.1. Aufbau

Für eine Fallstudie sind insbesondere fünf Komponenten wichtig. Sie müssen im Vorfeld behandelt werden um eine qualitativ hochwertige Studie durchführen zu können, deren Ergebnisse eine Aussagekraft haben. Die Komponenten sind[Yin14, S. 27]:

1. Die Frage der Studie
2. Die verbundenen Theorien, falls vorhanden
3. Die Analyseeinheiten
4. Die Logik welche die Daten mit den Hypothesen verbindet
5. Die Kriterien zur Interpretation der Untersuchungsergebnisse.

Im folgenden werden die Komponenten detailliert erläutert.

Studienfrage

Bei der Studienfrage ist es wichtig, dass sie das Thema richtig hinterfragt. Für eine Fallstudie sollte sie wie bereits zuvor erwähnt eine wie- oder warum-Frage sein. Die Frage ist der Ausgangspunkte der Studie und damit von enormer Bedeutung. Sie definiert die Hypothesen die in Frage kommen können und bestimmt welche Analyseeinheiten von Relevanz sein könnten. Damit ist die Formulierung der Studienfrage sowohl für die Auswahl, als auch für die Durchführung der Studie von enormer Bedeutung.

Hypothesen

Mit dem Aufstellen einer Hypothese wird bereits am Anfang versucht, die Fallstudie in eine richtige Richtung zu lenken. Durch die Hypothese wird bestimmt, welche Analyseeinheiten untersucht werden und welche Variablen genauer beobachtet werden. Der Konterpart dazu ist die Gegenhypothese, welche das genaue Gegenteil annimmt. Sie ergibt sich damit quasi automatisch sobald die Hypothese aufgestellt wurde. Ebenso von Relevanz ist hier die Nullhypothese, die ebenfalls automatisch gegeben ist und die Möglichkeit beschreibt, dass weder Hypothese noch Gegenhypothese bei der Auswertung wahr ist. Sie ist durch das Ausbleiben von beobachtbaren Änderungen gekennzeichnet. Eine Untersuchung kann auch mehrere Hypothesen gleichzeitig aufstellen und versuchen sie innerhalb der Studie zu beantworten.

Ebenso ist es möglich gar keine Hypothesen aufzustellen. Bei diesen Untersuchungen geht es eher darum das betroffene Forschungsgebiet besser kennen zu lernen und zu erkunden. Erst danach werden entsprechende Studien in Angriff genommen um die einzelnen Themen näher zu beleuchten. Nichtsdestotrotz sollte auch bei „erkundenden Studien" darauf geachtet werden, dass sie ein Ziel haben um die Analyseeinheiten und die Datenerhebung in eine richtige Richtung zu lenken.

Analyseeinheiten

Die Analyseeinheiten sind die Personengruppen oder auch einzelne Personen die von besonderem Interesse für die Fallstudie sind. Es können aber ebenso relevante Dokumente, Quelltexte oder Objekte wie beispielsweise Gemälde sein die untersucht werden. Im Laufe der Studie müssen im Falle von Menschen diese befragt und beobachtet werden, um die notwendigen Primärdaten für die Arbeit erheben zu können. Hier wird ebenso definiert und abgegrenzt, welche Personen nicht Teil der Untersuchung sind. Das betrifft alle Personen die zwar Teil des Kontextes der Untersuchung sind, aber keinen besonderen Einfluss auf die Ergebnisse haben.[Yin14, S. 31]

Hier sollte ebenso daran gedacht werden, eine Zeitspanne zu definieren in der die unterschiedlichen Gruppen und Objekte untersucht werden. [Yin14, S. 32] Ein Abklären des Zugangs zu allen Untersuchungsobjekten im Vorfeld ist von erheblicher Wichtigkeit.

Verbindungslogik

Die notwendige Logik, welche die Daten mit den Hypothesen in Zusammenhang bringt, kann teilweise schon im Vorfeld geklärt werden. Der Durchführende sollte bereits zu Beginn grob wissen, welche möglichen Ergebnisse gefunden werden können und ob diese die Hypothese bestätigen oder dem Theorem widersprechen. Eine genaue Beschreibung aller Möglichkeiten ist hier nicht möglich, aber eine Vorstellung sollte vorhanden sein.[Yin14, S. 34]

Interpretationskriterien

Die Interpretationskriterien zur Bewertung der Untersuchungsergebnisse sollte schon im Vorfeld grob geklärt werden. Damit ist gemeint, dass eine Vorüberlegung stattfinden sollte welche möglichen Ergebnisse zu welchen Schlüssen führen können. Ebenso werden hier rivalisierenden Hypothesen von der eigentlichen Hypothese abgegrenzt.[Yin14, S. 34f]

2.4.2.2. Einzelfallstudie

Abgesehen von den fünf bereits vorgestellten Entwurfsfragen ist bei einer Fallstudie noch zu entscheiden wie viele Fälle betrachtet werden. Durch die besonderen Umstände innerhalb dieser Diplomarbeit fällt die mögliche Betrachtung von Mehrfallstudien weg, weshalb nur Einzelfallstudien kurz erläutert werden.

Wie der Name schon beschreibt, wird bei einer Einzelfallstudie nur ein einziger Fall untersucht. Dabei wird eine im Vorfeld formulierte Theorie getestet, welche nur in einem extremen oder einzigartigen Kontext untersucht werden kann. [Yin14, S. 47] Der Kontext kann so komplex sein, dass er nicht in anderen Fällen gefunden werden kann und kann deshalb nicht in mehreren Fallstudien untersucht wird. Die Problematik hierbei ist, dass die Vergleichspunkte mit anderen Studien schwieriger zu finden sind. Dadurch kann nur schwer eine allgemeine Aussage über das Phänomen getroffen werden, ohne sich der Kritik zu stellen eine zu spezielle Umgebung zu benötigen um die Ergebnisse wiederholen zu können. Mehrfallstudien sind gegenüber dieser Problematik robuster.

Nichtsdestotrotz sind Einzelstudien der Normalfall bei einer Untersuchung und auch ein anerkanntes Mittel der Wahl zur Erklärung eines Phänomens. Bei der Einzelfallstudie werden aufgrund der Analyseeinheiten auch oft sogenannte „eingebette Fallstudien" verwendet. Hierbei handelt es sich zwar um einen einzigen Fall, jedoch wird dieser nochmals in kleine Unterstudien unterteilt. Damit wird dem Unterschied zwischen den verschiedenen Analyseeinheiten Rechnung getragen, die eventuell sehr unterschiedlich beobachtet und bewertet werden müssen. Im Gegensatz dazu steht die „ganzheitliche Fallstudie" die diese Unterteilung nicht vornimmt.

2.4.2.3. Qualitätsmerkmale

Für die Qualität des Untersuchungsergebnisse trägt die Vorbereitung eine wichtige Rolle. Umso besser die Studie geplant und dann auch entsprechend durchgeführt wird, umso weniger Angriffsfläche bieten die davon abgeleiteten Interpretationen für Mängel oder Fehler. Dabei gilt für alle empirischen Studien, dass vier Qualitätsmerkmale optimiert werden müssen. Es handelt sich dabei um[Yin14, S. 40ff]:

1. Konstruktionsvalidität
2. Interne Validität
3. Externe Validität
4. Zuverlässigkeit

Die vier Merkmale werden im folgenden kurz vorgestellt.

Konstruktionsvalidität

Bei der Konstruktionsvalidität geht es darum, dass es klar ersichtlich sein sollte, welche Ziele erreicht werden sollen und wie diese an entsprechende Beobachtungen geknüpft sind. Hierfür ist es sinnvoll mehrere Quellen innerhalb der Studie zu untersuchen. So sind genügend Informationen vorhanden um die gezogenen Schlüsse zu untermauern. Hierzu zählt auch, dass bereits bekannte Mängel bei der Datenerhebung in anderen Studien zum gleichen Thema diskutiert und bewertet werden, falls sie in der eigenen Studie verwendet werden.

Als Beispiel soll hier die Kriminalitätsrate in Städten dienen. Es ist bereits im Vorfeld bekannt, dass die Kriminalitätsrate nicht allein durch die Polizeiberichte beobachtet werden kann da ein Teil der Verbrechen dort nicht gemeldet wird. Deshalb sollten hier weitere Informationsquellen mit hinzugezogen werden um die Daten zu verbessern. Ebenso sollte darauf geachtet werden auf die Problematiken mit den verschiedenen Quellen einzugehen und diese entsprechend zu gewichten.

Eine weitere Möglichkeit, die Konstruktionsvalidität zu erhöhen, ist es eine Beweiskette darzulegen. Dies beschreibt eine zugrundeliegende Logik der ein Leser der Studie folgen kann um von der ursprünglichen Frage bis zu den Ergebnissen zu gelangen. Es sollte klar ersichtlich sein welche Daten zu welchen Ergebnissen führen.

Ebenso sollte sichergestellt sein, dass die Primärdaten ohne subjektive Interpretation im Bericht der Fallstudie auftauchen damit andere diese ebenso nachverfolgen können.

Als letzter möglicher Ansatzpunkt, um die Konstruktionsvalidität zu erhöhen, bietet sich die Durchsicht des Berichts durch entsprechend geschulte Personen oder Personengruppen an. Diese sollten sicherstellen, dass alle vorher genannten Prinzipien eingehalten wurden um ein nachvollziehbares Ergebnis zu erhalten.

Interne Validität

Die interne Validität ist immer dann gefährdet, wenn Schlüsse gezogen werden ohne die Gesamtheit zu überblicken. Hierzu zählt das Vertrauen auf Aussagen aus Interviews das eine bestimmte Relation vorhanden ist oder eine Situation vorgekommen ist, die nicht direkt von einem Beteiligten der Studiendurchführung beobachtet werden konnte. Werden diese Aussagen nicht sehr kritisch betrachtet und mehrfach abgesichert durch beispielsweise unterschiedliche Personenbefragungen oder anderen Dokumenten die den Vorgang belegen, ist die interne Validität gefährdet.

Ebenso kann die Studie angegriffen werden, wenn fälschlicherweise Relationen zwischen zwei Ereignissen aufgedeckt werden, obwohl diese nicht unmittelbar miteinander verbunden waren. Wenn ein nicht abgeklärter Faktor die Änderungen eventuell herbeigeführt wird, kann die Arbeit inhaltlich angegriffen werden.

Externe Validität

Externe Validität beschreibt die Problematik, dass Studienbeobachtungen innerhalb eines einzelnen Falles nicht auf die Gesamtheit übertragbar sind und keine allgemeinen Aussagen treffen können. Deshalb kann die externe Validität vor allem durch die Durchführung gleicher Fallstudien innerhalb eines ähnlichen Kontextes verbessert werden.

Zuverlässigkeit

Bei der Sicherstellung, dass eine Studie zuverlässige Ergebnisse liefert, geht es darum, einen möglichst genauen und nachvollziehbaren Versuchsaufbau zu beschreiben. Dies ermöglicht es anderen Forschern die gleichen Schritte in einem anderen Kontext einzuleiten und die Ergebnisse gut vergleichen zu können. Das Ziel ist es, dass andere Forscher zu den gleichen Ergebnissen gelangen, wenn sie dem Vorgehen der ersten Studie folgen.

2.4.2.4. Beweisführung

Um Material für die Fallstudie zu sammeln gibt es unterschiedliche Quellen um Daten zu erheben.[Yin14, S. 101ff]

Dokumentation

Dokumente enthalten sehr vielfältige Informationen und können nach einem ersten Beschaffen der Materialien zu jedem Zeitpunkt untersucht werden. Die Vorteile bei der Untersuchung von Dokumenten ist, dass sie zumeist keine Beeinflussung des Versuchsleiters oder der beobachtenden Personen enthält. Dafür ist natürlich die Sichtweise des Autors der Berichte zu berücksichtigen. Dokumente enthalten oft exakte Daten und Zahlen anhand derer bestimmte Sachverhalte genau nachvollzogen und belegt werden können.

Archivmaterial

Archivmaterial ist oftmals vergleichbar mit den normalen Dokumenten. Der Unterschied liegt beim Autor der Berichte. Archivmaterial beschreibt in den meisten Fällen Informationen die vom Staat oder Organisationen erfasst wurden, wie Kartenmaterial oder statistische Auswertungen. Im Gegensatz zur normalen Dokumentation ist Archivmaterial oftmals in sehr großen Mengen vorhanden und die Schwierigkeit liegt weniger in der Auswertung, sondern in der Auswahl der richtigen Dokumente.

Interviews

Einer der wichtigsten und am häufigsten verwendeten Quellen für Informationen sind Interviews. Fallstudien im besonderen haben oftmals nur über Interviews die Möglichkeit an bestimmte Informationen zu gelangen, da sie sich immer in einem einzigartigen Umfeld befinden. Der größte Unterschied zu Umfragen ist hierbei, dass der Befrager zwar versucht einer gewissen Form zu folgen die im Entwurf festgelegt wurde, aber dennoch auf einen normalen Gesprächsablauf achten muss. Ebenso ist es wichtig, dass die Fragen so gestellt werden, dass der Interviewpartner keine defensive Haltung einnimmt und damit das Ergebnis bewusst oder unbewusst beeinflusst. Bei der Fragestellung sollte ebenfalls darauf geachtet werden, dass die Sicht des Interviewpartners dabei herauskommt und voreingenommen suggestive Fragen nicht die Antwort beeinflussen.

Direkte Beobachtungen

Solange die Fallstudie einen Gegen- oder Umstand der Gegenwart als Thema hat, können die Phänomene direkt beobachtet werden. Bei direkten Beobachtungen muss darauf geachtet werden, dass möglichst wenig Einfluss auf das beobachtete Phänomen genommen wird. Beispielsweise kann auch schon das Dabeisitzen bei Geschäftstreffen zu Veränderungen im Verhalten der Teilnehmer führen. Diese Beobachtungen müssen dann im Nachhinein entsprechend gewichtet werden.

Beobachtungen als Teilnehmer

Beobachtungen als Teilnehmer zu machen ist nur dann möglich, wenn der Versuchsleiter ebenfalls ein Teil der Analyseeinheit ist. Die größte Problematik dabei ist, dass die Berichte dadurch oftmals stark verfärbt sind, da er ein Teil der Gruppe ist. Objektive Betrachtungen sind dabei teilweise nicht mehr durch den Teilnehmer möglich. Auf der anderen Seite können dadurch Daten aus erster Hand produziert werden, die Eindrücke in Sachverhalte ermöglichen die sonst nur schwer zugänglich wären.

Physikalische Artefakte

Bei den Artefakten handelt es sich um Geräte, Werkzeuge oder andere physische Objekte die untersucht werden können. Im allgemeinen wird diese Quellart relativ wenig in Fallstudien verwendet.

3. Fallstudie

3.1. Anforderungen

Im Vorfeld der Arbeit wurden mehrere Treffen mit Agilent vereinbart um den Rahmen der Fallstudie zu bestimmen. Dabei waren sowohl Chemiker, Entwickler als auch das Management vertreten. In Zusammenarbeit wurden erste Eckpunkte definiert, die zu einer Verbesserung bezüglich der Erstellung von funktionalen Tests führen sollten. Die Grundproblematik für Agilent lag in der mangelnden Wiederverwendung der einzelnen Testschritte. Dies hatte mehrere Gründe.

Zum einen ist die Menge an bisher vorhandenen Testschritten unüberschaubar für einen einzelnen Mitarbeiter. Da die Tests von unterschiedlichen Personen geschrieben werden, haben die Mitarbeiter oft nicht einmal genaue Vorstellungen, ob ein Testschritt bereits existiert, wenn sie ihn noch nie zuvor verwendet haben. Und auch im Falle, dass der Test selbst verfasst wurde oder zumindest bekannt ist, ist es zu viel verlangt, alle möglichen Testschritte zu kennen und entsprechend einzusetzen.

Zum anderen wurde das bisherige SpecFlow-Plugin nicht von allen Mitarbeitern maximal effizient verwendet. Anstatt die Autovervollständigung von SpecFlow einzusetzen um vorhandene Steps innerhalb des Systems zu finden, wurde stattdessen auf die Volltextsuche von Visual Studio zurückgegriffen. Wenn die Autovervollständigung genutzt wurde, erfolgte die Eingabe der Suchkriterien nicht stichwortartig sondern großteils satzweise. Das die Volltextsuche SpecFlow vorgezogen wurde, lag vor allem an der schlechten Performance des SpecFlow Plugins bei der Anzeige mehrerer hunderter Treffer.

Diese beiden Punkte konnten theoretisch mit einer Anpassung des SpecFlow Plugins verbessert werden, weshalb es die wichtigste Anforderung seitens Agilent war.

Zusätzlich dazu wurden weitere Funktionserweiterungen besprochen. Hierzu zählt die Suche nach Synonymen innerhalb des Vorschlagsystems. Der Wunsch von Agilent sah vor, dass für alle eingegebenen Wörter auch die Synonyme gesucht werden und für die Autovervollständigung genutzt werden. Da die Testschritte von unterschiedlichen Mitarbeitern entwickelt werden kommt es vor, dass für die gleiche Tätigkeit unterschiedliche Formulierungen verwendet werden. Diese Testschritte zu identifizieren sollte durch die Synonymsuche vereinfacht werden.

Die Verwendung von Stammwörtern war eine weitere Möglichkeit, die bei den Treffen besprochen wurde. Hier waren keine genauen Daten oder Erfahrungen vorhanden die eine

Notwendigkeit dieser Funktion belegt hätten. Aus einem theoretischen Standpunkt heraus wurden einige Szenarien erforscht, in denen diese Funktionen dennoch hilfreich sein könnten. Stammwörter ermöglichen es dem Ersteller eines neuen Tests auf Pluralformen und Konjugation der Wörter zu verzichten. Dies ist vor allem bei irregulären Verben von Bedeutung, bei denen sich unterschiedliche Zeitformen stark unterscheiden. Die entsprechenden Testschritte zu finden, die eine gleiche Tätigkeit beschreiben, ist dadurch erheblich schwieriger, da unter Umständen alle Zeitformen einzeln gesucht werden müssen. Um dies zu vermeiden sollte eine Stammsuche in das Plugin integriert werden.

