HOCHSCHULE
MITTWEIDA
UNIVERSITY OF
APPLIED SCIENCES

BACHELORARBEIT

Herr
Lukas Vorberg

Verhaltensgetriebene Entwicklung mit PHPspec und Behat

Hormersdorf, 25. Juli 2014

Fakultät: Elektro- und Informationstechnik

BACHELORARBEIT

Verhaltensgetriebene Entwicklung mit PHPspec und Behat

Autor/in:
Herr
Lukas Vorberg

Studiengang:
Multimediatechnik

Seminargruppe:
MK 10 w1-B

Erstprüfer:
Prof. Dr.-Ing.Frank Zimmer

Zweitprüfer:
Dipl. Ing. Stephan Reuther

Faculty of Information Technology
& Electrical Engineering

BACHELOR THESIS

Behavior Driven Development with PHPspec and Behat

author:
Mr.
Lukas Vorberg

course of studies:
multimedia technics

seminar group:
MK 10 w1-B

first examiner:
professor dr. engineer Frank Zimmer

second examiner:
diploma engineer Stephan Reuther

Bibliografische Angaben

Vorberg, Lukas:

Verhaltensgetriebene Entwicklung mit PHPspec und Behat

Behavior Driven Development with PHPspec und Behat

62 Seiten, Hochschule Mittweida, University of Applied Sciences,
Fakultät EIT, Bachelorarbeit, 2014

Abstract

Codequalität spielt eine große Rolle in der Softwareentwicklung. Deswegen wir andauernd nach neuen Möglichkeiten geforscht um diese zu verbessern. Verhaltensgetriebene Entwicklung und deren Programme sind relativ neu auf dem Markt. Aber der Anteil wächst stetig. Verhaltensgetriebene Entwicklung erfreut sich immer größer werdender Beliebtheit sowie Unternehmen die diese Methodik verwenden.

Diese Arbeit beschäftigt sich mit dem Potenzial was in der Verhaltensgetriebenen Entwicklung steckt. Hat diese Art zu Entwickeln einen Mehrwert in einem Softwareunternehmen und wenn ja welchen. Weiterhin soll untersucht wie die Einarbeitung neuer Mitarbeiter in diese Methodik funktioniert und es letztendlich ein Gewinn für das Unternehmen ist.

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

1.1 Motivation

Wenn ein Kunde eine Firma beauftragt für ihn ein Produkt anzufertigen, dann wird erwartet, dass dieses Produkt den vorher definierten Kriterien entspricht. Des Weiteren verfolgt das Entwickler-Unternehmen auch Kriterien, die es möglichst gut umzusetzen gilt. Geringe Zeit, hohe Produktivität, geringer Aufwand, gute Codearchitektur, gute Erweiterbarkeit, geringe Fehleranfälligkeit - dies sind alles Faktoren, die eine große Rolle in der Softwareentwicklung spielen.

Diese Faktoren umzusetzen, ist jedoch schwieriger als man denkt. Jeder Entwickler bringt sein eigenen Fingerabdruck mit in den Programmcode hinein. Er beschreibt seine Funktionen, Klassen und Methoden anders als der Andere. Somit wird die Ermittlung, was hinter den Namen der Funktionen, Klassen und Methoden steckt, unnötig kompliziert. Das gleiche gilt für die Einarbeitung neuer Mitarbeiter in ein vorhandenes Projekt bzw. um ein bestehendes Projekt weiterzuführen.

Eine weitere Wichtigkeit bei der Entwicklung einer Software ist, das sie fehlerfrei ist. Wie realisiert man dies? Häufig wird Software per Hand getestet. Aber was passiert, wenn Software regelmäßig erweitert bzw. verändert wird? Der Auftraggeber wird nur im seltensten Fall einen erneuten Test per Hand bezahlen. Somit wird häufig ungetesteter Programmcode ausgeliefert.

Diese Bachelor-Arbeit wird aufzeigen, wie die Entwicklung einer Software mit aktuellen Programmiermethoden aussehen kann und welcher Mehrwert dahinter steckt.

1.2 Zielsetzung

Im Laufe der Zeit wurde immer wieder nach neuen Wegen gesucht, dass gute Codequalität umgesetzt und gewährleistet werden kann. Mittlerweile existieren viele Methoden, um diesen Anspruch gerecht zu werden. Testgetriebene Entwicklung (kurz: TDD) ist eine der Methoden, die in der Vergangenheit sich großer Beliebtheit erfreute. Allerdings gilt diese Methode unter Programmierern nicht mehr als zeitgemäß. Somit wurde die Verhaltensgetriebene Entwicklung (kurz: BDD) entwickelt. BDD wird bereits in vielen Softwareunternehmen eingesetzt die Community hinter BDD sorgt mit neuen und aktualisierten Funktionen für viel Dynamik in der BDD-Welt. Ist BDD nur ein Hype? Oder steckt hinter dieser Methode eine neue und gute Art, ein Softwareprodukt zu entwickeln?

BDD ist eine Methode bzw. Herangehensweise, um Entwickler zu unterstützen Softwarelösungen für verschiedene Programmiersprachen zu entwickeln. Aber lohnt sich BDD auch für die PHP-Entwicklung? Es existieren 2 Programme die populär in der PHP-Welt sind, PHPspec und Behat. Es soll der Frage nachgegangen werden, wie PHPspec und Behat funktionieren und wie deren Aufbau ist. Schlussendlich soll auch geprüft werden, ob PHPspec und Behat zu empfehlen ist und ob deren Popularität gerechtfertigt ist. Wie kann BDD mit PHPspec und Behat in ein Unternehmen integriert werden und welchen Mehrwert bietet es den Entwicklern? Des Weiteren soll geprüft werden, wie sich die Entwicklungsprozesse durch diese Methoden verändern. Wie können Kunden, „Qualitäts-Sicherung“ (kurz QS) zuständige Mitarbeiter und Entwickler durch BDD Einfluss auf den Entwicklungsprozess einer Softwarelösung nehmen?