Eine weitere Anforderung war die Einschränkung der Suchergebnisse aufgrund von Tags. Jedes Szenario innerhalb der Tests von Agilent wird schon beim Erstellungszeitpunkt mit Tags versehen, die aus der Userstory abgeleitet werden. Diese gruppieren die unterschiedlichen Testszenarien in bestimmte Bereiche. Die Tags werden dabei, wie schon im Grundlagenteil erwähnt, über die Szenarien geschrieben und gelten somit für jeden zugehörigen Testschritt. Ein theoretischer Einwand war schon in der Vorbereitungsphase bekannt. Die Testschritte an einzelne Tags zu binden ist laut den Entwicklern von Cucumber ein Anti-Pattern und sollte somit vermieden werden. Es verhindert die Verwendung der Testschritte außerhalb des zugehörigen Kontextes und widerspricht damit der geforderten Unabhängigkeit eines einzelnen Testschrittes vom Rest des Szenarios. Trotz dessen wurde dieses Feature von Agilent als Anforderung festgehalten.

In den ersten Treffen wurde auch klar, dass der Prozess zur Erstellung der Testschritte ebenfalls verbessert werden musste. Hier gab es zunächst keine eindeutige Struktur wie der Vorgang im Normalfall aussieht. Die vorhandenen Werkzeuge wurden nicht entsprechend nutzbringend eingesetzt. Dies lag jedoch nur teilweise an den bereits identifizierten Problemen. Eine nicht zu vernachlässigende Problematik stellte die unzureichende Kenntnis im Umgang mit SpecFlow dar. Der effektive Einsatz des Werkzeugs war durch mangelndes Wissen teilweise nicht möglich. Dies sollte ebenfalls im Laufe der Studie verbessert werden.

3.2. Begründung der Fallstudie

Der Ausschluss der anderen empirischen Studienarten erfolgte bereits in Kapitel 2. Die Begründung, warum eine Fallstudie für die Fragestellung und die verfolgten Ziele innerhalb dieser Arbeit ideal ist, wurde jedoch noch nicht gegeben.

Die einzigartige Situation von Agilent erfordert eine ausführliche Auseinandersetzung mit den vor Ort vorhandenen Gegebenheiten und kann nicht in einem Laborexperiment nachgestellt werden. Das Forschungsthema ist damit nur im echten Leben untersuchbar und findet in der Gegenwart statt.

Die eingangs in der Einleitung gestellten Fragen befassen sich mit den Gründen, wie die Erstellung der Tests im Rahmen der verhaltensgetriebenen Entwicklung verbessert werden kann und passen damit sehr gut zu den typischen Fragestellungen in einer Fallstudie.

Dadurch, dass die Probanden der Studie alle Mitarbeiter von Agilent sind und die Alltagsarbeit Vorrang vor der Studie hat, ist die Kontrolle des Experimentators in diesem Fall sehr begrenzt. Das Management hat zwar bestätigt, dass der Einsatz des veränderten Werkzeugs gesichert ist, jedoch sind alle anderen Variablen nicht unter der Aufsicht des Durchführenden der Studie. So besteht kein Einfluss auf Positionswechsel, Verwendungsdauer eines einzelnen Mitarbeiters oder sonstige weitere wichtige Variablen, die in einem Experiment sehr scharf beobachtet werden müssten.

Aus diesen Gründen bietet sich aus den empirischen Studienarten nur die Fallstudie an, um das Phänomen vor Ort zu untersuchen.

3.3. Aufbau

Wie die Fragen zum Aufbau einer Fallstudie aus dem Grundlagenkapitel für die Situation bei Agilent passen, wird im folgenden erläutert.

Studienfrage

Die Studienfrage ist stark durch die Anforderungen von Agilent selbst geprägt. Es geht darum herauszufinden, wie die Erstellung von Akzeptanztests so verbessert werden kann, dass eine höhere Wiederverwendbarkeit der einzelnen Testschritte erreicht wird.

Ebenso ist für diese Studie von Interesse, ob kleine Verbesserungen an einem Werkzeug zur Testerzeugung die Effektivität der Tester steigern kann.

Hypothese

Die erste Hypothese ist, dass die Erstellung der Tests schneller und effizienter von statten geht, wenn entsprechende Werkzeugunterstützung vorhanden ist und eingesetzt wird. Diese These stützt sich dabei darauf, dass der Aufwand zum Aufsuchen der Testschritte mit den Änderungen am SpecFlow-Plugin theoretisch verringert werden kann.

Die zweite Hypothese ist, dass die Tester, durch die individuellen Anpassungen an ihre Problembereiche, mehr Motivation haben die Werkzeuge häufiger einzusetzen. Da bei Agilent bereits SpecFlow eingesetzt wird, jedoch nicht die erhofften Ziele mitgebracht hat, ist dies ebenso ein möglicher Ansatzpunkt zur Erklärung eventuell auftretender Verbesserungen.

Die dritte Hypothese ist, dass eine höhere Wiederverwendbarkeit der einzelnen Testschritte zu weniger fehleranfälligem Code führt, da weniger redundante Codestellen in den Akzeptanztests vorkommen. Zu viele Tests sind schwieriger zu warten und sind damit potentiell anfälliger für fehlerhaften Code. Bei Tests sollte deshalb genauso wie bei produktivem Quelltext darauf geachtet werden, dass nur die nötigen Tests beibehalten werden.[Bec03, S. 198] Die These kann nur indirekt überprüft werden, indem die notwendigen Änderungen an den Tests untersucht werden, nach dem sie initial erfasst wurden. Tests bei denen viele Steps wiederverwendet wurden, haben eine geringere Fehlerwahrscheinlichkeit, da die vorhandenen Steps bereits von anderen Tests auf Korrektheit überprüft wurden. Umso mehr vorhandene Steps in einem neuem Akzeptanztest verwendet werden, umso weniger Neuimplementierung ist von Nöten und damit verbunden wird die Anzahl an notwendigen Änderungen geringer.

Analyseeinheiten

Die Einheiten der Analyse innerhalb der Fallstudie sind die bei Agilent arbeitenden Chemiker und Softwareentwickler. Der Fokus liegt dabei auf den Chemikern, da sie die hauptsächlich die Erstellung der Tests vornehmen. Die Chemiker sind, abhängig von den Teams, auch die einzigen Mitarbeiter die aktiv mit der Autovervollständigung von SpecFlow arbeiten, um bereits vorhandene Testschritte wiederzufinden.

Aufgrund der Situation bei Agilent sind jedoch auch die Entwickler stark involviert, da sie teilweise zusammen mit den Chemikern oder auch komplett alleine die Erstellung der Tests übernehmen. Die Softwareingenieure sind in jedem Fall Nutznießer einer Verbesserung des Erstellungsprozesses, da sie unmittelbar auf den Testschritten ihre Implementierungen aufbauen. Dadurch sollte auch für sie eine spürbare Verbesserung auftreten, sobald die Wiederverwendung der Testschritte erhöht wird.

Verbindungslogik

Durch den dualen Ansatz einer qualitativen als auch quantitativen Erhebung von Primärdaten, ist es nötig eine Verbindungslogik beim Start der Fallstudie zu definieren. Dies ermöglicht eine Fokussierung auf die notwendigen Daten, die gesammelt werden müssen. Hierfür wird definiert, dass es zwei unterschiedliche Ebenen gibt in denen die Ergebnisse für einen Erfolg der Fallstudie sprechen können. Zunächst ist es auf der quantitativen Seite möglich einen Rückgang der nötigen Änderungen an einem Test nach dem initialen Eintragen im Versionsverwaltungssystem (VVS) festzustellen. Die Änderungen müssen sich dabei nicht ausschließlich auf eine nachträgliche Verwendung vorhandener Testschritte beschränken. Notwendige Änderungen, die nicht den Datenanteil der Tests betreffen, sprechen für eine schlechtere Struktur des Tests beim Erstentwurf. Die Möglichkeit optimal passende Schritte zu finden, ermöglicht eine bessere Struktur und würde im Falle der eindeutigen Verbesserung nach der Einführung die dritte Hypothese bekräftigen.

Im Bezug auf die qualitative Analyse wurde festgelegt, dass eine Verbindung zwischen notwendiger Arbeitszeit beim Erstellen eines Testskripts und der Unterstützung der Werkzeuge besteht. Ist hier eine Verbesserung nach der Einführung des Werkzeugs feststellbar, so spricht dies für die erste Hypothese.

Unabhängig davon wurden weitere Punkte festgelegt mit denen eine Verbesserung durch die Änderungen im Plugin feststellbar sind. Sollten die Ersteller der Testskripte eine subjektive Verbesserung beim Erstellen der Testschritte feststellen, so wäre dies auch unabhängig von quantitativen Übereinstimmungen ein Erfolg. Die qualitativen Rückmeldungen der Mitarbeiter sind daher auch als entscheidend zu betrachten. Die Antworten der Mitarbeitern können dabei auf alle drei Hypothesen einwirken.

Interpretationskriterien

Die erste Hypothese kann als bestätigt angesehen werden, wenn die qualitative Auswertung dafür spricht, dass ein Großteil (>75%) der Probanden besser mit dem Werkzeug zurecht kam und die Steps einfacher oder schneller gefunden werden konnten.

Die zweite Hypothese kann als bestätigt angesehen werden, wenn ein Großteil der Teilnehmer der Studie das Werkzeug auch nach dem Ende der Fallstudie, weiterhin den Einsatz des Werkzeugs vorsieht. Hier wird ebenfalls davon ausgegangen, dass die Hypothese bestätigt werden kann, wenn mehr als 75% der Teilnehmer das Werkzeug weiterhin einsetzen.

Um zu überprüfen, ob weniger Änderungen von Nöten sind, wird festgelegt, dass dies durch einen statistischen Test nachgewiesen werden muss. Sollte dabei herauskommen, dass die durchschnittliche Änderungsrate verringert wurde und dies mit einem akzeptablen Signifikanzniveau($\alpha <= 0,1$) bestätigt wird, so wird es als Bestätigung der dritten Hypothese bewertet.

3.4. Interview I

Die ersten Interviews wurden vor der Einführung des veränderten SpecFlow-Plugins mit allen beteiligten Mitarbeitern geführt. Darunter befanden sich drei Chemiker und neun Entwickler.

Die Fragen wurden so gewählt, dass eine Bestandsaufnahme der momentanen Situation gemacht werden konnte. Dies betrifft den verwendeten Prozess bei der Erstellung und Implementierung, die eingesetzten Hilfsmittel und den Aufwand für Suche, Änderung und Umsetzung der funktionalen Tests im Team.

Im Laufe der Interviews hat sich herausgestellt, dass die beteiligten Personen sehr unterschiedlich in der Vorgehensweise sind. Die betrifft in erheblichen Maße die Erstellung der

Steps in der Feature-Datei. Ebenso lassen sich deutliche Unterschiede feststellen, in der Art und Weise welche Hilfsmittel benutzt werden um den BDD Prozess zu unterstützen. Die quantitativen Angaben, die von den Befragten gegeben wurden, sind ausschließlich Erfahrungsangaben und wurden nicht speziell mit Hilfsmitteln erhoben. Angaben über eine Spanne an Möglichkeiten, beschreiben jeweils den einfachsten Fall und den schwersten Fall innerhalb der täglichen Arbeit.

Um die Primärdaten innerhalb der Diplomarbeit nicht zu verlieren, werden die Rohdaten mittels einer Tabelle dargestellt. Wichtig ist dabei, dass die Personen je nach Funktion im Team unterschiedlich befragt wurden. Da es auch Unterschiede außerhalb der Kategorisierung Chemiker und Entwickler gibt, unterscheidet sich die Menge an beantworteten Fragen von Person zu Person. Die genaue Fragestellung innerhalb des Interviews wurde dem Gespräch angepasst und entspricht demnach nur sinngemäß der hier gelisteten Frage.[BD06, S. 247] Nach der tabellarischen Darstellung der Antworten wird kleine zusammenfassende Interpretation des Befragers gegeben.

Bei den Interviews wurde darauf geachtet, die Prinzipien einer wissenschaftlicher Befragung einzuhalten. Hierzu zählt ein offener, einladender Stil um Raum für Erklärungen seitens der Experten zu geben.[BD06, S. 244] Das Interesse wurde durch non-verbale Gesten und das Anfertigen einer Mitschrift deutlich gemacht.[Alb09, S. 39] Abgesehen von einer Auseinandersetzung mit der zugehörigen Literatur und einigen Probeläufen mit Unbeteiligten zur Vorbereitung, konnte vom Durchführenden keine zusätzliche Praxiserfahrung gesammelt werden.

Zum Schutze der beteiligten Personen wird in den Tabellen auf eine anonyme Formulierung zurückgegriffen. Die Bezeichnungen „Chem." und „Entw." in den Tabellenköpfen steht jeweils für Chemiker und Entwickler und beschreibt die grundsätzliche Funktion im Team. Zusätzlich wurde die Zuordnung der Antworten zu der Position in der Tabelle zufällig ausgewählt. Dies bedeutet, dass beispielsweise eine geordnete Auflistung aller Inhalte in Spalte 6 untereinander nicht die Antworten ein und derselben Person wiedergibt.

Wie sieht Ihr bisheriger Prozess zur Erstellung eines neuen Testfalles aus?

Da die Antworten auf diese Frage sehr weitreichend waren, können die Daten nicht innerhalb einer Tabelle präsentiert werden. Daher wird alternativ auf eine visuelle Darstellung der Informationen zurückgegriffen. Der Prozess wird in Abbildung 3.1 dargestellt.

Der Prozess zur Erstellung ist bei den meisten Probanden sehr ähnlich und beginnt mit der Userstory. Nach einigen ersten Treffen, in denen besprochen wird, was tatsächlich testbar ist und insbesondere automatisiert funktional testbar ist, wird ein initiales Rumpfscript entworfen. Der erste Entwurf enthält dabei sowohl die Informationen aus den bereits vorhandenen Steps im System, als auch die Ergebnisse aus dem Treffen mit den Entwicklern. Ein Großteil der Chemiker entwirft das Skript dabei komplett autonom. Die andere Möglichkeit ist, dass das Skript mit einem Softwareentwickler abgesprochen wird. Ebenso gibt es Fälle in denen der Entwickler diesen Teil komplett übernimmt.

Danach werden die Testdaten vom Chemiker erstellt. Diese werden teilweise manuell (zum Beispiel mit Hilfe von Excel) errechnet. Dabei wird stark darauf geachtet, dass es möglichst echte Testdaten sind und keine künstlich erzeugten Ergebnisse im Test verglichen werden.

Nachdem die Feature-Datei in das Versionsverwaltungssystem eingetragen wurde, kann der verantwortliche Entwickler mit der Implementierung der StepDefinitions und dem damit verbundenen Produktivcode im System beginnen. Sollten dabei Probleme im Akzeptanztest durch den Entwickler festgestellt werden, wird teilweise nochmals ein Treffen angesetzt um die Fragen zu klären, die beim Softwareentwickler, während der Implementierung aufgetreten sind.

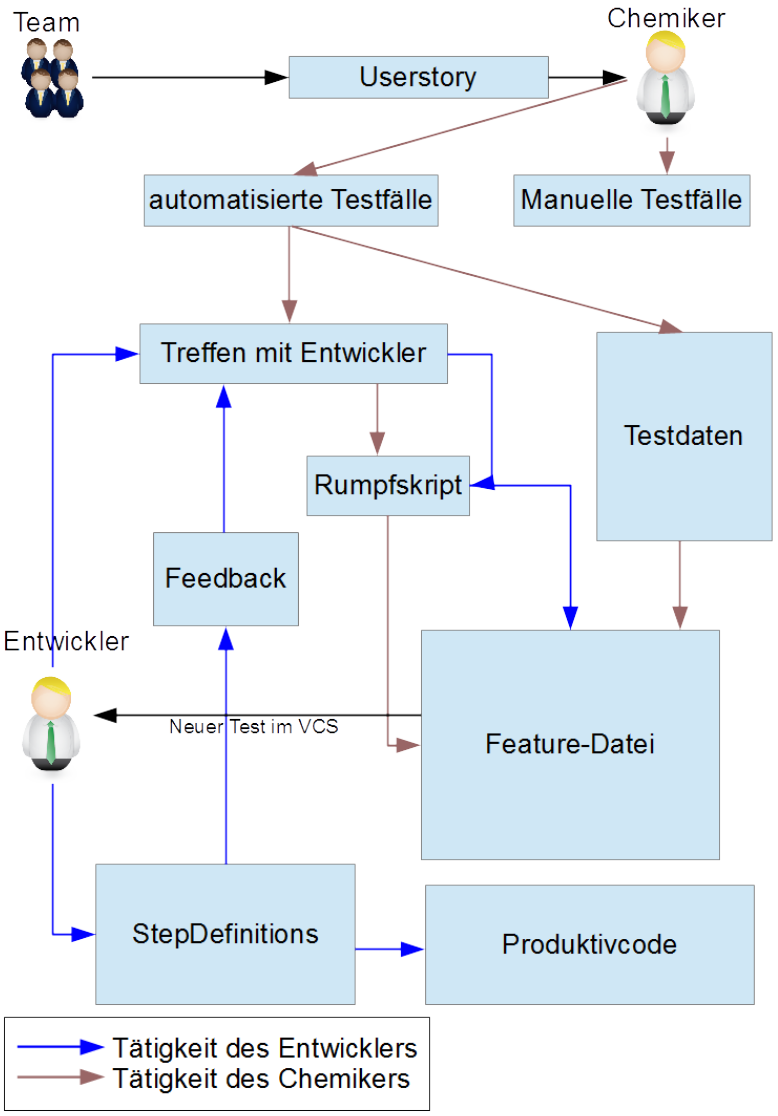

Abbildung 3.1.: Prozessablauf für funktionale Tests Agilent

Nach einer iterativen Verbesserung des Testskriptes und der entsprechenden Implementierung wird der Test in den automatisierten Bau- und Testprozess mit aufgenommen.

Wird das Rumpfscript allein oder gemeinsam im Team entwickelt?

Tabelle 3.1. Antwort auf die Frage: Wird das Rumpfscript allein oder gemeinsam im Team entwickelt?

	Chem.	Chem.	Chem.	Entw.	Entw.	Entw.
Wer?	Gemeinsam	Allein	Allein	Gemeinsam	Gemeinsam	Gemeinsam
	Entw.	Entw.	Entw.	Entw.	Entw.	Entw.
Wer?	Gemeinsam	Allein	Allein	Allein	Allein	Allein

Wie schon im Prozess beschrieben, arbeiten die Entwickler teilweise bei der Erstellung der Rumpfscripte mit. Der größere Teil der Chemiker übernimmt die Skripterstellung alleine

nach dem ersten Treffen mit einem Entwickler. In den anderen Fällen wird das Rumpfskript fast immer gemeinsam von Chemiker und Entwicklern erstellt.

Wird aktiv nach vorhandenen Steps gesucht oder wird das Rumpfscript einfach eingetragen?

Tabelle 3.2. Antwort auf die Frage: Wird aktiv nach vorhandenen Steps gesucht oder wird das Rumpfscript einfach eingetragen?

Methode	Chem.	Chem.	Chem.	Entw.	Entw.	Entw.
Codesuche VS	✓		✓	✓	✓	✓
Autoverv. SpecFlow		✓		✓		
Methode	**Entw.**	**Entw.**	**Entw.**	**Entw.**	**Entw.**	**Entw.**
Codesuche	✓					
Autoverv. SpecFlow	✓					

Um bereits vorhandene Steps innerhalb des Systems zu finden verwendet nur ein einziger Chemiker bereits die Autovervollständigung von SpecFlow. Die anderen nutzen die von Visual Studio angebotene Volltextsuche. Dabei teilweise mit der Einschränkung auf .feature-Dateien.

Bei den Entwicklern benutzen nur wenige die Autovervollständigung von SpecFlow. Diejenigen, die SpecFlow verwenden, benutzen jedoch alle zusätzlich die Volltextsuche von Visual Studio. Für einen Teil der Entwickler ist die Suche nach vorhandenen Steps nicht in ihrem Aufgabengebiet und daher wird keine Variante verwendet.

Benutzen Sie das vorhandene Visual Studio Plugin SpecFlow?

Tabelle 3.3. Antwort auf die Frage: Benutzen Sie das vorhandene Visual Studio Plugin SpecFlow?

Funktion	Chem.	Chem.	Chem.	Entw.	Entw.	Entw.
Autoverv. Vorlagegenerierung	✓	✓		✓		
Navigation		✓		✓		
StepDef-Generierung						✓
Funktion	**Entw.**	**Entw.**	**Entw.**	**Entw.**	**Entw.**	**Entw.**
Autoverv. Vorlagegenerierung	✓	✓		✓		
Navigation		✓				
StepDef-Generierung	✓		✓		✓	✓

Das Plugin war bereits zu Beginn der Studie im Einsatz, jedoch wurde es erstaunlich wenig im Arbeitsalltag eingesetzt. Nur einer der Chemiker verwendet das Plugin regelmäßig um passende Steps im System zu finden, der Rest benutzt die Volltextsuche von Visual Studio. Bei den Entwicklern beschränkt sich die Verwendung hauptsächlich auf die automatische Generierung von StepDefinitions aus der Feature-Datei heraus. Das dieser Anteil hoch ist, war zu erwarten, da dies einen sehr einfachen Ablauf hat und Kopieren&Einfügen-Aktionen verringert. Allgemein scheint die Akzeptanz bei den Entwicklern höher zu sein, was sich darüber erklären ließe, dass das Werkzeug speziell für Visual Studio und damit für Softwareentwickler gemacht wurde.

Welche Probleme sind mit SpecFlow verbunden, die den konsequenten Einsatz verhindern?

Tabelle 3.4. Antwort auf die Frage: Welche Probleme sind mit SpecFlow verbunden, die den konsequenten Einsatz verhindern?

Problem	Chem.	Chem.	Chem.	Entw.	Entw.	Entw.
Performanz	✓			✓	✓	
Ergebnis unzufrieden			✓		✓	
Setupprobleme						✓
Keine Probleme		✓				
Funktion	**Entw.**	**Entw.**	**Entw.**	**Entw.**	**Entw.**	**Entw.**
Performanz						
Ergebnis unzufrieden						
Setupprobleme						
Keine Probleme	✓	✓	✓	✓	✓	✓

Die Probleme mit dem momentanen Stand des Plugins sind unterschiedlich. Ein großer Faktor für diejenigen, die mit Hilfe von Visual Studio gesucht haben, ist die ursprüngliche Performanz des Plugins. Die meisten Entwickler sind sich einig, dass die Suche über Visual Studio schneller ist und SpecFlow dabei keinen Vorteil bietet.

Die meisten Chemiker nutzen das Werkzeug deshalb nicht, weil ihnen die Benutzung schwer fällt. Dies ist teilweise auf fehlendes Wissen im Umgang mit dem Werkzeug zurückzuführen. Dies betrifft beispielsweise die Art der Eingabe bei der Suche mit der Autovervollständigung von SpecFlow. Ein Teil der Mitarbeiter gibt immer komplette Sätze ein und versucht diese als Filter zu verwenden. Das fehlende Wissen, dass diese Art der Suche nicht vom Plugin unterstützt wird, führt zu unbefriedigenden Resultaten bei der Nutzung.

Nutzen sie das Werkzeug des vorherigen Studenten?

Beim Werkzeug des vorherigen Studenten handelt es sich um eine Möglichkeit aktive und passive Schreibweisen innerhalb eines Steps zu erkennen. Dies ist speziell für Agilent entwickelt worden, da die passive Schreibweise oft zu Duplikaten in den Steps geführt hat. Ein Beispiel für die aktive Formulierung wäre:

```
Given that I press the button.
```

Eine entsprechende passive Formulierung wäre:

```
Given the button is pressed.
```

Semantisch sind diese Steps zueinander sehr ähnlich und werden oftmals im Verbindungscode gleichermaßen umgesetzt. Wenn beide Steps innerhalb des Systems vorhanden sind, ist die Gefahr groß, dass notwendige Änderungen an einem der beiden nicht im anderen nachgetragen werden. Das Werkzeug des vorherigen Studenten sollte diese Problematik auflösen, indem die passiv formulierten Steps herausgefiltert werden können.

Um herauszufinden wie stark das Werkzeug eingesetzt wird, wurde danach gefragt, wer es noch in seiner Arbeit aktiv einsetzt.

Da es von keinem im Team eingesetzt wurde, wird hier auf eine tabellarische Darstellung der Antworten verzichtet. Im ersten Team wurde das Werkzeug nicht eingesetzt, da eine einmalige Verwendung für ausreichend angesehen wurde. Während einer Reinigungsphase wurden alle passiv formulierten Steps entfernt und die Erfahrungen daraus reichen aus,

um den Fehler nicht zu wiederholen. Ein Teil der Mitarbeiter wollte das Werkzeug weiter einsetzen, wurde jedoch wegen Fehlern im Programm davon abgeschreckt.

Innerhalb des zweiten Teams gab es keine Kenntnis, dass ein solches Werkzeug existiert.

Welche Verbesserungen wünschen sie sich, damit das Erstellen der Tests besser abläuft?

Tabelle 3.5. Antwort auf die Frage: Welche Verbesserungen wünschen sie sich, damit das Erstellen der Tests besser abläuft?

	Chem.	Chem.	Chem.	Entw.
Vorschlag	Glossar; aktiv/passiv Prüfer	Fuzzy-Suche	Synonymsuche; Duplikate	Duplikate
	Entw.	Entw.	Entw.	Entw.
Vorschlag	Performanz	einheitl. Sprache; Autoverv.*; Ignore-Tag*	einheitl. Sprache; Duplikate	Syntaxherv.*
	Entw.	Entw.	Entw.	Entw.
Vorschlag	Refactor-Hilfe; Tests gruppieren*	weniger Einträge	Testcomplete; UI Verb.	

Die gewünschten Verbesserungen sind vielfältig und spiegeln teilweise die besprochenen Änderungen am Plugin wider. Die Akzeptanz des verbesserten SpecFlows ist dennoch nicht automatisch gegeben, da sehr deutlich sichtbar ist, dass von den Mitarbeitern Änderungen gewünscht wurden, die schon im Originalzustand vorhanden sind (gekennzeichnet in der Tabelle durch *). Ebenso ist die Duplikatproblematik durch das Werkzeug des vorherigen Studenten bereits teilweise aufgelöst und mit Hilfe der bisherigen Autovervollständigung behandelbar. Die notwendige Einarbeitungszeit in die vorhandenen Werkzeuge scheint eine starke Hürde bei der Akzeptanz zu sein. Da die Änderungen an SpecFlow ebenfalls eine Prozessänderung nach sich ziehen werden, ist die Gefahr gegeben, dass damit keine ausreichende Testphase unternommen wird, um die Vorteile des neuen Systems genießen zu können.

Wieviel wird durchschnittlich an einem Test verändert nach dem initialen Einchecken?

Tabelle 3.6. Antwort auf die Frage: Wieviel wird durchschnittlich an einem Test verändert nach dem initialen Einchecken?

Änderungen	Chem.	Chem.	Chem.	Entw.	Entw.	Entw.
Schätzung	50%	<10%	5-10%	20-30%	>50%	30-40%
Änderungen	Entw.	Entw.	Entw.	Entw.	Entw.	Entw.
Schätzung	30%	10%	1%	0%	2-5%	10-15%

Die Prozentangaben geben nur eine grobe Einschätzung der Probanden wieder und wurden nicht speziell von ihnen erhoben. Es sind rein subjektive Schätzwerte.

Bezüglich der notwendigen Änderungen die nach dem initialen Einchecken der Testskripte stattfindet, ergibt sich ein stark unterschiedliches Bild. Im ersten Team werden laut Schätzung pro Test im Mittel etwa 40% der Testschritte geändert. Beim zweiten Team ist die Zahl erheblich geringer. Hier werden deutlich unter 10% der Testschritte verändert. Die Zahlen sind innerhalb der jeweiligen Teams kohärent und bestätigen sich dadurch gegenseitig.

Die große Diskrepanz zwischen den Teams könnte damit erklärt werden, dass eine striktere Trennung zwischen den Aufgabengebieten im zweiten Team stattfindet. Die Aussagen im Interview bestätigen diese Vermutung, da während des Gesprächs mehrfach bestätigt wurde, dass im zweiten Team die Testskripte nicht vom Entwickler überprüft werden. Im ersten Team ist es dagegen üblich, dass die Entwickler selbstständig Testskripte entwerfen und auch die Skripte des Chemikers nochmals auf Fehler überprüft werden. Dies könnte erklären, warum im ersten Team deutlich häufiger Änderungen am Skript vorgenommen werden. Ebenso könnte eine unterschiedliche Ausbildung der Chemiker innerhalb der Teams zu dieser Diskrepanz führen.

Wieviel der Arbeitszeit wird auf das Schreiben von Tests verwendet? Schließen sie dabei das Erstellen und die notwendigen Änderungen mit ein.

Tabelle 3.7. Antwort auf die Frage: Wieviel der Arbeitszeit wird auf das Schreiben von Tests verwendet?

Zeit	Chem.	Chem.	Chem.	Entw.	Entw.	Entw.
Schätzung	k.A.	k.A.	k.A.	1 - 1,5T	1T	2h - 1T
Änderungen	Entw.	Entw.	Entw.	Entw.	Entw.	Entw.
Schätzung	8 - 10h	1 - 2T	1h - 1T	4h - 1T	4h	4h - 2T

Die Zeitangaben der Entwickler zur Implementierung eines einzelnen Testskripts sind sehr unterschiedlich. Im Mittel wird jedoch ein Tag pro Test angegeben. Bei den Feature-Dateien die nur wenige Stunden benötigen, sind fast alle StepDefinitions für die in der Datei angegebenen Steps bereits vorhanden. Dies verkürzt die benötigte Zeit natürlich erheblich. In den Gebieten, in denen bisher kaum Userstories umgesetzt wurden, erhöht sich die benötigte Zeit entsprechend. Deshalb kann die Umsetzung der Testschritte auch teilweise mehrere Tage in Anspruch nehmen.

Zusammenfassung

In den durchgeführten Interviews hat sich herausgestellt, dass es starke Unterschiede zwischen den beiden Teams gibt. Sowohl der Erstellungsprozess als auch die Nutzung der vorhandenen Werkzeuge ist zu großen Teilen sehr unterschiedlich. Dies ist vermutlich auf die Rollenverteilung innerhalb des Teams zurückzuführen.

Das erste Team nutzt einen eher pragmatischeren Ansatz und passt die Vorgaben von BDD ihren individuellen Vorlieben an. Die Trennung von Entwicklern und Testern in ihre jeweils eigenen Aufgabenbereiche ist hier nicht so stark wie im zweiten Team. Die Arbeit des Erstellens eines Tests wird zumeist gemeinsam erledigt und damit auch an die Begrifflichkeiten und Vorstellungen eines Entwicklers angepasst.

Das zweite Team hält sich sehr stark an den Idealprozess, der in Büchern und Vorträgen zu BDD als empfehlenswert dargestellt wird. Die Sicht des Testers wird kaum hinterfragt und erlaubt damit eventuell Formulierungen, die weniger technisch sind.

Der Einsatz der Werkzeuge für das Erstellen von funktionalen Tests ist in beiden Fällen verbesserungswürdig. Dies betrifft nach den ersten Eindrücken nur zu einem Teil die technische Verbesserung des Werkzeugs. Der andere nicht unerhebliche Teil ist die Einführung eines neuen Prozesses, anhand dessen die Auseinandersetzung mit den Werkzeugen stattfinden kann. Einige der bereits vorhandenen Funktionen sind nicht bekannt oder werden nicht auf die vom Ersteller geplante Weise verwendet. Eine reine technische Verbesserung, ohne Veränderung des Prozesses beim Erstellen der funktionalen Tests wird voraussichtlich nicht zu einer Verbesserung führen.

Abgesehen davon haben die Befragungen innerhalb der jeweiligen Teams jedes mal ein kohärentes Bild ergeben. Dies ist für die Bewertung der Ergebnisse von ungeheurer Relevanz, da die Aussagen sich gegenseitig stützen. Damit kann davon ausgegangen werden, dass ein realistisches Bild durch die Befragungen gewonnen werden konnte.

3.5. Einführung des veränderten SpecFlow-Plugins

Die Einführung des verbesserten SpecFlow-Plugins erfolgte unmittelbar vor dem Beginn des nächsten Sprints. Alle Interviews waren bis zu diesem Zeitpunkt bereits abgeschlossen um eventuelle Beeinflussung der Antworten zu verhindern.

Die Einführung beinhaltete eine Vorstellung der Zielsetzung der Fallstudie und die Besprechung der einzelnen Maßnahmen die bereits unternommen wurden. Hierbei wurde zunächst auf die verbesserte Reaktionsgeschwindigkeit bei der Eingabe der Suchkriterien und die Einschränkung der Suchergebnisse eingegangen. Ebenso wurde die Synonymsuche und die Stammwortsuche vorgestellt und anhand von Beispielen erklärt.

Nach der Vorstellung aller technischen Verbesserungen wurde auf den geänderten Prozess aufmerksam gemacht. Der geänderte Prozess sieht vor, dass die Ersteller der Testskripte zunächst einen groben Vorschlag auf Papier oder in Gedanken entwickeln. Danach werden die „bedeutendsten" Verben und Nomen aus den jeweiligen Schritten herausgenommen und einzeln in der Autovervollständigung als Filterkriterium verwendet. Ungewöhnliche oder wenig verwendete Verben enthalten dabei den größten Informationsgehalt und sollte somit als erstes für die Einschränkung der Ergebnisse verwendet werden. Durch die Analyse aller vorhandener Feature-Dateien konnte ermittelt werden, dass Nomen wie „file" oder „method", Verben wie „load" oder „shown" und Artikel beziehungsweise Bindewörter wie „the", „in" oder „a" sehr häufig in den Steps vorkommen. Die Eingabe dieser Wörter hat keinen nennenswerten Informationsgehalt und schränkt die Suche kaum ein. Deswegen sollten besser selten genutzte Verben und Nomen eingeben werden um die Suche initial zu starten. Wird diese Vorgehensweise verwendet, so können innerhalb von nur wenigen Worten eine handvoll von Suchergebnissen angezeigt werden.

Eine der wichtigsten Aspekte bei der Einführung war, zu betonen, dass keine vollständigen Sätze eingegeben werden sollen. Da dies mehrfach innerhalb der Interviews als normale Vorgehensweise beschrieben wurde, bedurfte es hier erheblichem Nachdruck, damit dies nicht weitergeführt wird.

Am Ende des Foliensatzes wurden noch allgemeine Tipps zur Verwendung von Visual Studio und SpecFlow gegeben. Danach erfolgte eine Live-Demonstration des geänderten Werkzeugs anhand derer alle Änderungen mit Beispielen gezeigt wurden.

Nach den Installationsanweisungen wurde Raum für Fragen gegeben.

Die Änderungen wurden von allen zunächst gut angenommen und es gab wenige Fragen zur Funktionsweise. Der wohl wichtigste Beitrag war, dass es in der vorgestellten Version des Werkzeugs nicht möglich ist, einen schnellen Überblick über die Anzahl der gezeigten Suchergebnisse zu bekommen. Dies wurde von allen einstimmig als Problematik anerkannt und entsprechend als Verbesserungsvorschlag für das Werkzeug aufgenommen. Die verbesserte Version mit einer Anzeige, wie viele Ergebnisse in der Liste angezeigt werden, wurde noch vor der tatsächlichen Einführung an den Arbeitsplätzen bereitgestellt.