1.3 Grenzen und Bedingungen der Arbeit

Die vorliegende Arbeit richtet sich an Personen mit Grundkenntnissen im Bereich Web- und Software-Entwicklung. Die Arbeit beschäftigt sich vorrangig mit der Verhaltensgetriebenen Entwicklung mittels Behat und PHPspec. Allerdings kann mit dieser Forschung kein umfangreicher und vollständiger Vergleich aller Programmiermethoden und Programmen stattfinden, die aktuell von Unternehmen verwendet werden. Die Betrachtung der verschiedenen Entwicklungsmethoden würde den zeitlichen Rahmen deutlich übersteigen.

Der Kernpunkt der Arbeit liegt in der Auswertung der gestellten Kriterien an PHPspec und Behat. Um die Kriterien zu testen, wurde eine Erweiterung für einen bestehenden Onlineshop entwickelt.

1.4 Aufbau der Arbeit

Die Bachelorarbeit gliedert sich insgesamt in 6 Kapitel.

Im Kapitel „**Grundlagen**“ wird ein Einblick über Codequalität gegeben, sowie über fundamentale Kenntnisse über „testgetriebener Entwicklung“ als auch „verhaltensgetriebener Entwicklung“. Des Weiteren wird in diesem Kapitel erklärt, wie die Entwicklungsumgebung aufgebaut ist, auf der die Kriterien überprüft werden.

Das Kapitel „**Kriterien**“ umfasst alle Faktoren nach deren überprüft werden soll und die beiden Programme PHPspec und Behat getestet werden.

Das Kapitel „**PHPspec**“ beinhaltet eine ausführliche Beschreibung der Funktionsweise sowie grundlegendes Know-How. Weiterhin werden Resultate der aufgeführten Kriterien geprüft und geschildert.

Im Kapitel „**Behat**“ wird ebenfalls die Funktionsweise, das Verhalten sowie die Vor- und Nachteile näher beleuchtet und beschrieben. Wie schon bei PHPspec werden auch in diesem Kapitel Resultate von Tests präsentiert und näher erläutert.

Das „**Fazit**“ beurteilt die Programme und beschreibt Vor- und Nachteile die für bzw. gegen eine Verwendung von „verhaltensgetriebener Entwicklung“ mit PHPspec und Behat stehen.

Am Schluss wird ein „**Ausblick**“ gegeben, wozu die Arbeit nützlich ist und was man auf dessen Grundlage verwirklichen kann.

2 Grundlagen

2.1 Codequalität

Codequalität ist ein Teilaspekt der Softwarequalität und somit essentiell für die Softwareentwickling.

Folgende Parameter[1] beschreiben, was die Qualität eines Programmcodes ausmacht:

1) Lesbarkeit
    a. Der Programmcode sollte für einen Entwickler einfach zu lesen und zu verstehen sein.
    b. Prägnant sind dabei die Klassen-, Methoden- und Variablenbezeichnungen.
2) Testbarkeit
    a. Der Aufbau des Programmcodes sollte so sein, dass der spätere Import in einen Test funktioniert - sofern nicht schon test- bzw. verhaltensgetrieben entwickelt wurde.
3) Variabilität
    a. Der Programmcode muss einfach anzupassen und erweiterbar sein.
4) Flexibilität
    a. Abhängigkeiten im eigenen Programmcode-Fundament und den Implementierungen sollten gering gehalten werden.
    b. Statisch programmierte Voraussetzungen über Datengröße, konkrete Klassen sowie Datenstrukturen machen den Programmcode schwieriger adaptierbar.
5) Performance
    a. Die Verwendung der Systemressourcen durch den Programmcode sollte unnötige Belastung der CPU oder dem Hauptspeicher vermeiden.

---

[1] vgl. http://de.wikipedia.org/wiki/Codequalität%C3%A4t
http://stackoverflow.com/questions/405243/how-do-we-define-code-quality