3.6. Interview II

Nach der Nutzungsphase von insgesamt 6 Wochen, was zwei Sprints entspricht, wurden erneut Interviews mit allen Beteiligten geführt. Hierbei musste leider festgestellt werden,

dass es unabhängig vom Werkzeug erhebliche Probleme gab, die eine Verwertung der Interviewergebnisse erschwert. Es war schon vor der Durchführung klar, dass die relative kurze Nutzungsphase von nur zwei Sprints, nur eine begrenzte Anzahl an neu geschriebenen funktionalen Tests hervorbringen würde. Leider haben weitere externe Faktoren die Lage noch schwieriger gemacht. Hierzu zählen Positionswechsel innerhalb der befragten Teams, Auslandsaufenthalte, wochenlanger Urlaub einzelner Mitarbeiter und technische Probleme. Die technischen Probleme beschreiben dabei nicht Hindernisse bei der Einführung oder Nutzung des veränderten SpecFlow-Plugins, sondern im speziellen eine umfassende Umstellung des Versionsverwaltungssystems bei Agilent. Die notwendigen Anpassungen an ein neues System benötigten zusätzliche Zeit, in der weniger produktiv an der Umsetzung von Userstories gearbeitet werden konnte. Durch die Umstellung und die schon zuvor gelisteten Probleme war es einigen Mitarbeitern nicht möglich mit der Erstellung und Implementierung funktionaler Tests überhaupt anzufangen. Andere konnten nicht so viele Userstories bearbeiten, wie es sonst der Fall gewesen wäre.

Durch die Natur einer Fallstudie ist dies zu einem gewissen Grad zu erwarten, wiegt aber besonders schwer bei einer Einzelfallstudie, bei der nicht auf die Ergebnisse von anderen Untersuchungen zurückgegriffen werden kann. Durch die spezielle Ausrichtung auf Agilents Expertenteam, ist es innerhalb dieser Arbeit nicht möglich weitere passende Datenpunkte zu sammeln.

Trotzdem konnten wichtige Informationen aus den Interviews gewonnen werden, die im folgenden aufbereitet dargestellt werden. Dies geschieht erneut in Tabellenform um die Primärdaten übersichtlich darstellen zu können. Wie schon zuvor sind die hier beschriebenen Fragen nur eine allgemeine Formulierung und spiegeln nicht hundertprozentig den Wortlaut wider, welcher in der Befragung verwendet wurde.

Um die Anonymität der Mitarbeiter von Agilent zu wahren und den Vergleich mit den vorherigen Antworten zu erleichtern, werden alle Personen weiterhin aufgelistet, auch wenn sie keine Antworten auf die Fragen liefern konnten. Wie schon zuvor lassen die Antworten einer Spalte keinen Rückschluss auf ein und dieselbe Person zu.

Haben sie das veränderte SpecFlow-Plugin eingesetzt?

Tabelle 3.8. Antwort auf die Frage: Haben sie das veränderte SpecFlow-Plugin eingesetzt?

Antwort	Chem.	Chem.	Chem.	Entw.	Entw.	Entw.
Ja	✓	✓	✓	✓	✗	✗
Nein/Grund					Extern	Kommunikation
Anzahl Tests	3	2	1	2-3	0	1
Antwort	Entw.	Entw.	Entw.	Entw.	Entw.	Entw.
Ja	✓	✗	✗	✗	✗	✗
Nein/Grund		0 Tests	Extern	0 Tests	Kein Nutzen	Kein Nutzen
Anzahl Tests	1	0	0	0	2	1

Der Anteil der Mitarbeiter die das veränderte Werkzeug eingesetzt haben ist leider gering. Nur fünf der zwölf beteiligten hat es eingesetzt. Der Mehrwert des Plugins ist am deutlichsten für die Mitarbeiter zu spüren, die selbst Tests erfassen. Deshalb haben alle Chemiker es auch eingesetzt. Die Entwickler sahen im Werkzeug größtenteils keinen Vorteil für ihre Arbeit und haben somit nicht den Aufwand auf sich nehmen wollen, den die Installation und eventuell vorhandene Probleme mit sich bringen würde. Dies ist schade für die Auswertung, jedoch auch zu einem gewissen Grad verständlich, da die Verbesserungen nicht direkt an die Entwickler gerichtet waren. Die eventuell verbesserten Testskripte betreffen

die Softwareentwickler nur indirekt, indem die Implementierung der StepDefinitions einfacher fällt. Abgesehen davon, haben die meisten Entwickler keinen direkten Nutzen, wenn sie nicht selbst aktiv nach vorhandenen Steps suchen müssen.

Der Grund „Extern" beschreibt, dass externe Beweggründe den Mitarbeiter davon abgehalten haben das Werkzeug einzusetzen. Dies ist nicht gleichbedeutend mit den übrigen Antworten, die keinen Nutzen gesehen haben. Aus Gründen des Datenschutzes wird auf diese nicht weiter eingegangen.

Gab es Probleme oder Fehler beim Einsatz?

Tabelle 3.9. Antwort auf die Frage: Gab es Probleme oder Fehler beim Einsatz?

Probleme	Chem.	Chem.	Chem.	Entw.	Entw.	Entw.
Nein	✓	✗	✓	✓	-	-
Ja, Art		Konfiguration				
Probleme	Entw.	Entw.	Entw.	Entw.	Entw.	Entw.
Nein	✓	-	-	-	-	-
Ja, Art						

Technische Probleme am Plugin waren kein Hinderungsgrund für die Nutzung der verbesserten Funktionen. Nur für einen Mitarbeiter gab es bei der erstmaligen Installation Probleme bezüglich der richtigen Konfiguration des Plugins. Danach sind auch bei diesem Mitarbeiter keine weiteren Probleme aufgetaucht.

Dies ist deshalb wichtig, als das die Ergebnisse nicht durch technische Widrigkeiten verfälscht worden sind. Dies kann durch die gegebenen Antworten ausgeschlossen werden.

Haben die Veränderungen einen positiven Effekt für das Schreiben der Tests gehabt?

Tabelle 3.10. Antwort auf die Frage: Haben die Veränderungen einen positiven Effekt für das Schreiben der Tests gehabt?

Effekt	Chem.	Chem.	Chem.	Entw.	Entw.	Entw.
Positiv	✓	✓	✓	✓	-	-
Effekt	Entw.	Entw.	Entw.	Entw.	Entw.	Entw.
Positiv	✓	-	-	-	-	-

Von allen Mitarbeitern die das Werkzeug verwendet haben, war die Einschätzung, dass die Änderungen einen positiven Effekt für das Schreiben von Tests hatten.

Dies ist die allgemeine Einschätzung. Es wurden bezüglich jeder der Verbesserung weitere Fragen gestellt, um diese zu bewerten.

Konnten sie vorhandene Steps schneller beziehungsweise einfacher finden?

Tabelle 3.11. Antwort auf die Frage: Konnten sie vorhandene Steps schneller beziehungsweise einfacher finden?

Verbesserung	Chem.	Chem.	Chem.	Entw.	Entw.	Entw.
Schneller	✓	✓	✓	✓	-	-
Zusatz		Zielgerichteter	Angenehmer		-	
Verbesserung	Entw.	Entw.	Entw.	Entw.	Entw.	Entw.
Schneller	✓	-	-	-	-	-
Zusatz	Manchmal					

Diejenigen die mit dem Werkzeug gearbeitet haben, um vorhandene Steps aufzufinden, haben bemerkt, dass sie die Steps einfacher und schneller finden können. Die Suche wurde als zielgerichteter, angenehmer und zeitsparender beschrieben.

Die Entwickler haben keinen nennenswerten Unterschied bei der Implementierung der Steps feststellen können.

Wie war die Performanz des Werkzeuges?

Tabelle 3.12. Antwort auf die Frage: Wie war die Performanz des Werkzeuges?

	Chem.	Chem.	Chem.	Entw.	Entw.	Entw.
Performanz	Besser	Gut	Gut	Kein Unterschied/Problem	-	-
	Entw.	Entw.	Entw.	Entw.	Entw.	Entw.
Performanz	Gut	-	-	-	-	-

Die zuvor vorhandenen Probleme, mit der Performanz des ursprünglichen Plugins, konnten nach den Antworten der Mitarbeiter vollständig beseitigt werden. Niemand hatte eine negative Erfahrung beim Aufsuchen der Schritte aufgrund schlechter Antwortzeiten während der Bedienung. Lediglich ein Entwickler hatte davor kein Problem mit der Performanz zu beklagen und konnte dadurch keinen Unterschied feststellen.

Wie empfanden Sie die Einschränkung der Ergebnisse?

Tabelle 3.13. Antwort auf die Frage: Wie empfanden Sie die Einschränkung der Ergebnisse?

	Chem.	Chem.	Chem.	Entw.	Entw.	Entw.
Gut?	✓	✓	✓	✓	-	-
Besserer Wert?	10	<25		25	-	
	Entw.	Entw.	Entw.	Entw.	Entw.	Entw.
Gut?	✓	-	-	-	-	-
Besserer Wert?	40					

Der initial gesetzte Schwellwert von 25 Suchergebnissen wurde größtenteils gut von den Mitarbeitern bewertet. Keiner der Chemiker hat berichtet, dass die 25 Suchwerte tatsächlich durchsucht wurden. Stattdessen wurde mit Hilfe weiterer prägnanter Nomen und Verben die Suche verfeinert. Bei den Chemikern war daher eher eine Tendenz hin zu geringeren Schwellwerten feststellbar.

Nur einer der Entwickler gab an, dass er lieber mehr Suchergebnisse auf einmal sehen würde.

Wurden Synonyme vom Team eingepflegt?

Hier wird erneut auf die tabellarische Form verzichtet, da die Antworten alle gleich waren. In keinem der beiden Teams wurden zusätzliche Synonyme eingepflegt. Der Konsens war, dass zu wenig Zeit zur Verfügung stand um Synonyme einzupflegen. Es hätte einer designierten Person gebraucht, die die Synonyme zusammenträgt und in das Plugin eingibt.

Wurde die Stammsuche von Ihnen verwendet?

Auch bei dieser Frage waren sich alle Mitarbeiter einig. Die Stammsuche wurde nicht bewusst verwendet.

Der Nutzen dieser Funktion war für die Beteiligten nicht offensichtlich und hatte keine Auswirkungen darauf, ob die Mitarbeiter auf Konjugation der Worte geachtet haben oder nicht.

Welche Verbesserungen können Sie sich am Werkzeug vorstellen, damit das Erstellen der Tests noch einfacher abläuft?

Tabelle 3.14. Antwort auf die Frage: Welche Verbesserungen können Sie sich am Werkzeug vorstellen, damit das Erstellen der Tests noch einfacher abläuft?

Verbesserungen	Chem.	Chem.	Chem.	Entw.	Entw.	Entw.
Vorschlag	-	Syn. nutzen	Link zu Tabelle	-	-	-
Verbesserungen	Entw.	Entw.	Entw.	Entw.	Entw.	Entw.
Vorschlag	Benennung	-	-	-	-	-

Es wurden kaum Verbesserungsvorschläge für das Werkzeug genannt. Eine prägnantere Benennung der Knöpfe, innerhalb der GUI des Autovervollständigungdialogs und Verweise zu den Tabellen die zu den Steps gehören wurden als Verbesserungen vorgeschlagen. Zusätzlich gab es den Wunsch, die verwendeten Synonyme tatsächlich einzupflegen, was allerdings schon als Funktion vorhanden ist. Hier fehlt es nur an der Ausführung der Verantwortlichen im Team.

Es wurde ebenso von etlichen Mitarbeitern geäußert, dass sie noch nicht genügend Zeit hatten, um sich mit dem Werkzeug intensiv auseinander zu setzen. Es kann davon ausgegangen werden, dass weitere Verbesserungsvorschläge im Laufe der weiteren Nutzungsphase auftauchen.

Hat sich der Ablauf beim Erstellen der Tests für Sie verändert?

Da nur eine Person hierauf ausführlicher geantwortet hat, wird auf die tabellarische Darstellung verzichtet.

Der Wunsch wurde geäußert, dass das alte System der Eingabe beibehalten werden könne. Dies bezieht sich darauf, dass im alten Prozess oftmals komplette Sätze als Steps eingegeben wurden. Da die neue Vorgehensweise davon ausgeht, dass nur prägnante Verben oder Nomen als Suchkriterium benötigt werden, bedurfte es einer Umstellung.

Die Umstellung auf diesen neuen Prozess wurde nicht von allen als positiv empfunden und eine bessere Unterstützung des alten Systems wäre für Sie hilfreich gewesen.

Ihrer Einschätzung nach, wie viel wird durchschnittlich an einem Test verändert nach dem initialen Einchecken?

Tabelle 3.15. Antwort auf die Frage: Wie viel wird durchschnittlich an einem Test verändert nach dem initialen Einchecken?

Änderungen	Chem.	Chem.	Chem.	Entw.	Entw.	Entw.
Schätzung	-	10%	10%-100%	20%-30%	-	-
Änderungen	Entw.	Entw.	Entw.	Entw.	Entw.	Entw.
Schätzung	0%	-	-	-	5%	10%

Um eine qualitative Ansicht auf die notwendigen Änderungen nach der Einführung zu bekommen, wurde diese Frage erneut gestellt. Die Spanne der Änderungen ist wie im vorherigen Interview von Team zu Team sehr unterschiedlich. Die Tendenzen bleiben aber in beiden Fällen gleich.

Haben sie Veränderungen an den Testskripten bemerkt, die relevant für die Implementierung waren?

Auch hier wird auf eine tabellarische Darstellung der Antworten verzichtet, da alle Rückmeldungen gleich waren. Kein Entwickler hat irgendeine Veränderung an den Rumpfskripten wahrgenommen.

Ihrer Einschätzung nach, wie viel der Arbeitszeit wird für das Implementieren von Tests verwendet?

Tabelle 3.16. Antwort auf die Frage: Wie viel der Arbeitszeit wird für das Implementieren von Tests verwendet

Zeit	Chem.	Chem.	Chem.	Entw.	Entw.	Entw.
Schätzung	k.A.	k.A.	k.A.	2T	-	-
Änderungen	Entw.	Entw.	Entw.	Entw.	Entw.	Entw.
Schätzung	<1T	-	-	-	3-4h	4h

Die benötigte Zeit für das Implementieren der Tests ist weiterhin sehr unterschiedlich und hängt hauptsächlich davon ab, wie viele Stepdefinitions bereits im Gebiet des Tests vorhanden sind. Allgemein konnte keine nennenswerte Veränderung festgestellt werden, was durch die Antworten auf die vorherige Frage zu erwarten ist.

Sind die Veränderungen an SpecFlow sinnvoll? Werden sie es weiterhin nutzen?

Tabelle 3.17. Antwort auf die Frage: Sind die Veränderungen an SpecFlow sinnvoll und werden sie es weiterhin nutzen?

Nutzen	Chem.	Chem.	Chem.	Entw.	Entw.	Entw.
Vorteil	✓	✓	✓	✓	-	-
Weiterhin verwenden	✓	✓	✓	✓	-	-
Änderungen	Entw.	Entw.	Entw.	Entw.	Entw.	Entw.
Vorteil	✓	-	-	-	-	-
Weiterhin verwenden	✓	-	-	-	-	-

Die Rückmeldungen bezüglich des Nutzen der Änderungen war durchweg positiv. Alle Mitarbeiter, die das Werkzeug eingesetzt haben, waren vom Vorteil gegenüber der Originalversion überzeugt. Ebenso wurde von allen bestätigt, dass sie es für ihre weitere Arbeit, über die Fallstudie hinaus, verwenden werden.

Zusammenfassung

Die eingangs erwähnten Probleme haben leider dazu geführt, dass nicht sehr viele Datenpunkte innerhalb der zweiten Interviewrunde erhoben werden konnten. Dies macht einen Vergleich sehr schwierig. Trotz dessen kann festgestellt werden, dass die Änderungen positive Effekte auf die Belegschaft beim Erstellen von funktionalen Tests hatten. Diese beziehen sich jedoch ausschließlich auf diejenigen, die tatsächlich Feature-Dateien erstellen. Für die Entwickler, welche nur die Implementierung durchführen, konnte absolut keine Änderung festgestellt werden.

Die implementieren Verbesserungen am Plugin haben vor allem die Ausführungsgeschwindigkeit verbessert und somit die Benutzerfreundlichkeit stark erhöht. Alle negativen Aspekte diesbezüglich konnten beseitigt werden. Die Reduzierung der Einträge auf maximal 25 und die ausschließliche Anzeige der StepDefinitions wurde positiv aufgenommen. Die Stammsuche wurde von niemandem aktiv wahrgenommen und die Mitarbeiter haben es nicht als einen Mehrwert betrachtet. Gleiches gilt für die Synonymsuche. Auch diese wurde von keinem Probanden eingesetzt und kann damit nicht bewertet werden. Beziehungsweise muss der Nutzen der Funktion in Frage gestellt werden. Trotz dessen das die Synonymsuche nicht verwendet wurde, gab es die Rückmeldung, dass versucht wird diese in Zukunft einzusetzen und zu pflegen. Ein längerer Beobachtungszeitraum könnte hier Aufschluss darüber geben, ob die Funktion keinen Nutzen hatte oder die verfügbare Zeit keine Verwendung zugelassen hat.

Insgesamt wurden zu wenige Userstories innerhalb der beiden Sprints implementiert um aussagekräftige Daten zu erzeugen. Es kann lediglich festgestellt werden, dass sowohl die Stammsuche, als auch die Synonymsuche keine Notwendigkeit für die Mitarbeiter darstellen. Im Gegensatz dazu wurde die Einschränkung der Suchergebnisse allgemein sehr gut bewertet, da sie scheinbar schneller zum Ergebnis führt.

Der wohl größte Faktor für die Veränderungen ist der neu eingeführte Prozess. Abgesehen von einer Rückmeldung, die sich bessere Abwärtskompatibilität gewünscht hat, wurde der neue Prozess sehr gut angenommen. Das keine vollständigen Sätze mehr geschrieben werden, erleichtert für viele die Suche. Die stichwortartige Eingabe der Suchbegriffe ermöglicht es, dass Testschreiber nur wenige Vorstellungen über die benötigten Schritte haben müssen um mit der Suche zu beginnen. Durch die zusätzliche Einschränkung der Suchergebnisse finden die Nutzer häufiger und schneller die passenden Steps im System.