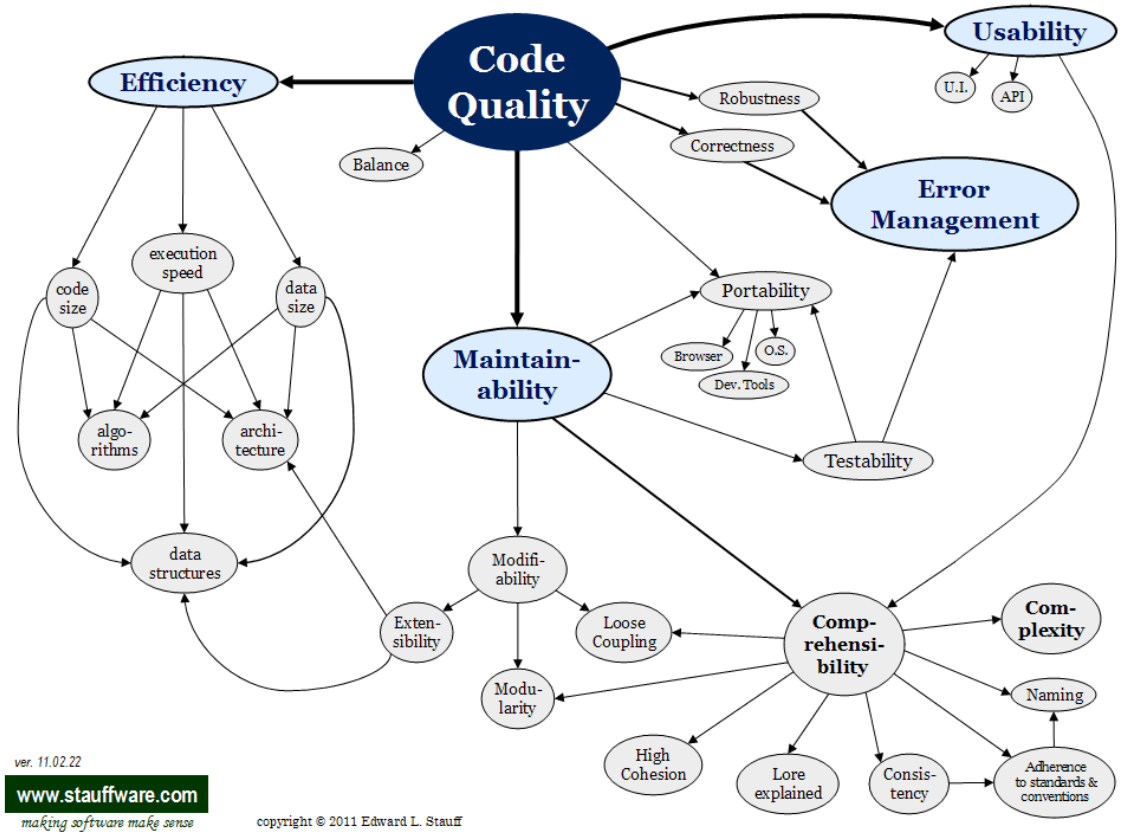

Abbildung 1 Skizze, Definition von Codequalität[2]

An diesen Parametern wird erkennbar, dass während des gesamten Prozesses einer Softwareentwicklung die Codequalität im Auge behalten werden muss.

Um die bestmöglichste Codequalität zu garantieren, existieren verschiedene Ansätze den Entwickler zu unterstützen. Schon bei der Analyse der Anforderungen und dem Design für eine Funktion, versuchen verhaltens- und testgetriebene Ansätze den Entwickler entscheidend zu unterstützen.

---

[2] vgl. http://www.stauffware.com/d_and_d/CodeQuality.png

2.2 Testgetriebene Entwicklung

Wenn eine neue Funktionalität in einem Programm implementiert bzw. eine Funktionalität angepasst und erweitert werden soll, wie stellt man sicher, dass es im Nachhinein zu keinerlei Problemen kommt? Die Funktionalität per Hand zu testen ist aus mehreren Gründen nicht vorteilhaft.

Die testgetriebene Entwicklung versucht dieses Problem zu beheben. Um dieses Ziel zu erreichen, wird als Erstes mit einem Test die Funktionalität spezifiziert. Nach der Fertigstellung des Tests wird der Programmcode entwickelt.

Das führt dazu, dass nun die komplette Funktionalität überprüft werden kann. Wenn unerwünschte Seiteneffekte entstehen, die auf Grund einer Codeänderung an einer anderen Stelle auftreten, können sie nun durch die Tests herausgefunden und dadurch behoben werden.

2.2.1 Testgetriebener Entwicklungszyklus

TDD folgt folgendem Ablauf[3]:

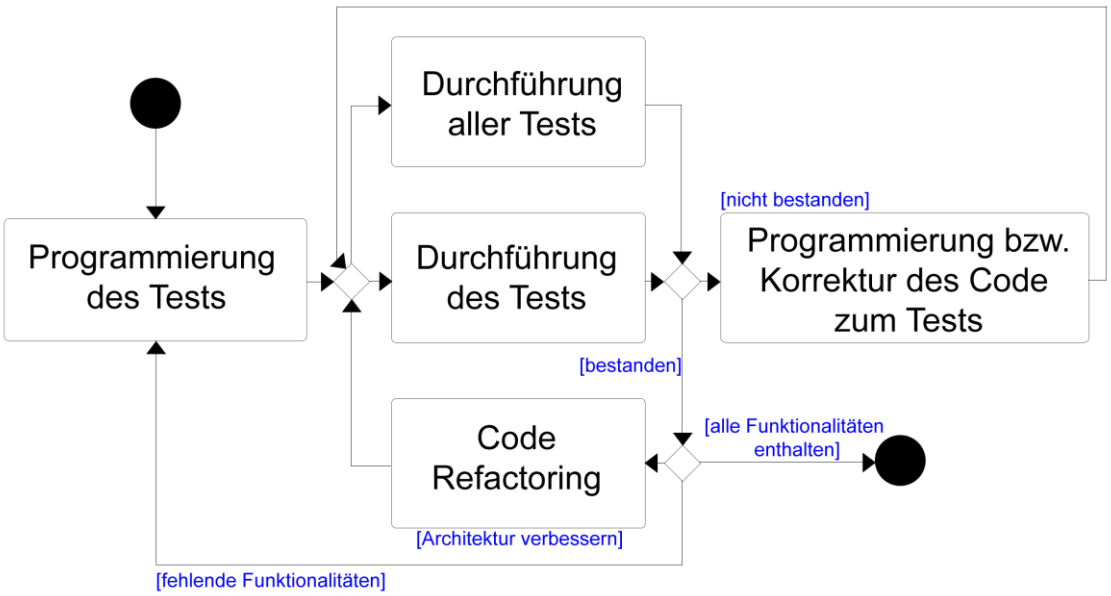

Abbildung 2 Zyklus einer testgetriebenen Entwicklung

[3] vgl. http://daraff.ch/2010/01/einfuhrung-test-driven-development/

Wie der Abbildung zu entnehmen ist, beginnt der Entwicklungsprozess mit der Entwicklung eines Tests. Dieser Test wird ausgeführt und schlägt fehl.

Nun folgt die Implementierung des Programmcodes bis der Test erfolgreich ist.

Danach ist es möglich, alle Tests durchzuführen, um mögliche Seiteneffekte aufzudecken und zu beheben.

Nach dieser erneuten Prüfung ist es dem Entwickler gestattet, ein „Code Refactoring“ durchzuführen. Hierbei wird die Architektur des Programmcodes verbessert und Redundanzen entfernt. Daraufhin muss ein erneuter Testdurchlauf stattfinden um sicherzustellen, dass alle Funktionalitäten funktionieren. Ist das „Code Refactoring“ abgeschlossen, ist es möglich eine weitere Funktionalität zu implementieren und den beschriebenen Zyklus erneut zu durchlaufen. Sind alle gewünschten Funktionalitäten enthalten und die Architektur optimiert, ist die Entwicklung abgeschlossen.