Die einheitliche Aussage, dass das Plugin auch weiterhin verwendet wird, lässt auf eine nachhaltige Verbesserung durch die Veränderungen an SpecFlow schließen. Die qualitative Bewertung der Modifikationen am Plugin zeigen, dass die Erstellung von funktionalen Tests, mit kleinen Veränderungen an den verwendeten Werkzeugen, verbessert werden kann. Die Annahme der kleinen Veränderungen durch das Team erfolgen leichter. Ebenso ist festzustellen, dass Änderungen, die keinen direkten Einfluss auf die Produktivität des Mitarbeiters haben, nicht gut angenommen werden.

3.7. Analyse des Versionsverwaltungssystem

Um quantitative Daten innerhalb der Studie zu erheben, wurde eine Analyse des Versionsverwaltungssystems (VVS) gemacht. Dies erlaubt eine objektive Betrachtung der notwendigen Änderungen innerhalb der Feature-Dateien. Vor der Einführung von GIT bei Agilent wurde SVN als Versionsverwaltungssystem eingesetzt. Die komplette Historie aller Änderungen wurde von SVN in GIT übernommen und für die Studie bereitgestellt. Daher erfolgte die Analyse über die GIT-Historie.

Aus Gründen der Vertraulichkeit können die funktionalen Tests nicht als Primärdaten an diese Studie angehängt werden. Daher wird nur das Vorgehen bei der Analyse beschrieben und im Anschluss die Ergebnisse präsentiert.

Im Vorfeld gab es einige Konventionen die von Belang sind. Zum einen erstellen die Chemiker oftmals die Feature-Dateien für die Kollegen. Dies geschieht deshalb, weil die Chemiker eine Kontrollfunktion über die Benennung der Feature-Dateien ausüben. Die Userstories werden in einem separaten System gepflegt und mit vorgegebenen Bezeichner befüllt. Diese müssen exakt dem Namen der Feature-Dateien entsprechen und werden somit fast immer vom Chemiker angelegt.

Falls Entwickler die Aufgabe der Erstellung des Rumpfskriptes übernehmen, wird der initiale Commit der Datei vom Chemiker gemacht. Die Datei beinhaltet zu diesem Zeitpunkt nur die Benennung des Szenarios und einer sehr kurzen Beschreibung aus der Userstory. Ein spezieller „@Ignore"-Tag in der Datei verhindert die automatische Ausführung durch den Integrationsserver. Zusätzlich dazu wird ein immer gleicher Inhalt aus einer Vorlage als Steps in die Datei generiert. Diese enthält unter anderem einen Step „Given I have entered 50 into the calculator".

Damit die Änderungen an dieser Vorlage nicht die Daten verfälschen, ist der initiale Commit in solchen Fällen verzögert. Erst der nächste Eintrag, welcher die Steps der Vorlage entfernt, wird als Ausgangszustand betrachtet.

Die Zuordnung, welcher Mitarbeiter den Test ursprünglich erstellt hat, erfolgt aufgrund dieses Ausgangszustands. Damit wird sichergestellt, dass im Falle der reinen Vorlagengenerierung der richtige Verantwortliche für den initialen Commit gefunden wird.

Eine weitere Einschränkung bei der Betrachtung der Änderungen ist der verwendete Zeitraum. Um die Daten besser vergleichen zu können, werden nur Änderungen gezählt, die innerhalb eines dreiwöchigen Zeitraums nach dem initalen Commit stattgefunden haben. Dies ist deshalb wichtig, da die Beobachtungszeit nach der Einführung des veränderten Werkzeugs sehr begrenzt ist. Um die Vergleichbarkeit zu erhöhen wird deshalb nur ein kleiner Zeitraum nach dem Ausgangszustand betrachtet. Zusätzlich wird damit vermieden, dass größere Umbauten an den Tests die erhobenen Daten nicht zu sehr beeinflussen. Trotz dessen ist es möglich, dass innerhalb des untersuchten Zeitraumes größere Umbauten an einigen Feature-Dateien gemacht wurden. Aufgrund der Menge der untersuchten Tests sollte der Einfluss dennoch gering sein. Es bleibt jedoch eine nicht auszuschließende Fehlerquelle, die bei der Betrachtung der Ergebnisse berücksichtigt werden muss.

Weiterhin musste beachtet werden, dass ausschließlich Änderungen an den Steps betrachtet werden. Modifikationen an den Datentabellen fallen, nach Aussagen der Mitarbeiter, viel häufiger an. Diese sind für die Fallstudie nicht von Belang, da sie völlig unabhängig davon sind, ob passende Steps verwendet wurden. Gleiches gilt für Kommentare und Beschreibung. Auch diese haben keinen nennenswerten Informationsgehalt und werden deshalb nicht als relevante Änderung eingestuft. Aus diesen Gründen werden ausschließlich Änderungen an Zeilen betrachtet, die mit einem der möglichen Schlüsselwörter beginnt, die schon in Kapitel 2 vorgestellt wurden. Einzige Ausnahme stellen Zeilen dar, welche auskommentiert wurden, sich jedoch auf einen Step beziehen. Diese werden ebenfalls mit in die Änderungen gezählt.

Nach diesen Ausnahmen wird nun der generelle Ansatz erklärt. Für jede .feature-Datei innerhalb des Versionsverwaltungssystems werden alle Commits betrachtet, die drei Wochen nach dem initialen Checkin stattgefunden haben. Die Differenz zwischen Ausgangszustand und dem Zustand nach drei Wochen wird analysiert und auf Änderungen an Steps untersucht. Die geänderten Zeilen werden gezählt. Dabei wird anders als in GIT gewöhnlich jede Änderung an einer Zeile nur einmal gezählt. Die Modifikation einer Zeile wird nicht als hinzufügen und entfernen von einer Zeile gewertet, sondern als eine einzelne Operation angesehen. Das Hinzufügen einer neuen Zeile oder das Löschen einer bereits vorhandenen wird ebenso jeweils als eine Veränderung gezählt. Die komplette Anzahl der geänderten Zeilen in der Feature-Datei zum Zeitpunkt des ersten echten Commits ist die Basis mit der verglichen wird.

4. Vorgenommene Anpassungen an SpecFlow

SpecFlow wurde an verschiedenen Stellen angepasst. Diese wurden bereits in Kapitel 3 als Anforderungen festgehalten. An dieser Stelle wird genauer auf deren Umsetzung innerhalb von SpecFlow und Visual Studio eingegangen.

4.1. Geschwindigkeit

Die erste Idee bezüglich der Performanzverbesserung war, dass ein ineffektiver Index verwendet wird, um die einzelnen Steps zu halten und dann zu untersuchen. Nach eingehender Recherche konnte diese Behauptung allerdings widerlegt werden. Sowohl der Aufbau des Index als auch die Suche darin erfolgte im ursprünglichen Zustand sehr schnell und der komplette Abgleich benötigte nur einen Bruchteil einer Sekunde. Nach intensiver Fehlersuche konnte dann die Vermutung geäußert werden, dass das Problem bei der Darstellung des Dialoges in MS VS liegt. Eigene Tests als auch die Zuhilfenahme des MS Supports haben bestätigt, dass viele und lange Vorschläge innerhalb der Autovervollständigung die Performanzeinbrüche verursachen.

Eine mögliche Abhilfe dafür ist die Einschränkung der Suchergebnisse. Dies beschränkt zwar die Anzeige aller möglichen Steps die zur bisherigen Eingabe passen würden, löst aber die Performanzprobleme komplett. Der damit verbundene Nachteil ist allerdings nicht besonders schwerwiegend. Mehr als 25 Vorschläge zu überblicken ist für einen Entwickler durchaus schwierig und sollte normalerweise damit behoben werden, dass mehr Informationen eingegeben werden, um die Suche weiter einzuschränken. Man könnte sogar davon ausgehen, dass die Einschränkung dem User eher hilft, nicht völlig überfordert zu sein von der Anzahl der möglichen Einträge. Dazu wurden allerdings keine weiteren Recherchen erhoben. Stattdessen wurde im Vorschlagsdialog der IDE eine Möglichkeit gegeben, die Anzahl der angezeigten Vorschläge selbst zu bestimmen. Dadurch sind alle möglichen Vorschläge einen Mausklick entfernt und der Nachteil verkommt nur noch zu einer minimal verschlechterten Bedienbarkeit des Dialogs.

Entsprechend des Vorschlags in der Einführung wurde darauf geachtet, dass die Einschränkung der angezeigten Suchergebnisse nicht zu einer weiteren Problematik wird. Deshalb wurde ein Zähler in den Dialog eingebaut, der die momentane Anzahl an angezeigten Begriffen bezüglich der maximal Möglichen darstellt.

4.2. Beschränkung auf StepDefinitions

Um die angezeigten Informationen im Autovervollständigungsdialog noch stärker zu kondensieren, wurde eine weitere Einschränkung in die Suche eingebaut. Hierbei handelt es sich um die Möglichkeit, dass ausschließlich StepDefinitions angezeigt werden und alle zugehörigen Steps wegfallen. Abbildung 4.1 veranschaulicht den Zustand ohne aktivierte StepDefinition-Suche. Bei den in eckigen Klammern dargestellten Einträgen im Dialog handelt es sich um StepDefinitions. Die darunter eingerückten Einträge stellen jeweils einen Step dar, der vom regulären Ausdruck der StepDefinition erfasst wird. Man kann sehr gut erkennen, dass die StepDefinitions bei Agilent bereits sehr oft wiederverwendet werden, was ein sehr gutes Zeichen ist. Um eine schnelle Übersicht über mögliche passende Einträge aufgrund des bereits eingegebenen Satzes "And I see" zu erhalten, ist jedoch deshalb schwer. Es sind zu viele Einträge vorhanden, die keinen direkten Mehrwert bei der Suche nach passenden Steps bieten und wichtige Plätze in den 25 möglichen einnehmen.

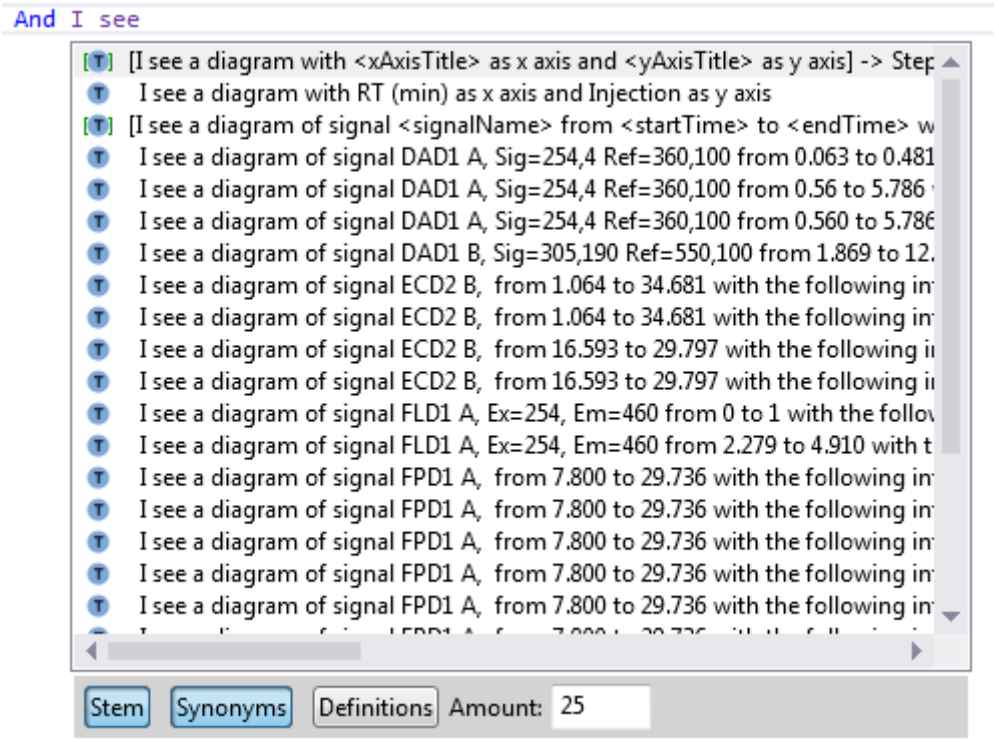

Abbildung 4.1.: Plugin mit deaktivierter StepDefinition-Suche

Die StepDefinition-Suche wurde implementiert um dieses Problem zu lösen. Hierbei werden ausschließlich StepDefinitions und somit mehr sinnvolle Einträge innerhalb der gegebenen Vorschläge angezeigt. Abbildung 4.2 veranschaulicht dies. Wie zuvor wurde die gleiche Datenbasis verwendet und die Eingabe enthält zu diesem Zeitpunkt "And I see".

Es ist sofort zu erkennen, dass es nur 8 StepDefinitions gibt, die auf die bisherige Eingabe passen. Ein Durchblättern der vielen Einträge zuvor entfällt mit dieser Funktion.

4.3. Synonyme

Die Synonymsuche sollte semantisch gleiche Steps erkennen, auch wenn sie syntaktisch unterschiedlich sind. Die zuvor nicht erkannte Problematik dabei ist aber, dass die Zuordnung eines Synonyms kontextabhängig ist. Je nach Kontext kann das Wort als Nomen oder Verb

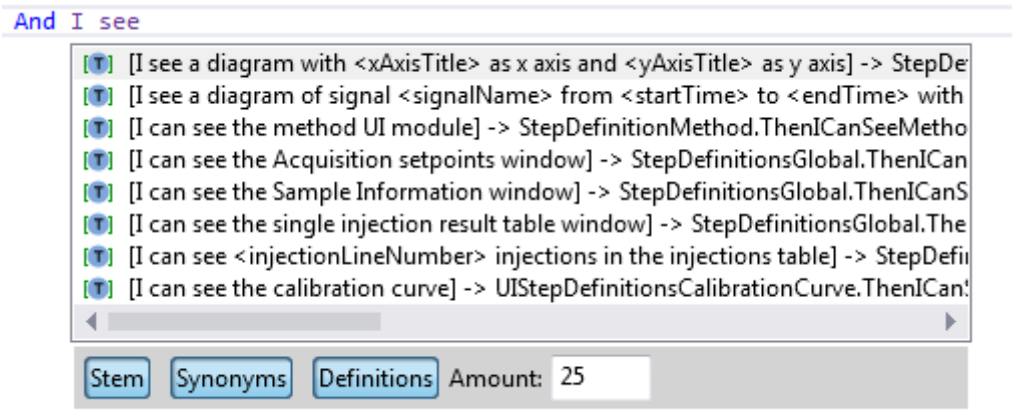

Abbildung 4.2.: Plugin mit aktivierter StepDefinition-Suche

im Test gemeint sein. Damit nicht alle Synonyme für sowohl Verben als auch Normen des Wortes gesucht werden, muss zunächst herausgefunden werden um welche Wortart es sich handelt. Dies ist aber nur automatisch über entsprechende Algorithmen möglich, wenn der komplette Satz gegeben ist. Ohne die komplette Satzstruktur zu kennen, kann keine automatische Aussage darüber gemacht werden, ob das Wort ein Nomen oder ein Verb ist. Den kompletten Satz einzutippen, wenn nach einem eventuell vorhandenen Step gesucht wird, ist aber kontraproduktiv. Zum einen ist es bei einer Autovervollständigung nicht intuitiv alle Wörter zu schreiben, sondern nur den wichtigsten Teil, um zu sehen, ob entsprechende Treffer gefunden werden. Zum anderen ist die Voraussetzung, dass der Anwender den kompletten und fertigen Step hinschreibt absurd, wenn er Unterstützung von einem Werkzeug erwartet. Die Problematik, den Step nicht weiter zu verändern, sondern einfach das bisher schon geschriebene weiter zu verwenden kommt dabei ebenfalls zum tragen.

Aufgrund des neu formulierten Prozesses zum Erstellen eines Testskripts ist diese Möglichkeit nicht gegeben. Stattdessen wurde lediglich eine auf einem Glossar eingeschränkte Synonymsuche implementiert. Diese sucht nach der Eingabe von bestimmten Wörtern die im Glossar vorkommen danach, ob dieses Wort im Glossar ein Synonym hat. Bei einem entsprechenden Treffer wird die Suche auch auf das Synonym erweitert. Hier muss jedoch anders als eventuell zunächst angenommen darauf geachtet werden, dass das Wort komplett eingegeben wird. Eine Teilsuche kann nicht erfolgen, da das dann zu einer *-Suche ausarten würde und Ergebnisse liefern würde, die teilweise unerklärlich für den Anwender sind. Der damit verbundene Schaden wiegt nicht die gewonnenen Vorteile auf und wurde deshalb in Absprache mit Agilent weggelassen. Ebenso wie die Einschränkung der Einträge wurde die Synonymsuche in den Dialog des Plugins integriert, um die Funktion beliebig an- oder auszuschalten.

Die Abbildungen 4.3(a) und 4.3(b) veranschaulichen die Funktionsweise der Synonymsuche anhand eines einfachen Beispiels. Im Glossar befindet sich zu diesem Zeitpunkt ein Eintrag mit „select” und „click” als Synonyme.

4.4. Stammsuche

Die Stammsuche konnte wie ursprünglich geplant umgesetzt werden. Dabei werden offene Wörterbücher (OpenOffice.org) verwendet um die eingegebenen Wörter auf ihren Stamm hin zu untersuchen und dann mit den Einträgen in der Liste zu vergleichen, die ebenfalls alle als Stammwörter vorliegen. Die Darstellung der Steps in der Autovervollständigung bleibt davon unberührt, um weiterhin sinnvoll konjugierte Sätze angezeigt zu bekommen.

(a) Synonymsuche deaktiviert.

(b) Synonymsuche aktiviert.

Abbildung 4.3.: Gegenüberstellung der Suchergebnisse ohne Synonymsuche und mit

Damit können bei der Erstellung weitestgehend auf Singular oder Pluralüberlegungen verzichtet werden. Ebenso fällt der Einfluss der Konjugation der Verben weg. Wie auch in den beiden anderen konkreten Implementierungsdetails wurde hier auf eine Möglichkeit im Dialog des Plugins geachtet, bei der die Option explizit an- oder ausgeschaltet werden kann.

4.5. Tageinschränkung

Die Möglichkeit, die Steps weiter einzuschränken, aufgrund der im Szenario definierten Tags wurde, ebenfalls implementiert. Allerdings wurde diese Option nach der ersten Interviewrunde verworfen, da die Teams zu unterschiedlich in der Herangehensweise sind. Die ursprüngliche Planung sah vor, dass die Tags für die unterschiedlichen Szenarios verwendet werden, um die Suche weiter einzuschränken. Die Option verringerte die möglichen angezeigten Steps erheblich. Das Problem dabei ist, dass die Tags nicht für diese Art der Testschrittformulierung gedacht sind. Tags stellen lediglich eine Gruppierung der Testschritte bereit, um zum Beispiel schnelle Tests von langsamen zu unterscheiden. Sie sollten nicht für die Limitierung der Schritte genutzt werden die nur in einem bestimmten Kontext ausführbar sind. Das führt zu deutlich schlechter formulierten Schritten und entsprechenden Implementierungen.

Da die Teams sich in dieser Hinsicht grundlegend unterscheiden und die Tageinschränkung von einem theoretischen Standpunkt aus nicht ratsam ist, wurde dieses Feature entfernt. Der neu entwickelte Prozess konnte nur so einheitlich für beide Teams gleichermaßen entwickelt werden. Die Vorteile der Tageinschränkung konnten die Nachteile der unterschiedlichen Behandlung der Teams nicht aufwiegen.

4.6. Weitere Änderungen

Zusätzlich dazu wurden weitere kleinere Verbesserungen am Code vorgenommen die hier nicht im Detail beschrieben werden. Dabei handelt es sich hauptsächlich um kleine Performanzoptimierungen und verbesserte Bedienbarkeit. Erwähnenswert hierbei ist nur der Wegfall der Untersuchung auf bestimmte Reihenfolgen von Steps die dadurch eine weitere Einschränkung ermöglichen. Durch die angestrebte Veränderung im Erstellungsprozess im Zusammenhang mit den anderen Änderungen ist es möglich, die Steps unabhängig von den davor folgenden zu finden. Dadurch ist diese Funktionalität nicht benötigt und wurde deshalb nicht weiter implementiert.

Um eine Übersicht für einen Vergleich zu bekommen wird in Abbildung 4.4 der Originalzustand des Plugins gezeigt und in 4.5 der entsprechend modifizierte Zustand.

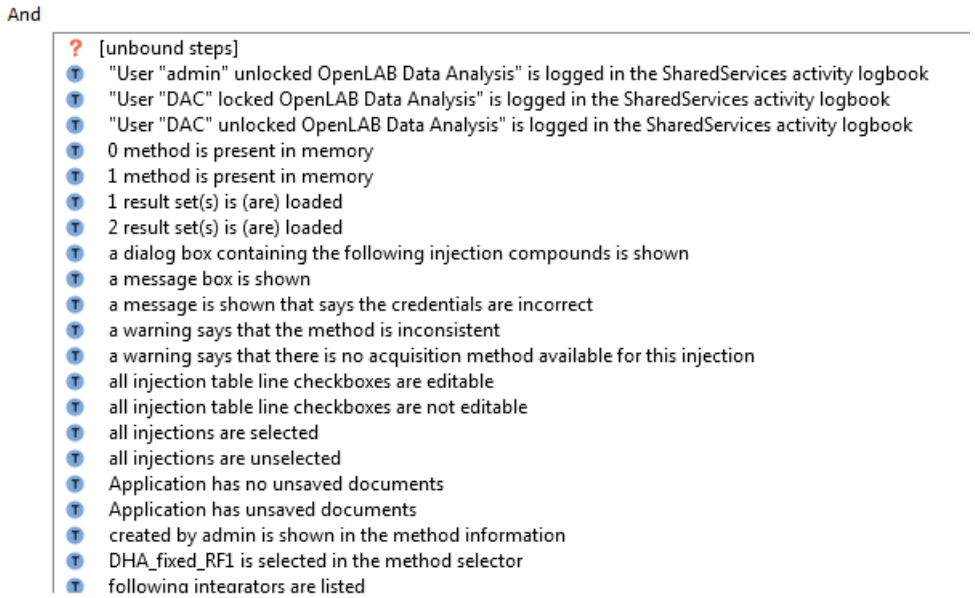

Abbildung 4.4.: Plugin im Originalzustand

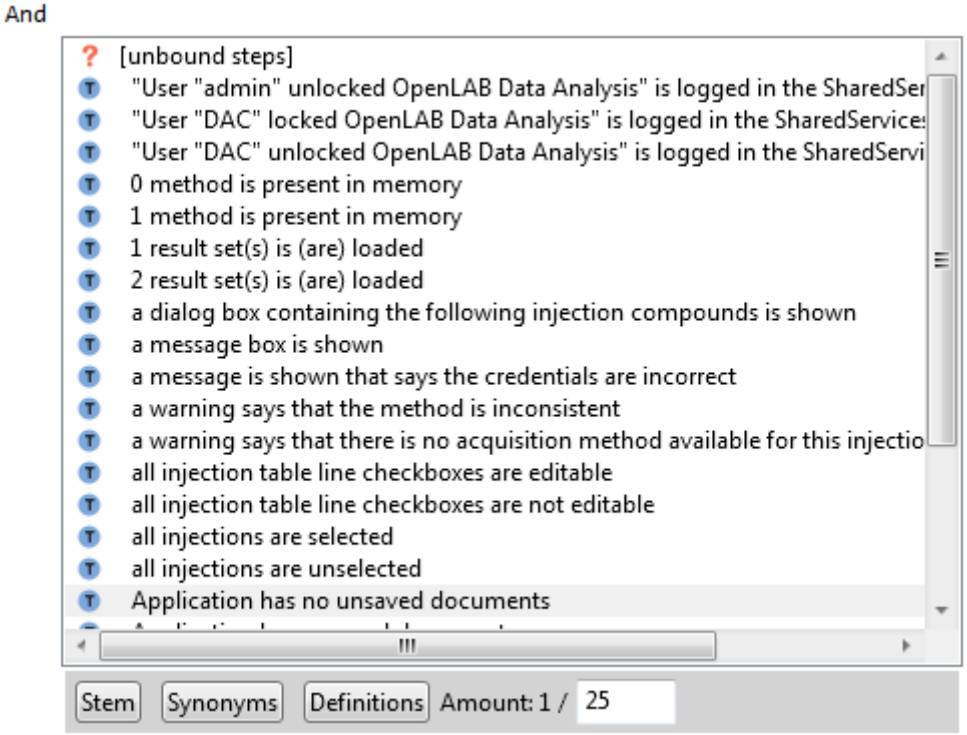

Abbildung 4.5.: Plugin nach den Änderungen

5. Auswertung

Trotz der Ausfälle bei den Mitgliedern der Fallstudie und der geringen Anzahl an neu geschriebenen Testskripten innerhalb der zweiten Phase, können Ergebnisse aus der Studie gezogen werden.

Es wird zunächst auf die Auswertung der qualitativen Daten aus den Interviews eingegangen. Danach folgt die quantitative Auswertung aus der Analyse des Versionsverwaltungssystems. Abschließend wird zusammenfassend beantwortet, ob die Ziele erreicht wurden und die Fragen aus Kapitel 1 geklärt werden konnten.

5.1. Interviews

Die Interviews zu Beginn der Fallstudie haben eindeutig gezeigt, dass die bisherige Werkzeugunterstützung bei Agilent keinen guten Anklang bei den Mitarbeitern hatte. Nur ein Drittel der Chemiker nutze die Autovervollständigung für die anfallende Arbeit. Bei den Entwicklern verwendete nur die Hälfte die angebotene Hilfe für das Aufsuchen der vorhandenen Testschritte. Die Entwickler setzten ausschließlich oder zumindest hauptsächlich die Volltextsuche von Visual Studio ein.

Dieser Umstand konnte mit Hilfe der Anpassungen an SpecFlow fast komplett beseitigt werden. Alle Chemiker verwenden nun die Autovervollständigung regelmäßig für ihre Arbeit und die Codesuche wurde nicht mehr von ihnen eingesetzt. Auch diejenigen Entwickler, welche das Werkzeug eingesetzt haben, berichteten von einer positiven Erfahrung und einfacher aufzufindenden Steps. Das Vorgehen wurde zumeist als „zielgerichteter", „einfacher" und „schneller" beschrieben. Damit konnte die erste Hypothese aus Kapitel 3, dass die Erstellung der Tests mit einer entsprechend angepassten Werkzeugunterstützung schneller und effizienter von statten geht, qualitativ bestätigt werden. Die Anzahl der Anwender ist zwar leider geringer als erhofft, jedoch ist die Rückmeldung dabei eindeutig und wird somit als Bestätigung für die erste Hypothese verwendet.

Die Probleme, die anfangs bezüglich des Plugins gemeldet wurden, konnten fast komplett beseitigt werden. Die GUI reagiert schnell auf Eingaben und filtert die Ergebnisse in einer angenehmen Geschwindigkeit, ohne die Oberfläche einzufrieren. Die Performanz wurde von den Befragten durchweg als „gut" oder „verbessert" beschrieben.

Die Einschränkung der Suchergebnisse auf maximal 25 im Standardfall wurde ebenfalls von allen als positiv beschrieben. Bei den Aussagen darüber, ob ein anderer Standardwert

besser gewesen wäre, sind beide Richtungen vertreten. Ein Teil der Mitarbeiter hätte lieber noch weniger Ergebnisse, um noch schneller eine Übersicht zu bekommen. Einem anderen Mitarbeiter waren es nicht genug Ergebnisse auf einmal. Der Wert ist sicherlich stark subjektiv geprägt und die Möglichkeit einer individuellen Anpassung des Wertes ist in jedem Fall sinnvoll.

Probleme mit dem geänderten Plugin waren kaum vorhanden und beschränkten sich auf Installationsproblem, eine als „unpräzise" empfundene Benennung der Knöpfe im Dialogfenster und einer notwendigen Anpassung der Arbeitsweise aufgrund der Umstellung. Es war zu erwarten, dass der kleine zur Verfügung stehende Zeitraum dazu führt, dass die Umstellung einen nicht unerheblichen Teil an Problematiken verursacht. Allerdings wurde dies nur von einer Person als störend berichtet. Die Installationsproblematik konnte innerhalb kürzester Zeit gelöst werden und die Benennung der Knöpfe wurde lediglich von einer Person als störend empfunden. Es ist sicherlich möglich, eine präzisere Formulierung zu verwenden, jedoch ist dies für den Einsatz und die darauf aufbauende Auswertung nicht von Bedeutung.

Die Schätzungen zur benötigten Arbeitszeit zum Implementieren der Tests haben sich nicht aussagekräftig verändert. Die Werte schwanken dabei von wenigen Stunden bis zu maximal zwei Tagen. Dies ist sehr stark davon abhängig, welche Userstory gerade implementiert wird. Das war zu erwarten und dient hauptsächlich um zu belegen, dass die Erstellung von funktionalen Tests einen großen Teil der Arbeitszeit ausmacht und damit Potential für Verbesserungen hat.

Die Schätzungen bezüglich der notwendigen Änderungen nach dem initialen Rumpfskript sind nicht zurückgegangen, was ebenfalls zu erwarten war, da es schon in den ersten Interviews als schwierig empfunden wurde, hierzu eine eindeutige Antwort zu geben. Sehr gut sichtbar ist dies anhand der Rückmeldung eines Chemikers, welcher von einer Änderungsrate zwischen „10% und 100% berichtet hat. Die Testskripte sind zu unterschiedlich was Aufwand, Schwierigkeit und vorhandene Zeit betrifft. Gute Schätzungen über alle Tests abzugeben ist deshalb nur sehr schwer machbar. Um hier eindeutige Aussagen treffen zu können, muss eine quantitative Analyse gemacht werden.

Der geänderte Prozess wurde allgemein sehr gut zusammen mit den Änderungen angenommen. Es ist nicht möglich zu unterscheiden, ob die Änderungen am Werkzeug oder die Änderung am Prozess die positiven Effekte ausgelöst haben, von denen die Mitarbeiter berichten. SpecFlow stand bereits vorher zur Verfügung, wurde jedoch nicht von allen genutzt. Dies kann mehrere Erklärungen haben. Die einfachste wäre, dass die genannten Probleme eine Nutzung erschwerten und damit der theoretische Vorteil der Autovervollständigung nicht in eine Zeitersparnis für die Mitarbeiter umgewandelt werden konnte. Hierfür spricht die als schlecht bewertete Performanz am deutlichsten. Ebenso kann hierfür die Erfahrung derjenigen herangezogen werden, die bereits zuvor SpecFlow regelmäßig eingesetzt haben. Diese Mitarbeiter haben ebenfalls eine Verbesserung bemerkt, die ausschließlich auf die Anpassungen im Werkzeug zurückzuführen sind.

Eine andere Erklärung wäre, dass die Mitarbeiter sich nicht genügend in das SpecFlow-Plugin eingearbeitet hatten. Hierfür spricht der Umstand, dass versucht wurde, komplette Sätze als Suchkriterium einzugeben. Da die Autovervollständigung von SpecFlow auch im Ursprungszustand darauf ausgelegt war, dass lediglich bestimmte Schlüsselwörter eingegeben werden, waren die Suchergebnisse, wie zu erwarten, schlecht. Das Ausweichen auf die Volltextsuche, in welcher dann mit Schlüsselwörtern gesucht wurde, macht diese Erklärung wahrscheinlich.

In diesem Fall sind die Anpassungen an das Werkzeug im Vergleich eher unbedeutend und die berichteten Verbesserungen wurden hauptsächlich durch den geänderten Prozess

verursacht. Das die Synonymsuche, als auch die Stammsuche kaum verwendet wurde, bekräftigt diesen Erklärungsansatz. Obwohl diese Verbesserungen in Gesprächen mit Agilent gewünscht wurden, haben sie keinen Effekt für die tägliche Arbeit gehabt.

Es ist die Meinung des Autors, dass die Änderungen am Werkzeug trotzdem von Nöten waren, damit die Verbesserungen auftraten. Der Hauptpunkt ist dabei die verbesserte Performanz, die durch die Einschränkung der Ergebnisse erreicht wurde, als auch die standardmäßige ausschließliche Anzeige der StepDefinitions. Diese Änderungen ermöglichen es dem Ersteller der Tests schneller einen Überblick zu gewinnen und die Suche besser auf die relevanten Steps einzuschränken. Zusammen mit dieser Vereinfachung war die Annahme einer neuen Herangehensweise für die Mitarbeiter deutlich einfacher. Nur mit den Verbesserungen wurde der theoretische Vorteil bei der Verwendung der Autovervollständigung auch direkt für die Mitarbeiter ersichtlich, ein praktischer Vorteil.

Die Antworten des Expertenteams bestätigen damit die Annahme, dass die Anwender individuell angepasste Software häufiger einsetzen und die Funktionsvorteile besser nutzen können. Damit kann aus den bisherigen qualitativen Daten auch die zweite Hypothese belegt werden.

5.2. Versionsverwaltungssystem Auswertung

Für die quantitative Auswertung der Tests wurden zunächst etliche Daten erhoben, wie in Kapitel 3.7 beschrieben. Hierzu zählen unter anderem die verantwortliche Person, die notwendigen Änderungen und der initiale Zustand des Testskripts. Um eine aussagekräftige Analyse zu ermöglichen, muss zuerst entschieden werden, welche Tests für die Auswertung verwendet werden. So ist es möglich alle Tests seit der Einführung von Akzeptanztests zu betrachten oder nur einen neueren Teil. Um Lerneffekte, welche innerhalb der Zeit vor der Modifikation des Werkzeugs stattfanden, besser beurteilen zu können, wurden sowohl alle Tests betrachtet, als auch nur die neueren aus dem letzten Jahr. Die Test aus dem Jahr 2013 stellen eine bessere Vergleichsbasis dar, da weiterhin genügend Tests (70) vorhanden sind, jedoch die Anfangsproblematiken mit der neuen Technik innerhalb der eingesetzten Jahre verschwunden sein sollten.

Insgesamt wurden 190 funktionale Tests vor der Einführung der Änderungen an SpecFlow analysiert. Dabei wurden 17 unterschiedliche Personen identifiziert, welche Feature-Dateien initial eingecheckt haben. Ein Teil der Mitarbeiter war schon zur Beginn der Studie nicht mehr in der ursprünglichen Position tätig und wurde somit nicht Teil der Untersuchung. Deshalb wurden lediglich 148 Tests untersucht, die von den Teilnehmern der Studie erstellt wurden.

Nach der Einführung des modifizierten Werkzeugs wurden insgesamt 13 weitere funktionale Tests geschrieben. Die Anzahl der Ersteller lag bei sieben Personen. Davon waren drei der Mitarbeiter nicht Teil der Studie und eine Person hatte in der Zeit das Werkzeug nicht für die Erstellung genutzt. Deshalb können lediglich fünf der Tests für die Auswertung verwendet werden.

5.2.1. Prozentuale Änderungen der untersuchten Tests

Die wohl wichtigste Kennzahl ist die durchschnittlich benötigte Änderungsrate an den Tests. Tabelle 5.1 zeigt die wichtigsten Werte für die unterschiedlichen Zeiträume an. Die verwendeten Primärdaten können jeweils aus Anhang A entnommen werden.

Tabelle 5.1. Durchschnittliche Änderungen an Tests

	Alle bis zur Einführung	Nur 2013	Nach der Einführung
Änderungen	27,0%	22,4%	13,7%

Tabelle 5.2. Primärdaten für abhängigen t-Test

Person	Bevor (X_i)	Danach (Y_i)	$D_i = X_i - Y_i$	$(D_i - \overline{X}_D)^2$
1	0,442	0,307	0,134	0,009
2	0	0,0454	-0,045	0,076
3	0,663	0,0579	0,605	0,139

Der Tabelle kann entnommen werden, dass es bereits ohne die Einführung des Werkzeugs Verbesserungen gegeben hat. Die Erfahrungen im Umgang mit der neuen Technik und die erweiterte Basis an bereits vorhandenen Tests hat zu einer Verbesserung beigetragen. Nach der Einführung konnten die durchschnittlich notwendigen Änderungen erheblich gesenkt werden.