2.2.2 Testgetriebene Entwicklung als Designstrategie

Testgetriebene Entwicklung (TDD) eignet sich hervorragend, um Anwendungen auf eventuelle Fehler zu überprüfen. Aber TDD ist mehr als nur eine Methode zum Testen.

Die Vorgehensweise, zuerst einen Test für die gewünschte Funktionalität zu schreiben, ermöglicht dem Entwickler frühestmöglich zu erkennen, ob das Design der Funktionalität verwendbar sein wird.

Die Prinzipien der testgetriebenen Entwicklung sind kontinuierliche Designverbesserungen.[4]

2.2.3 Nachteile der Testgetriebenen Entwicklung

Dan North, Dave Astels und weitere Lehrer im Bereich TDD sind zu folgenden Schluss gekommen:

„It must be possible to present TDD in a way that gets straight to the good stuff and avoids all the pitfalls..“[5]

---

[4] vgl. http://www.it-agile.de/wissen/praktiken/agiles-testen/testgetriebene-entwicklung-tdd/

[5] vgl. http://dannorth.net/introducing-bdd/

Nutzer von TDD haben sich immer wieder dieselben Fragen zu stellen:

- Wo fängt man zu testen an?
- Was muss getestet werden?
- Was muss nicht getestet werden?
- Wieviel muss getestet werden?
- Wie benennt man die Tests?
- Wie versteht man warum ein Test fehlschlägt?

Der Grund für diese immer wiederkehrenden Fragen liegt im Vokabular, das TDD verwendet. Wörter wie “TestSuite” oder “TestCase” lassen erahnen, dass es um das “Testen” geht. Aber wie im Punk 2.2.2 beschrieben, ist dies nur ein Nebenprodukt.

TDD ist eine Designstrategie.

Programme wie “PHPunit”, die für TDD entwickelt wurden, lassen erahnen, dass es sicher um Klassen und Methoden und deren korrekte Ausführung geht. Vielmehr ist der Fokus von TDD auf das Verhalten der Software gerichtet.

Aufgrund von diesen Problemen wurde BDD ins Leben gerufen.

2.3 Verhaltensgetriebene Entwicklung

BDD wurde ursprünglich 2003 von Dan North als Weiterentwicklung von TDD bekannt gemacht.[6]

Dan North führte dabei syntaktische Konventionen für Unit Tests ein. Als „Unit-Tests“ bezeichnet man Überprüfungen ob Komponenten wie gewünscht funktionieren. Er entwickelte „JBehave“ als Ersatz für „JUnit“, das alle verwandten Wörter von „Test“ mit dem Wort „Verhalten“ ersetzt hat. „JUnit“ ist ein Framework zum Testen von Java-Programmen welches von „JBehave“ durch veränderte Namenskonventionen abgelöst wurde.

Wieso führte Dan North eine Vokabular Umstellung durch? Edward Sapir und Benjamin Whorf bildeten eine Hypothese die aussagt, dass die Sprache, die wir nutzen, unser Denken beeinflusst[7]. Wollen wir unsere Denkweise verändern, hilft es demzufolge nach, die Sprache zu verändern.

Die Testgetriebene Entwicklung führte dazu, dass viele Entwickler den Entwicklungszyklus nicht optimal verwendet haben. Deswegen kam Dan North auf die Idee, durch Namenskonventionen das Verhalten in den Mittelpunkt zu rücken.

Die Basis von BDD sind flexible Methoden, die darauf abzielen, Teams mit wenig Erfahrung in agiler Softwareentwicklung, den Einstieg zugänglicher und effizienter zu gestalten.

2.3.1 Syntaktische Konventionen

Von „test“ zu „should“

Das Verb “test” wurde mit dem Hilfsverb “should” ergänzt. “Die Klasse soll (should) folgendes tun” bedeutet, dass ein Test nur für die aktuelle Klasse definiert werden kann.

Dadurch behält man den Fokus und erreicht, dass wenn ein Test nicht in die Klassenbezeichnung passt, das Verhalten in eine andere Klasse ausgelagert werden

---

6 vgl. http://www.ymc.ch/behavior-driven-development-with-behat-co-mehr-als-nur-testen

7 vgl. http://de.wikipedia.org/wiki/Sapir-Whorf-Hypothese

kann. Die Absicht hinter “should” ist, den Fokus des Entwicklers auf das Verhalten der Funktionalität zu richten und nicht auf die Entwicklung automatisierter Tests. Durch automatisierte Tests versucht man eine möglichst hohe Testabdeckung zu erreichen.

Bezeichnungen von Methoden und Klassen

Ein aussagekräftiger Methodenname oder Klassenname ermöglicht eine sofortige Identifizierung, welche Funktion die Methode oder Klasse hat.

Deswegen entschloss sich Dan North dazu, Methodennamen und Klassennamen als einen ganzen Satz zu definieren[8]. Somit ergeben sich in Verbindung von Klassen- und Methodennamen lesbare Sätze.

Weiterhin ist solch eine Konvention sinnvoll, wenn ein Test fehlschlägt. Die Ausgabe des Tests ist dadurch übersichtlicher, weil man sofort erkennen kann, um welche Methode bzw. Klasse es sich handelt und welche Funktionalität diese Methode und Klasse erfüllen soll.

Bezeichnungen der von TDD verwendeten Begriffe

Der Wechsel von „Test“ zu „Verhalten“ hat die Sichtweise der Funktion von Tests entscheidend verändert. Dan North berichtete, dass nach dieser Umbenennung viele seiner Schüler einen „A-Ha Effekt“ bekommen hatten.[9]

Fragen, wie „Wie soll der Test heißen?“ oder „Wie viel muss getestet werden?“, wurden irrelevant, weil man nun wusste, dass es ein Satz sein soll, der beschreibt, was als nächstes passieren soll und da es nun offensichtlich wurde, dass das Verhalten zu überprüfen sei.

Viele Fragen werden schließlich von allein beantwortet, da man nicht mehr an einen Test denkt, sondern an ein Verhalten was überprüft werden soll.