Der Sprung von fast 10% muss aus verschiedenen Blickwinkeln betrachtet werden. Zum einen sind die bereits vorhandenen Steps ein nicht zu vernachlässigender Faktor, wenn die Veränderung betrachtet wird. Dies könnte Teil der Erklärung sein, warum die Tests weniger Veränderungen benötigten als in den Jahren davor. Diese fließen jedoch auch zu einem großen Teil in die 2013 geschriebenen Tests ein und eine so starke Veränderung ist damit nicht zu erklären. Eine andere Möglichkeit ist, dass die Tests in einem Gebiet geschrieben wurden, indem eher einfachere Bedingungen überprüft werden mussten. Es ist deshalb möglich, dass die Verbesserung rein dadurch erklärt werden kann, dass die Tests besonders einfach für den jeweiligen Ersteller erschienen. Ein Ausschluss der beiden Überlegungen ist aufgrund der Datenmenge nicht möglich und muss bei der Bewertung beachtet werden.

Unter der Annahme, dass innerhalb der letzten beiden Sprints nicht ausschließlich einfache Tests geschrieben wurden und der Annahme, dass die kleine Datenmenge repräsentativ für die weiteren Tests ist, kann geschlossen werden, dass es einen positiven Effekt durch die Veränderungen am SpecFlow-Plugin gegeben hat.

5.2.2. Gepaarter Student's t-Test

Um zu überprüfen, ob die Veränderungen statistisch belegt werden können, wurde ein Student's t-Test verwendet. Der Student's t-Test ermöglicht es, zu entscheiden, ob eine entsprechende Nullhypothese verworfen werden kann. Konkret in diesem Fall bedeutet dies, dass es keine Verminderung in der durchschnittlich benötigten Änderungsrate für die Akzeptanztests gibt. Beim t-Test wird unterschieden, ob eine oder zwei Stichproben verwendet werden. Bei einem t-Test mit zwei Stichproben muss untersucht werden, ob die Proben abhängig oder unabhängig sind. Im Falle der Studie sind die Stichproben voneinander abhängig, da zu unterschiedlichen Zeitpunkten das Ergebnis der jeweiligen Personen untersucht wird. Die sogenannte „Behandlung", welche die Probanden erhielten, ist dabei das modifizierte Werkzeug und die damit verbundene Schulung. Ob die gemessene Veränderung statistisch signifikant ist, kann demnach mit einem abhängigen t-Test mit zwei Stichproben untersucht werden[TH08][S. 146]. Die verwendete Formel ist

$$t = \frac{\overline{X}_D - \mu_0}{\sqrt{S_D^2/n}} \tag{5.1}$$

$\overline{X}_D$ ist dabei der arithmetische Mittelwert der Differenz aus den beiden Stichproben. Mit den Primärdaten aus Tabelle 5.2 kann dies einfach ermittelt werden über

$$\overline{X}_D = \frac{1}{n} * \sum_{i=1}^{n} D_i = \frac{0,134 - 0,045 + 0,605}{3} = 0,231 \tag{5.2}$$

Die Daten aus der Tabelle werden dabei von den Primärdaten aus Anhang A.1 und A.2 gespeist. Die jeweils ersten drei Blöcke in den Rohdaten wurden für die Werte in der Tabelle verwendet und stellen bei X_i und Y_i jeweils den im Mittel notwendigen Anteil an Änderungen für Tests dieser Person dar.

μ_0 ist in der t-Formel die prognostizierte Abweichung vom Mittelwert. Im Normalfall, wenn untersucht wird, ob die Nullhypothese verworfen werden kann, ist $\mu_0 = 0$. S_D ist die Standardabweichung und n beschreibt die Größe der verbundenen Stichprobe. Die Standardabweichung ist schnell errechnet.

$$S_D^2 = \frac{\sum (D_i - \overline{X}_D)^2}{n-1} = \frac{0,226}{2} = 0,113 \tag{5.3}$$

Damit kann nun t durch 5.1 errechnet werden.

$$t = 1,193 \tag{5.4}$$

Der Freiheitsgrad „df" ist in dieser Art von t-Test immer n-1 und der t-Wert kann damit aus verfügbaren t-Tabellen mit den unterschiedlichen Werten verglichen werden. Da die beobachteten Veränderungen nur für eine Richtung interessant sind, kann mit den einseitigen Werten verglichen werden. Ist der Eintrag in der Tabelle für ein akzeptiertes Signifikanzniveau kleiner als der errechnete t-Wert, kann die Nullhypothese verworfen werden.

Es sollte beachtet werden, dass bei allen hier angegebenen Rechnungen nicht alle Stellen nach dem Komma angegeben wurden, um Platz zu sparen. Um exaktere Werte zu erhalten wurde die Berechnung zusätzlich mit Hilfe eines Statistikprogramms durchgeführt. Dies ermöglicht es den p-Wert exakt zu bestimmen, mit dessen Hilfe bestimmt werden kann, welches Signifikanzniveau angenommen werden müsste, damit die Nullhypothese verworfen werden kann. Der errechnete p-Wert liegt bei 0,1775.

Dies bedeutet, dass ein Signifikanzniveau von über 17% angenommen werden müsste um die Nullhypothese verwerfen zu können. Anders formuliert bestünde eine 17%-Chance, dass die Nullhypothese fälschlicherweise ausgeschlagen wurde. Da es in wissenschaftlichen Arbeiten nicht üblich ist, über ein Signifikanzniveau von 0.05% ($\alpha = 0.05$) hinauszugehen, ist es zu diesem Zeitpunkt leider nicht möglich eine statistische signifikante Aussage über die Veränderungen zu treffen.

Da der Datensatz nach der Einführung des Werkzeugs noch sehr klein ist, kann es durchaus möglich sein, dass ein erneuter Test mit weiteren Daten ein anderes Ergebnis liefert.

5.2.3. Alternative Bewertung

Es ist möglich die Daten alternativ zu betrachten und zu bewerten, wenn bestimmte Annahmen gemacht werden. Es wird zunächst die Annahme getroffen, dass alle Tests bei denen keine Änderungen vorgekommen sind einen Ausreißer darstellen und dies ebenfalls für Tests mit einer Änderung von über 100% gilt. Damit reduziert sich die Datenbasis vor der Einführung des Werkzeugs und die Paarung der Daten auf der 0 verschwindet. Wird nun zusätzlich angenommen, dass die untersuchten Tests lediglich einer zufällige Ziehung entnommen wurden, kann davon ausgegangen werden, dass die Tests von 2013 und die Tests in 2014 voneinander unabhängig sind.

Unter diesen Annahmen kann ein weiterer t-Test durchgeführt werden um die Nullhypothese eventuell zu widerlegen. Dabei handelt es sich um einen t-Test mit zwei unabhängigen Stichproben und angenommener ungleicher Varianz, da diese nicht vor dem Test als gleich

angenommen werden kann. Der hier beschriebene Test ist auch bekannt unter dem Namen „Welch's-Test". Die dazugehörige Formel für t ist in 5.5 gegeben. [TH08, S. 145]

$$t = \frac{\mid \overline{X} - \overline{Y} \mid}{\sqrt{\frac{S_X^2}{n_1} + \frac{S_Y^2}{n_2}}} \tag{5.5}$$

Die zugehörigen Freiheitsgrade werden über die Formel in 5.6 berechnet und anschließend ganzzahlig gerundet.

$$df = (\frac{s_x^2}{n_1} + \frac{s_y^2}{n_2})^2 / (\frac{s_x^2/n_1)^2}{n_1 - 1} + \frac{(s_y^2/n_2)^2}{n_2 - 1}) \tag{5.6}$$

Auch dieser Test wurde mit Hilfe eines Statistikprogramms durchgeführt um genaue Ergebnisse zu bekommen.

Das Ergebnis ist ein t-Wert von 1,38 mit einem Freiheitsgrad df von 5,94. Leider kann auch mit diesen stark gefilterten Daten und den getroffenen Annahmen keine signifikante Änderung zwischen den Tests davor und den Tests danach gefunden werden. Der p-Wert liegt bei 0,108 und es bliebe damit eine Wahrscheinlichkeit von über 10%, dass die Nullhypothese fälschlicherweise verworfen wurde.

Es bleibt damit nichts anderes übrig, als weitere Daten zu sammeln und diese zu einem späteren Zeitpunkt erneut zu analysieren. Es ist gut möglich, dass dies zu einem anderen Ergebnis führt.

5.3. Zusammenfassende Auswertung

Die quantitative Auswertung konnte leider keine eindeutige Aussage darüber machen, ob die Verbesserungen nach der Einführung rein zufällig waren oder nicht. Trotzdem kann ein Trend innerhalb der prozentual benötigten Änderungen in den Tests nach der Einführung im Vergleich zu davor gesehen werden. Die Veränderung ist dabei so stark, dass es unwahrscheinlich ist, das dies lediglich durch Lerneffekte erklärt werden kann.

Dies korreliert mit den qualitativen Aussagen, die alle sehr positiv waren. Die Teilnehmer konnten, abgesehen von einer erwarteten Umstellung in der Arbeitsweise, keine negativen Aspekte an den Veränderungen an SpecFlow feststellen. Bei den positiven Aspekten dagegen wurden von einer verbesserte Fokussierung, einfacher und schneller Suche und allgemein verbesserter Reaktionsfreudigkeit beim Umgang berichtet.

Betrachtet man die quantitative und qualitative Auswertung gemeinsam, können die Eingangs gestellten Fragen beantwortet werden.

Wie wirken sich kleine Verbesserungen eines Vorschlagsystems zur halbautomatisierten Testerzeugung auf das Entwicklungsteam aus?

Vor allem die qualitative Auswertung lässt den Schluss zu, dass kleine Verbesserungen vom Entwicklungsteam besser angenommen werden können. Der konstante Druck der täglichen Aufgaben lastet auf allen Beteiligten eines Entwicklungsteams und gehört zum Alltag eines jeden erfolgreichen Unternehmens. Große Umstellungen im Prozess, bei denen nicht sofort ersichtlich ist, welcher Mehrwert für den einzelnen Mitarbeiter dabei herauskommt, sind schwerer anzunehmen als kleine. Wenn die kleinen Veränderungen zusätzlich noch speziell angepasst auf die Bedürfnisse und Wünsche der Mitarbeiter zugeschnitten werden, fällt die Akzeptanz leicht.

Auf der anderen Seite konnte jedoch auch gesehen werden, dass auch kleine Veränderungen nicht angenommen werden, wenn aus der Sicht der Mitarbeiter kein direkt ersichtlicher

Vorteil gegeben ist. SpecFlow wurde hauptsächlich für die Chemiker im Team verbessert und hatte damit nur indirekte Auswirkungen auf die Softwareentwickler. Aus diesem Grund wurde das Werkzeug nur von denjenigen eingesetzt, die sich direkte Verbesserungen im Arbeitsalltag vorstellen konnten.

Ebenso wurden Teile der Verbesserungen nicht sofort verwendet, da sie mit einem deutlichen Mehraufwand verbunden sind und keine Verantwortlichen dafür festgelegt wurden. Das die Synonyme nicht eingepflegt wurden, verhindert den effektiven Einsatz der Funktion und schmälert den theoretisch vorhandenen Vorteil. Das ein Teil der Mitarbeiter explizit den Wunsch geäußert hat, dass dies in Zukunft verwendet wird, lässt nur darauf schließen, dass der Arbeitsaufwand zur initialen Erstellung der Synonymliste zu hoch war. Dies ist im Nachhinein durchaus nachvollziehbar, da es etliche Veränderungen innerhalb der Teams gab, die nicht Teil des Arbeitsalltags waren und somit zusätzliche Ressourcen und Zeit beansprucht haben. Gleiches gilt für die Unterstützung der Stammsuche. Der Mehraufwand einer Umstellung bei der Eingabe rechtfertigt aus Sicht der Mitarbeiter offensichtlich nicht den versprochenen Nutzen.

Insgesamt kann damit festgehalten werden, dass kleine Verbesserungen des Vorschlagsystems nur dann Auswirkungen auf das Entwicklungsteam haben, wenn die Mitglieder einen direkten Nutzen für ihren Arbeitsalltag im Vorhinein erkennen können. Alle anderen Veränderungen, auch wenn es sich nur um kleine Modifikationen handelt, werden nicht sofort angenommen und benötigen entweder eine Überarbeitung, mehr Zeit oder müssen von außen erzwungen werden. Letzteres ist nur dann gerechtfertigt, wenn bewiesen werden kann, dass die Änderungen langfristige Verbesserungen mit sich bringen, was hier jedoch nicht der Fall war. Die Aussagen aus den Interviews lassen darauf schließen, dass mehr Zeit benötigt wird, damit die Mitarbeiter alle Verbesserungen nutzen und bewerten können.

Die zweite Hypothese, dass Mitarbeiter individuell angepasste Werkzeuge besser annehmen, kann aus den vorhandenen Daten trotzdem bestätigt werden. Das geänderte Plugin wurde zwar nicht komplett bei allein Beteiligten eingesetzt, jedoch sind diejenigen die das Werkzeug verwendet haben, vom Nutzen überzeugt und verwenden es alle über den Studienzeitraum hinaus.

Wie viel Nachbearbeitung der Tests kann eingespart werden, wenn das System an die Bedürfnisse vor Ort zugeschnitten wird?

Die quantitativen Daten lassen leider keine Aussage darüber zu, wie viel Nachbearbeitung der Tests eingespart werden kann. Daher kann die dritte Hypothese, dass eine bessere Werkzeugunterstützung zu einer höheren Wiederverwendung der einzelnen Testschritte und damit verbundenen Vorteile für die Änderungsrate und Fehlerrate mit sich bringt, nicht bestätigt werden.

Es müssen mehr Testskripte im Laufe der nächsten Wochen und Monate geschrieben werden um erkennen zu können, ob die Einsparungen statistisch signifikant sind. Aufgrund der bisherigen Daten kann lediglich vermutet werden, dass es einen Trend nach der Einführung durch das veränderte SpecFlow-Plugin gegeben hat. Die ersten Zahlen sprechen hier für eine Verbesserung.

Wie gut werden die Anwender durch die Verbesserungen in der Testerstellung unterstützt?

Die qualitative Auswertung hat eindeutig gezeigt, dass die Anwender durch die Verbesserungen an SpecFlow unterstützt werden. Die Aussage, dass die Modifikation ein zielgerichteteres Vorgehen ermöglicht und allgemein ein einfacheres und schnelleres Auffinden der vorhandenen Steps ermöglicht, kann als großer Erfolg angesehen werden. Die Erstellung

der Tests nimmt einen nicht unerheblichen Teil der Arbeitszeit ein und jede Effizienzsteigerung in diesem Bereich ist damit aus Sicht eines Arbeitgebers wünschenswert und eindeutig positiv. Zusätzlich dazu kann festgehalten werden, dass alle Mitarbeiter, die das Werkzeug eingesetzt haben, auch explizit bestätigt haben, dass sie das Werkzeug weiterhin nutzen werden. Dies lässt darauf schließen, dass die Verbesserungen nicht auf Lasten der Mitarbeiter ausgetragen werden, sondern auch diese gerne damit arbeiten.

Auf der anderen Seite muss jedoch auch betrachtet werden, dass dies nur für einen Teil der Mitarbeiter der Fall war. Die Softwareentwickler haben von keiner nennenswerten Veränderungen durch den Einsatz des Werkzeugs berichtet. Aus der Sicht eines Entwicklers, welcher ausschließlich die Implementierung der Akzeptanztests vornimmt, konnte keinerlei Verbesserung festgestellt werden. Aus theoretischer Sicht, wäre durch den Zusammenhang zwischen Chemiker und Entwickler eine Verbesserung wünschenswert gewesen. In der Praxis konnte dies jedoch nicht erkannt werden. Es besteht auch hier die Möglichkeit, dass dies auf den mangelnden Daten beruht. Eine indirekte Verbesserung für die Aufgaben des Entwicklers ist nicht komplett auszuschließen, sobald mehr Tests geschrieben wurden.

Da die erste Hypothese sich darauf stützt, dass die Steps einfacher oder schneller gefunden werden können, wenn eine entsprechende Werkzeugunterstützung vorhanden ist, kann die Hypothese bestätigt werden. Die Modifikationen wurden zwar nicht von allen Teilnehmern als hilfreich eingestuft, jedoch von denjenigen, die Steps aktiv suchen, wurde es als nützlich bewertet. Zu Beginn der Studie konnte nicht abgesehen werden, dass ein Teil der Entwickler nicht nach vorhandenen Steps in ihrer Arbeit sucht, weshalb diese nicht zu den 75% aus den Interpretationskriterien gehören. Betrachtet man ausschließlich diejenigen, die schon zuvor aktiv nach Steps gesucht haben, so kann man aus den Antworten der Probanden folgern, dass 100% bestätigen, dass die Arbeit sich mit den Änderungen erleichtert hat. Deshalb wird die erste Hypothese als bestätigt angesehen.

Damit konnten alle eingangs gestellten Fragen beantwortet werden. Es wäre vorteilhaft gewesen, wenn mehr Daten zur Verfügung gestanden hätten um die Auswertung eindeutig abzuschließen. Für den Moment kann lediglich festgestellt werden, dass die Einführung eines speziell angepassten und verbesserten Werkzeugs zur halbautomatisierten Testerzeugung sichtbare Vorteile für die Mitarbeiter bei Agilent erbracht hat.

6. Zusammenfassung und Ausblick

Die Fallstudie wurde gestartet um zu erkunden, ob kleine angepasste Veränderungen an einem eingesetzten Werkzeug bereits Auswirkungen auf die halbautomatisierte Erzeugung von Akzeptanztests hat. Akzeptanztests bieten die Möglichkeit eine einfache Überprüfung der implementierten Funktionen innerhalb eines Systems durch einen technisch unversierten Anwender vorzunehmen, indem natürlichsprachliche Formulierungen für den Aufbau des Testes verwendet werden. Die damit verbundene Technik nennt sich Behaviour Driven Development.

Die Firma Agilent hatte bereits zum Beginn der Studie zwei Teams aus Experten auf dem Gebiet des Behaviour Driven Development und wollte den Vorteil weiter ausbauen. Die Verbesserungspotentiale bei Agilent bezogen sich zum einen auf technische Verbesserungen der eingesetzten Werkzeuge und zum anderen auf eine optimierte Herangehensweise bei der Nutzung der bereits verfügbaren Techniken. Wie schon von Agilent zu Beginn an vermutet, konnte innerhalb der Studie gezeigt werden, dass auch nach jahrelangem Einsatz der Techniken noch Verbesserungspotential in den Teams vorhanden ist.