Statt “Validierung der Software” spricht man fortwährend nur noch von “Spezifikation der Software”. Einen Testfall (engl. Testcase) gibt es nun auch nicht mehr, denn man bezeichnet es als „Kontext“.

---

[8] vgl. http://www.ymc.ch/behavior-driven-development-with-behat-co-mehr-als-nur-testen

[9] vgl. http://dannorth.net/introducing-bdd/

Die universelle Sprache

Bis zu diesem Punkt hat sich BDD als eine große Hilfe für Programmierer herausgestellt. Allerdings schaffte BDD nicht alle am Projekt Beteiligten mit einzubeziehen und für Nicht-Programmierer einen Mehrwert daraus erkennbar zu machen. Aufgrund dieser Problematik wurde eine universelle Sprache eingeführt, um die Anforderungen an das Projekt zu spezifizieren. Diese universelle Sprache ermöglicht es, dass Analysten, Tester, Entwickler und Kunden eine gemeinsame Sprache sprechen.

Anhand dieser Sprache, die ein konsistentes Vokabular stellt, werden Mehrdeutigkeiten und Missverständnisse verhindert. Diese traten vor allem auf, wenn technische Mitarbeiter mit Mitgliedern des Managements sprachen.

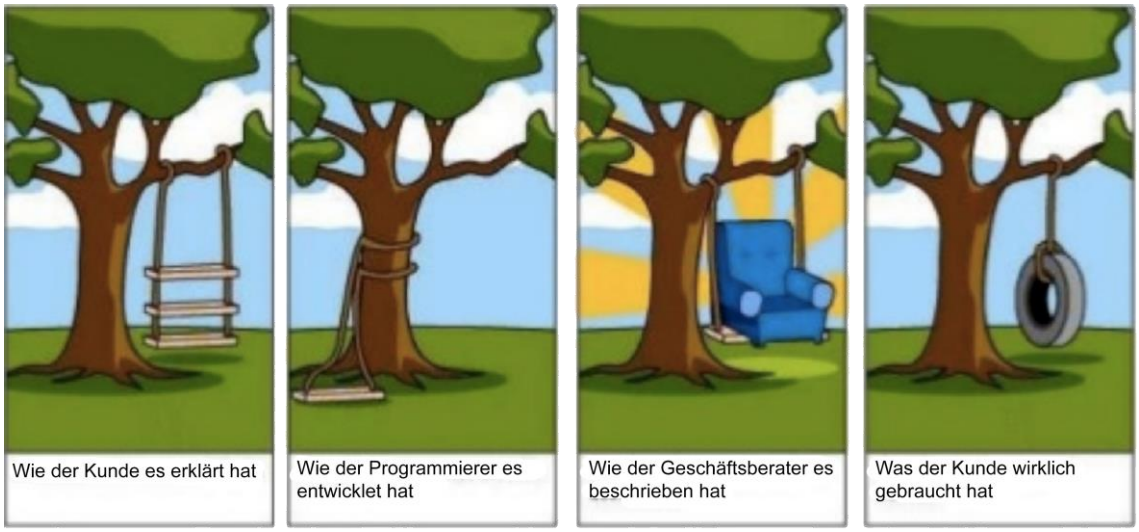

Abbildung 3 Missverständnisse in der Teamkommunikation[10]

[10] vgl. https://github.com/VKMohan/JustDocs/blob/master/Vinod%20Kumar%20M/JustDocs/cucumberpresentation-anztbsigist-110720060144-phpapp02.ppt, Seite 3

2.3.2 Die universelle Sprache als Grundlage für teamorientierte Programmierung

Durch eine gemeinsame Sprache wird die Möglichkeit gegeben, sich innerhalb des Teams, dem Kunden und dem zukünftigen Nutzer, abzusprechen und auch zu beraten. BDD ermöglicht fortan abgeleitete Dokumentationen sowie gemeinsam stattfindende Diskussionen bevor die Funktionalität entwickelt wird.

2.3.3 BDD als agile Methode

Dan North sagte folgendes zu BDD:

„Behavior-Driven Development is about implementing an application by describing is behavior from the perspective of is stakeholders“[11]

Das Ziel von BDD ist die Implementierung einer Funktionalität durch Beschreibung ihres Verhaltens aus der Sicht des Kunden. Die Ansichtsweise des Kunden muss verstanden werden, um die Kriterien, die an die Funktionalität gestellt werden, zu erfüllen.

Hierbei tritt die universelle Sprache in Kraft, die bei der Vermittlung zwischen Entwickler und Kunden hilft. Durch diesen Ansatz fokussieren sich Entwickler nicht darauf wie ein Objekt aufgebaut ist, sondern im Vordergrund stehen die Interaktionen zwischen Personen und dem System.

---

[11] vgl. *David Chelimsky, Dave Astels, Bryan Helmkamp, Dan North, Zach Dennis, and Aslak Hellesoy. The RSpec Book - Behaviour-Driven Development with RSpec, Cucumber, and Friends. Pragmatic Programmers, 2010.*

2.3.4 Verhaltensgetriebener Entwicklungszyklus

Wie bei TDD findet auch bei BDD ein zyklischer Entwicklungs-Prozess statt.

Schritt 1: Entwicklung von Automated Acceptance Tests

Als Erstes wird das Verhalten der Funktionalität beschrieben. Diese werden „Automated Acceptance Tests“ genannt.

Der Sinn besteht darin, dass ein Kunde keine Liste mit den ganzen Anforderungen einfach dem Entwickler weiterreicht. Vielmehr ist dies eine Möglichkeit, dass Kunde und Entwickler beim Beschreiben des Verhaltens zusammenarbeiten und sich gegenseitig Feedback geben können.

Der Unterschied zu Unit Tests besteht darin, dass dieser Test nicht an Entwickler gerichtet ist, sondern die Kriterien der Funktionalität wiederspiegelt aus Sicht des Kunden. Damit der Automated Acceptance Test von Kunde und Entwickler lesbar und bewertbar ist, werden Schlüsselwörter eingeführt.

Die Schlüsselwörter „Given“, „When“ und „Then“ dienen dazu, die Sprache des Tests möglichst einfach zuhalten.

Given a precondition
When an event occurred
Then an outcome is achieved

„Given“ bezeichnet den Kontext für das Szenario.
„When“ beschreibt, dass eine Aktion bzw. ein Ereignis ausgelöst wird.
„Then“ stellt das erwartete Resultat der Aktion dar.

Bank-User-Story

As an Account Holder I want to withdraw cash from ATM so that I can get money when the bank is closed

Given the account is balance \$100
And the card is valid
And the machine contains enough money
When the Account Holder requests \$20
Then the ATM should dispense \$20

```
And the account balance should be \$80
And the card should be returned
```

Der mit „Given“ hergestellte Kontext beschreibt ein Bankkonto mit einem Guthaben von 100$. Des Weiteren wurde definiert, dass die Bankkarte valide ist und der Bankautomat genügend Geld beinhaltet. Wenn der Kontoinhaber 20$ abheben möchte, dann gibt der Geldautomat 20$ raus und der aktuelle Kontostand beträgt 80$ und die Bankkarte wird wieder ausgegeben.

Schritt 2: Implementierung

Nachdem der Automated Acceptance Test abgeschlossen wurde, beginnt nun die Implementierung.

Es wird schrittweise der Acceptance Test abgearbeitet - Zeile für Zeile. Demzufolge muss zuerst ein Konto mit 100$ Guthaben implementiert werden. Das Vorgehen kann mithilfe von TDD realisiert werden. Das TDD Vorgehen bezeichnet den „red-green-refactor“ Zyklus, wie im nachfolgenden Bild zu entnehmen ist. Zunächst wird ein Szenario entwickelt was fehlschlägt. Danach wird Schrittweise jede Zeile implementiert, bis der Test erfolgreich ist. Anschließend wird diese Implementierung refaktorisiert.

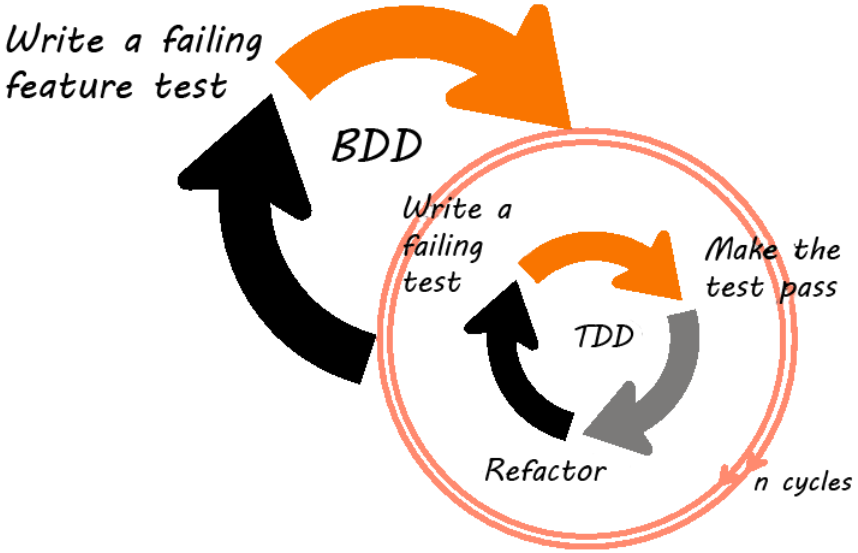

Abbildung 4 BDD Zyklus[12]

2.3.5 Vorteile

Tests als lebendige Dokumentation[13]

Den Szenarien ist die Spezifikation der Funktionalität zu entnehmen. Das bietet den Vorteil, dass sie von Kunden sowie Entwicklern gleichermaßen gelesen werden kann. Des Weiteren zeigt diese „Dokumentation“ beim Ausführen eine Rückmeldung an, inwieweit die Funktionalität noch den Anforderungen entspricht. Aufgrund dieses Verhaltens veraltet die Dokumentation nicht, wie es bei traditionellen Dokumentationen der Fall wäre. Weiterhin wird auf Grund des zentralen Speicherorts Zeit eingespart, um Code, Test und Dokumentation synchron zu halten. Durch diese Dokumentation erkennt man, was der Entwickler versucht zu realisieren. Gerade wenn die Funktionalität von einem anderen Entwickler erweitert werden soll, ist dies sehr hilfreich.

---

12 vgl. http://blog.bughuntress.com/wp-content/uploads/2013/09/bdd-cycle-around-tdd-cycles1.png

13 vgl. http://www.mehdi-khalili.com/bdd-to-the-rescue

Flexibles Design

Bei Erweiterung der Software muss häufig das Architektur des Codes angepasst werden. Aber durch die Tests kann die Software geprüft werden, nachdem Änderungen implementiert wurden. Somit lassen sich schnell und effizient die Änderungen überprüfen.

Universelle Sprache

Wie bereits erwähnt ist die „universelle Sprache“ eine der großen Stärken von BDD. Durch Programme und Techniken wie User-Stories, Acceptance Tests, Story-Maps und Szenarien, die das Entwicklerteam und das Management verwenden, entsteht eine Sprache, die jeder Mitarbeiter im Team und der Kunde verstehen kann.

BDD als teamorientierte Entwicklung

BDD gibt keine hundertprozentige Lösung für die allgemeinen Kommunikationsprobleme in einem Team. Aber es ermutigt, sich auszutauschen, gemeinsam, im Team und mit dem Kunden, Kriterien für die Funktionalität zu besprechen, Funktionen zu priorisieren und das Design der Software aus der Sicht des Kunden zu entwickeln.

Überlegtes Entwickeln

Viele Entwickler stürzen sich förmlich in das Programmieren einer Software hinein ohne wichtige Vorüberlegungen zu treffen. Durch BDD wird der Entwickler gezwungen sich über die Kriterien der Funktionalität Gedanken zu machen.