Die eingesetzten Werkzeuge waren Microsoft Visual Studio und das darauf aufbauend Plugin SpecFlow. SpecFlow wurde nicht komplett von den Teams angenommen und es wurden nicht alle Vorteile der Software genutzt um den Arbeitsalltag der Mitarbeiter effizienter zu gestalten. Stattdessen wurde oft auf eine suboptimale Ausnutzung der Volltextsuche der Entwicklungsumgebung zurückgegriffen. Diesen Umstand zu verändern war Hauptbestandteil der praktischen Arbeit.

Die Problematiken mit umfassenden Veränderungen in einem Entwicklungsprozess sollten vermieden werden und stattdessen lag der Fokus auf gezielten Verbesserungen innerhalb des bereits vorhandenen Arbeitsalltages. Die Interviews zeigten auf, dass es einzelne Punkte innerhalb des SpecFlow-Plugins gab, die geändert werden können, damit eine Verbesserung des gesamten Erstellungsprozess erfolgt. Die notwendigen Änderungen wurden aus theoretischer Sicht untersucht und mit den praktischen Aussagen der Mitarbeiter verglichen.

Die wichtigsten Änderungen betrafen dabei eine optimierte Reaktionsfreudigkeit der Benutzeroberfläche und die Einschränkung der Suchergebnisse, um eine bessere Übersicht zu bekommen und die Daten kompakter darzustellen. Theoretisch erarbeitete Änderungen, wie die Unterstützung einer Synonym- und Stammsuche, wurden ebenfalls implementiert.

Nach der Implementierung und der Einführung des Werkzeugs, zusammen mit einer veränderten Herangehensweise, konnte in den anschließenden Interviews festgestellt werden,

dass die Modifikationen an SpecFlow durchaus eine Verbesserung mit sich gebracht haben. Die am besten bewerteten Verbesserungen kamen dabei aus den praktischen Anforderungen, welche ursprünglich hauptsächlich von den Mitarbeitern bei Agilent kamen. Die rein theoretisch getriebenen Veränderungen, wie die Stammsuche wurden nicht sehr gut angenommen und konnten damit nicht zu einem Mehrwert beitragen. Dies bestätigte die Annahme, dass kleine angepasste Veränderungen erheblich besser angenommen werden können, wenn die Mitarbeiter innerhalb weniger Minuten eine direkte Verbesserung für ihren Arbeitsalltag erkennen können.

Es ist nicht auszuschließen, dass auch größere Veränderungen einen positiven Einfluss auf die Erstellung der Tests gehabt hätten. Jedoch lassen die Nutzungsphase und die Interviews vor der Einführung den Schluss zu, dass es sich bei der ursprünglichen Einführung des SpecFlow-Plugins um eine zu starke Veränderung gehandelt hat und damit nicht alle theoretisch vorhandenen Vorteile ausgenutzt werden konnten.

In einer quantitativen Analyse der Tests aus dem Versionsverwaltungssystems konnte bisher leider keine signifikante Veränderung bestätigt werden. Hier wurden die Akzeptanztests auf die durchschnittlich benötigten Änderung nach dem initialen Eintrag in das Versionsverwaltungssystems hin untersucht. Die Nullhypothese ist dabei, dass sich keine Veränderung zwischen dem Zustand vor und nach der Einführung des geänderten SpecFlow Plugins bezüglich der notwendigen Änderungen an einem Testskript ergeben hat. Allerdings müsste bei einem gepaarten t-Test ein Signifikanzniveau von über $\alpha = 0,17$ angenommen werden, damit die Nullhypothese verworfen werden könnte. Eine irrtümliche Verwerfung wäre damit sehr wahrscheinlich. Die Menge der bisher vorhandenen Daten lässt durchaus den Schluss zu, dass weitere Daten eine andere Aussage bekräftigen könnten.

Die reine Betrachtung der Tests vor und nach der Einführung ohne statistisches Verfahren zeigt den Trend auf, dass die notwendigen Änderungen an den Tests durchaus zurückgegangen sind. Es wurden nach der Einführung durchschnittlich fast 10% weniger Änderungen an einem Test vorgenommen. Lerneffekte konnten zwar nicht vollständig ausgeschlossen werden, jedoch wird davon ausgegangen, dass die Minderung um 10% nicht ausschließlich dadurch zustande gekommen ist.

Ein Ansatzpunkt für eine hierauf aufbauende Arbeit wäre es, mehr Datenpunkte aufzunehmen und eine entsprechende Analyse erneut durchzuführen. Weitere Interviews nach einer längeren Nutzungsphase und die quantitative Auswertung mit mehr Datenpunkten präsentieren durchaus die Möglichkeit einer eindeutigen Aussage darüber, ob kleine Verbesserungen an einem eingesetzten Werkzeug die Testerstellung verbessern könnten. Die bisherigen Aussagen sprechen stark dafür, dass diese Daten von Agilent weiter erhoben werden und zu einem späteren Zeitpunkt zur Auswertung zur Verfügung stehen.

Weiterhin wäre es interessant, diesen Ansatz auch bei anderen Unternehmen oder weiteren Teams innerhalb von Agilent anzuwenden. Alle Modifikationen an SpecFlow können einen Vorteil für andere Teams bereitstellen die Behaviour Driven Development einsetzen. Hier mehr Teams zu involvieren könnte dazu führen, die Wirksamkeit des Ansatzes besser beoder widerlegen zu können. Entsprechende Mehrfallstudien könnten aufschlussreiche Daten darüber geben, ob der Ansatz generalisierbar ist.

In weiteren Arbeiten könnte ebenfalls untersucht werden, inwiefern die bisher nicht so stark eingesetzten Funktionen der Synonym- und Stammsuche eine Auswirkung auf die Testerstellung haben. Der theoretische Vorteil konnte bisher leider nicht untersucht werden, da die Funktionalität zu wenig genutzt wurde. Diese Daten stehen eventuell ebenfalls zu einem späteren Zeitpunkt bei Agilent zur Verfügung, da die Aussage der Mitarbeiter war, dass diese Funktionen in Zukunft genutzt werden sollen. Eine Analyse der Daten zu einem späteren Zeitpunkt könnte damit auch Aussagen über die bisher nicht verwendeten Funktionen liefern.

Literaturverzeichnis

[Alb09] Albers, Sönke ; Klapper, Daniel [. (Hrsg.) ; Konradt, Udo [. (Hrsg.) ; Walter, Achim [. (Hrsg.) ; Wolf, Joachim [. (Hrsg.): *Methodik der empirischen Forschung.* 3., überarbeitete und erweiterte Auflage. Wiesbaden, 2009 (SpringerLink : Bücher)

[BD06] Bortz, Jürgen ; Döring, Nicola: *Forschungsmethoden und Evaluation für Human- und Sozialwissenschaftler : mit ... 87 Tabellen.* 4., überarb. Aufl. Heidelberg : Springer Medizin Verl., 2006 (Springer-Lehrbuch). – ISBN 3–540–33305–3; 978–3–540–33305–0

[Bec03] Beck, Kent: *Test-driven development : by example.* Boston, Mass. : Addison-Wesley, 2003 (The Addison Wesley Signature Series). – ISBN 0–321–14653–0

[Bec10] Beck, Kent ; Andres, Cynthia (Hrsg.): *Extreme programming explained : embrace change.* 2. ed., 9. print. Boston : Addison-Wesley, 2010 (The XP Series). – ISBN 978–0–321–27865–4. – Includes bibliographical references and index

[Che10] Chelimsky, David: *The RSpec book : behaviour-driven development with RSpec, Cucumber, and Friends.* P 1.0 print., December 2010. Raleigh, NC : The Pragmatic Bookshelf, 2010. – ISBN 978–1–93435–637–1; 1–93435–637–9. – Erscheint: 01. April 2010

[Glo13] Gloger, Boris: *Scrum : Produkte zuverlässig und schnell entwickeln.* 4., überarb. Aufl. München : Hanser, 2013. – ISBN 3–446–43338–4; 978–3–446–43338–0

[Hel] Hellesøy, Aslak: *Cukes.info Homepage.* `http://cukes.info/`

[Kni10] Kniberg, Henrik: *Kanban and Scrum - making the most of both.* lulu.com, 2010. – ISBN 9780557138326

[Nor06] North, Dan: *Introducing BDD.* `http://dannorth.net/introducing-bdd/`. Version: März 2006

[RHRR12] Runeson, Per ; Host, Martin ; Rainer, Austen ; Regnell, Bjorn: *Case study research in software engineering : guidelines and examples.* Online-Ausg. Hoboken, N.J, c2012. – Includes bibliographical references (p. 221-233) and index. - Description based on print version record

[SS13] Schwaber, Ken ; Sutherland, Jeff: *The Scrum Guide.* `https://www.scrum.org/Portals/0/Documents/Scrum%20Guides/2013/Scrum-Guide.pdf`. Version: 2013

[TH08] Toutenburg, Helge ; Heumann, Christian ; Schomaker, Michael (Hrsg.) ; Wissmann, Malte (Hrsg.): *Induktive Statistik : eine Einführung mit R und SPSS.* 4., überarb. u. erw. Aufl. Berlin : Springer, 2008 (Springer-Lehrbuch). – ISBN 978–3–540–77509–6

[WH12] Wynne, Matt ; Hellesoy, Aslak: *The Cucumber book : behaviour-driven development for testers and developers*. Dallas. Tex. [u.a.] : Pragmatic Bookshelf, 2012 (The pragmatic programmers). – ISBN 978–1–934356–80–7; 1–934356–80–8. – Erscheint: 01. Dezember 2011

[Yin14] Yin, Robert K.: *Case study research : design and methods*. 5. ed. Los Angeles, Calif. [u.a.] : Sage, 2014. – ISBN 1–4522–4256–9; 978–1–4522–4256–9

Anhang

A. Daten der Testskript-Analyse

Im folgenden sind die Daten der Analyse gelistet. Die BlÃ¶cke sind durch horizontale Linien voneinander getrennt und stellen die analysierten Tests einer einzelnen Person dar.

Tabelle A.1.: Rohdaten aller Tests nach der Einführung der Änderungen an SpecFlow

Testnummer	**# Zeilen**	**# Änderungen**	**Anteil der geänderten Zeilen**
1	32	12	0,375
2	14	2	0,1428571429
3	37	15	0,4054054054
4	22	1	0,0454545455
5	69	4	0,0579710145
6	84	0	0
7	26	0	0
8	80	4	0,05
9	104	0	0
10	85	25	0,2941176471
11	14	0	0
12	9	0	0
13	107	22	0,2056074766

Tabelle A.2.: Rohdaten aller Tests von 2013

Testnummer	**# Zeilen**	**# Änderungen**	**Anteil der geänderten Zeilen**
1	105	16	0,1523809524
2	24	14	0,5833333333
3	100	0	0
4	126	41	0,3253968254
5	15	14	0,9333333333
6	82	71	0,8658536585
7	51	17	0,3333333333
8	24	4	0,1666666667
9	39	11	0,2820512821
10	100	63	0,63
11	47	2	0,0425531915
12	59	18	0,3050847458
13	22	13	0,5909090909

14	37	12	0,3243243243
15	17	12	0,7058823529
16	87	120	1,3793103448
17	78	30	0,3846153846
18	40	16	0,4
19	11	0	0
20	26	0	0
21	43	5	0,1162790698
22	69	24	0,347826087
23	82	23	0,2804878049
24	12	38	3,1666666667
25	32	55	1,71875
26	125	0	0
27	81	16	0,1975308642
28	33	14	0,4242424242
29	13	5	0,3846153846
30	20	0	0
31	19	0	0
32	55	0	0
33	11	0	0
34	10	0	0
35	38	0	0
36	30	8	0,2666666667
37	53	0	0
38	81	2	0,024691358
39	71	28	0,3943661972
40	26	10	0,3846153846
41	71	2	0,0281690141
42	20	8	0,4
43	143	45	0,3146853147
44	90	41	0,4555555556
45	89	4	0,0449438202
46	69	3	0,0434782609
47	80	5	0,0625
48	102	45	0,4411764706
49	63	41	0,6507936508
50	21	3	0,1428571429
51	105	43	0,4095238095
52	16	0	0
53	111	28	0,2522522523
54	204	5	0,0245098039
55	189	1	0,0052910053
56	137	0	0
57	65	65	1
58	44	2	0,0454545455
59	115	0	0
60	16	13	0,8125
61	28	4	0,1428571429
62	50	6	0,12

63	35	0	0
64	30	0	0
65	13	0	0
66	21	18	0,8571428571
67	11	0	0
68	44	0	0
69	63	1	0,0158730159
70	11	1	0,0909090909

Tabelle A.3.: Rohdaten aller Tests vor der Einführung der Änderungen an SpecFlow

Testnummer	**# Zeilen**	**# Änderungen**	**Anteil der geänderten Zeilen**
1	17	8	0,4705882353
2	9	3	0,3333333333
3	20	6	0,3
4	33	6	0,1818181818
5	25	3	0,12
6	26	24	0,9230769231
7	84	90	1,0714285714
8	24	12	0,5
9	24	19	0,7916666667
10	19	41	2,1578947368
11	9	5	0,5555555556
12	60	43	0,7166666667
13	23	8	0,347826087
14	39	4	0,1025641026
15	13	21	1,6153846154
16	22	26	1,1818181818
17	37	18	0,4864864865
18	101	1	0,0099009901
19	10	0	0
20	14	20	1,4285714286
21	105	16	0,1523809524
22	24	14	0,5833333333
23	100	0	0
24	126	41	0,3253968254
25	15	14	0,9333333333
26	82	71	0,8658536585
27	51	17	0,3333333333
28	24	4	0,1666666667
29	39	11	0,2820512821
30	100	63	0,63
31	47	2	0,0425531915
32	59	18	0,3050847458
33	22	13	0,5909090909
34	37	12	0,3243243243
35	17	12	0,7058823529
36	87	120	1,3793103448
37	78	30	0,3846153846
38	40	16	0,4

39	11	0	0
40	26	0	0
41	94	23	0,2446808511
42	43	5	0,1162790698
43	74	3	0,0405405405
44	25	3	0,12
45	37	21	0,5675675676
46	89	24	0,2696629213
47	89	13	0,1460674157
48	308	4	0,012987013
49	30	4	0,1333333333
50	73	8	0,1095890411
51	48	24	0,5
52	69	24	0,347826087
53	11	108	9,8181818182
54	82	14	0,1707317073
55	39	10	0,2564102564
56	33	3	0,0909090909
57	25	4	0,16
58	27	5	0,1851851852
59	35	0	0
60	135	174	1,2888888889
61	65	22	0,3384615385
62	82	23	0,2804878049
63	32	55	1,71875
64	125	0	0
65	81	16	0,1975308642
66	33	14	0,4242424242
67	13	5	0,3846153846
68	198	0	0
69	20	0	0
70	53	144	2,7169811321
71	182	26	0,1428571429
72	21	18	0,8571428571
73	11	0	0
74	44	0	0
75	63	1	0,0158730159
76	11	1	0,0909090909
77	81	2	0,024691358
78	151	6	0,0397350993
79	118	38	0,3220338983
80	71	28	0,3943661972
81	88	88	1
82	122	33	0,2704918033
83	139	50	0,3597122302
84	150	57	0,38
85	26	10	0,3846153846
86	92	3	0,0326086957
87	238	18	0,0756302521
88	94	145	1,5425531915

89	71	2	0,0281690141
90	104	3	0,0288461538
91	121	9	0,0743801653
92	92	18	0,1956521739
93	193	10	0,0518134715
94	120	12	0,1
95	20	8	0,4
96	87	9	0,1034482759
97	12	38	3,1666666667
98	143	45	0,3146853147
99	90	41	0,4555555556
100	89	4	0,0449438202
101	69	3	0,0434782609
102	80	5	0,0625
103	102	45	0,4411764706
104	63	41	0,6507936508
105	21	3	0,1428571429
106	105	43	0,4095238095
107	16	0	0
108	123	0	0
109	111	28	0,2522522523
110	82	0	0
111	19	0	0
112	32	0	0
113	76	0	0
114	110	17	0,1545454545
115	31	0	0
116	10	0	0
117	34	3	0,0882352941
118	8	0	0
119	10	9	0,9
120	45	0	0
121	10	0	0
122	14	0	0
123	8	0	0
124	24	6	0,25
125	42	10	0,2380952381
126	12	0	0
127	29	2	0,0689655172
128	32	0	0
129	15	1	0,0666666667
130	15	1	0,0666666667
131	85	0	0
132	35	0	0
133	25	0	0
134	23	0	0
135	12	6	0,5
136	77	0	0
137	18	0	0
138	7	0	0

139	25	0	0
140	10	0	0
141	9	0	0
142	11	0	0
143	24	6	0,25
144	16	10	625
145	35	9	0,2571428571
146	116	79	0,6810344828
147	72	37	0,5138888889
148	54	0	0
149	143	19	0,1328671329
150	79	0	0
151	204	5	0,0245098039
152	189	1	0,0052910053
153	48	0	0
154	27	0	0
155	44	0	0
156	137	0	0
157	77	8	0,1038961039
158	126	0	0
159	65	65	1
160	44	2	0,0454545455
161	69	40	0,5797101449
162	28	1	0,0357142857
163	28	21	0,75
164	197	0	0
165	17	0	0
166	146	0	0
167	139	0	0
168	188	0	0
169	115	0	0
170	16	13	0,8125
171	28	4	0,1428571429
172	50	6	0,12
173	35	0	0
174	30	0	0
175	13	0	0
176	33	4	0,1212121212
177	90	13	0,1444444444
178	9	1	0,1111111111
179	76	5	0,0657894737
180	19	0	0
181	55	0	0
182	11	0	0
183	10	0	0
184	38	0	0
185	88	4	0,0454545455
186	30	8	0,2666666667
187	53	0	0

188	28	0	0
189	11	2	0,1818181818
190	35	0	0