Der richtige Fokus

BDD hilft den Fokus auf die Nöte des Kunden zu setzen. Der Fokus liegt nicht mehr auf Tests, Klassen oder Methoden. Stattdessen ist das Verhalten der Funktionalität der entscheidende Aspekt.

Das Selbstvertrauen des Entwicklers

Durch eine Testabdeckung der entwickelten Funktionalitäten wird größtenteils verhindert, dass unerwünschte Seiteneffekte auftreten. Durch die Tests entsteht beim Entwickler mehr und mehr Vertrauen in den eigenen Code und verhindert, dass Endnutzer Fehler in der Software entdecken.

2.3.6 Nachteil

Der ausführbare Code in den Szenarien erstreckt sich intern über mehrere Methoden. Dadurch ist es im Vergleich zu TDD aufwendiger den internen Ablauf eines Szenarios nachzuvollziehen.

2.4 Entwicklungsumgebung

2.4.1 PHP-Projekt

Im Vorfeld dieser Bachelorarbeit wurde ein fiktiver Onlineshop entwickelt. Dieser basiert auf dem PHP-Framework „Symfony“, welches später näher erläutert wird.

Der Onlineshop hat sich auf den Vertrieb von Büchern spezialisiert. Bei Beginn der Bachelorarbeit lag der Funktionsumfang auf folgende Bereiche:

Startseite

Die Startseite des Onlineshops beinhaltet eine komplette Liste aller vorhandenen Bücher mit näheren Informationen zu Preis, Titel, Autor und Kategorie. Dazu besteht bei jedem Buch die Möglichkeit es in der Einzelansicht aufzurufen und in den Warenkorb zu legen, die Menge anzugeben und den kompletten Warenkorb zu entleeren.

Einzelansicht

Die Einzelansicht enthält eine detaillierte Ansicht eines einzelnen Artikels mit Informationen zu Preis, Titel, Autor, Kategorie sowie einer ausführlichen Beschreibung. Ebenfalls ist eine Möglichkeit enthalten den Artikel in den Warenkorb zu legen.

Warenkorb

Der Warenkorb ist auf der Startseite sowie der Einzelansicht sichtbar und listet alle Artikel mit Menge sowie einem Gesamtpreis auf. Weiterhin gibt es darüber die Möglichkeit den Warenkorb zu entleeren und die Artikel in dem Warenkorb zu bezahlen.

Das Design dieses Onlineshops ist auf das nötigste reduziert, da der Schwerpunkt auf die Entwicklung einer neuen Funktionalität liegt.

2.4.2 Symfony

Symfony ist ein PHP-Framework. Um besser nachzuvollziehen was Symfony eigentlich ist, wird nachfolgend der Begriff Framework erläutert.

Was ist ein Framework

Ein Framework stellt eine grundlegende Architektur sowie viele Basisfunktionalitäten zur Verfügung. Das erspart dem Entwickler sehr viel Zeit. Deswegen ist es nicht nötig bei jedem Projekt „das Rad“ neu zu erfinden, sondern es ist möglich vorgefertigte Module (Bundles) einzubinden. Mittels eines Frameworks wird der Programmieraufwand sowie das Risiko einen Fehler zu verursachen geringer. Durch die Nutzung einer grundlegenden Architektur und vorgefertigten Bundles steigt die Produktivität und somit steht mehr Zeit zur Verfügung, um sich auf das Wesentliche zu konzentrieren.

Das grundlegende Prinzip von Frameworks ist: „Investiere in die Aufgabe, nicht in die Technologie“[14]. Wenn zum Beispiel ein Authentifizierungsformular vom Kunden gewünscht ist, dauert die Entwicklung 2-3 Tage. Bei einem Framework wie Symfony stehen vorgefertigte Module frei zur Verfügung um genau diese Funktionalität zu erfüllen. Demzufolge erspart sich der Programmierer hiermit 2-3 Tage und kann sich dementsprechend um wichtigere Komponenten kümmern.

Was ist Symfony

Symfony wurde von der französisch-stämmigen Firma „SensioLabs“ unter der Leitung von „Fabien Potencier“ im Jahr 2005 ins Leben gerufen und stets weiterentwickelt. Momentan ist die aktuelle Version von Symfony 2.5. Der Grund zur Entwicklung des Frameworks „Symfony“, war der Start des Frameworks „Ruby on Rails“. Entwickler wünschten sich vergleichbares für PHP verwenden zu können. Aus diesem Wunsch wurde „Symfony“ ins Leben gerufen.[15]

---

[14] vgl. http://symfony.com/why-use-a-framework

[15] vgl. http://de.wikipedia.org/wiki/Symfony

Symfony ist ein MVC-Framework. MVC bezeichnet eine Strukturierung für die Softwareentwicklung in Datenmodell, Präsentation und Programmsteuerung. D.h. durch das MVC-Entwurfsmuster ist der Quelltext sehr sauber und strukturiert. Eine Änderung ist deswegen wesentlich einfacher möglich. Das Framework unterstützt viele erprobte Komponenten sowie Design-Paradigmen der modernen Software-Architektur.

Vorteile von Symfony

- Bundles
    - Das Framework besteht aus Modulen (Bundles), die voneinander vollkommen unabhängig nutzbar sind.
    - Die Bundles sind Funktionalitäten, wie z.B.: ein Gästebuch-Bundle, welches alle Daten enthält, die für ein funktionierendes Gästebuch notwendig sind.
    - Bundles enthalten nicht nur Klassen und Methoden sondern ebenfalls auch die Ressourcen wie Grafiken, Javascripts und Stylesheets.
- Templateengine
    - Symfony bietet von Haus eine eingebaute Templateengine an, die sich “Twig” nennt.
    - Diese Templateengine hat sich ebenfalls die Drupal Community in ihr Framework integriert.[16]
- Routing
    - URLs werden durch die Konfiguration automatisch auf die richtigen PHP-Klassen geleitet.
    - Diese URLs können auch dynamisch generiert werden.
- Standartformate
    - Die Konfiguration von Symfony ist außerdem dazu fähig die Ausgabe von bestimmten URLs auf Formate wie z.B.: JSON festzulegen.

[16] vgl. http://2levelsabove.com/symfony-the-mother-of-all-php-frameworks/

- Reputation
    - Symfony ist ein stabiles Framework und international sehr bekannt. Es wird von vielen großen Webseiten und Unternehmen wie wetter.com, yahoo.com oder bbc.com verwendet.[17]
    - Des Weiteren ist die Community von Symfony sehr aktiv und wird ebenfalls von aktiven Entwicklern, Integratoren und Benutzern unterstützt.
    - An Laravel, Drupal und deren Community ist zu erkennen wie wertvoll Symfony tatsächlich ist, denn diese bedienen sich sehr vielen Bundles von Symfony wie z.B.: Twig, Validator, Translation, Process, Serializer oder YAML.[18]

2.5 Sonstige Grundlagen

2.5.1 Composer

Der Composer ist ein Programm, welches ein Abhängigkeiten-Management in PHP ist. Es erlaubt benötigte Bibliotheken, Plugins und Frameworks zu definieren, welche dann in das Projekt installiert werden. Diese Definition von Bibliotheken, Plugins und Frameworks geschieht in der Composer.json Datei.

[17] vgl. http://phpmagazin.de/Zend-Framework-2-vs.-Symfony-2.1-und-der-Kampf-um-die-Aufmerksamkeit-064623.html

[18] vgl. http://2levelsabove.com/symfony-the-mother-of-all-php-frameworks/

3 Kriterien für die Beurteilung der Programme

Nachfolgend werden Kriterien erläutert, die wichtig zur Beurteilung von PHPspec und Behat sind. Es wird untersucht, ob sich PHPspec und Behat für verhaltensgetriebene Entwicklung eignet, vor allem welchem Mehrwert es für Unternehmen stellt. Diese Kriterien decken folgende Bereiche ab:

- Codequalität
- Zeitersparnis
- Einrichtung
- Teamorientiertheit
- Testabdeckung
- Dokumenation
- Einarbeitung neuer Mitarbeiter
- Grenzen von PHPspec und Behat

3.1 Entwicklungsumgebung

3.1.1 Technische Voraussetzungen

Welche Technischen Voraussetzungen müssen PHPspec und Behat erfüllen um lauffähig zu sein und ihr volles Potenzial zu entfalten? Welche Voraussetzungen sind an den Server gestellt um alle Funktionalitäten der beiden Programme ausführen zu können?

3.1.2 Einrichtung

Die Einrichtung von PHPspec und Behat auf einem Server oder einer lokalen Entwicklungsumgebung ist unabdingbar. Deswegen muss überprüft werden wie Problematisch und Schwierig diese Einrichtung ist. Welche Probleme treten bei der Installation auf und wie kann man diese lösen.

3.2 Teamorientierte Programmierung

3.2.1 Einfluss des Kunden auf die Entwicklungsphase des Projektes

BDD hat den Vorteil, dass es sogenannte User-Stories verwendet. Diese werden von dem Kunden entwickelt und beschreiben die Kriterien die an die Software gestellt werden. Überprüft werden soll, wie dies in der Praxis aussieht und welche Kenntnisse der Kunden dafür benötigt. Wie groß ist der Einfluss des Kunden wirklich und inwieweit kann der Kunde die Entwicklung beeinflussen?

3.2.2 Zusammenarbeit von „Qualitäts-Sicherung“ zuständigen Mitarbeitern und Entwicklern

Die Zusammenarbeit im Team kann sehr unterschiedlich ausgeprägt sein. Es hängt von dem zu entwickelnden Projekt ab, aber auch von der Programmierungsart. Es soll überprüft werden, wie die Prozesse in einem Team durch Verhaltensgetriebener Entwicklung verändert werden und welchen Vorteil diese „neuen“ Bedingungen bieten.

3.2.3 Transparenz und Übersicht des Projekt-Prozesses und des Status innerhalb des Teams

Ein wichtiges Kriterium für die Entwicklung ist die Übersicht und die Transparenz. Diesen Fragen soll nachgegangen und geprüft werden, wie die Übersicht durch PHPspec verbessert wird und wie sich die Transparenz in dem gesamten Projekt-Prozesses verhält.

3.3 Dokumentation

BDD bietet den Vorteil, dass Dokumentationen als eine Art „Nebenprodukt“ entstehen. Inwieweit eignet sich diese Dokumentation als tatsächliche Dokumentation. Wie leicht verständlich, zugänglich, wie gut lesbar, funktional und korrekt ist die Dokumentation? Diese Fragen sollen mit diesem Kriterium beantwortet werden.

3.4 Einarbeitung neuer Mitarbeiter

In der Situation, dass ein neuer Mitarbeiter ein Projekt, was mit PHPspec oder Behat erstellt wurde, übernimmt bzw. ein Projekt erweitert. Wie leicht und einfach fällt es diesem Mitarbeiter daran zu entwickeln? Wie leicht ist es im Vergleich zu anderen Programmiermethoden, den Überblick darüber zu erhalten, was die Software wirklich leistet? Wie verhält es sich mit Entwicklern die zum ersten Mal in Kontakt mit BDD kommen?

3.5 Grenzen von PHPspec und Behat

PHPspec verarbeitet das interne Verhalten einer Funktionalität und Behat das externe Verhalten. Was sind die Begrenzungen dieser beiden Programme? Wozu sind sie fähig und wozu nicht?

3.6 Testabdeckung

Testabdeckung (oder engl.: Code Coverage) beschreibt das alle Funktionalitäten einer Software mit Tests abgedeckt sind. Wie funktioniert dies bei PHPspec und Behat. Was ist der Vorteil davon?

4 PHPspec

4.1 Grundlagen

Zwei BDD Programme ernten sehr viel Aufmerksamkeit in der PHP-Community - Behat und PHPspec. Behat beschreibt das externe Verhalten der Software mittels der lesbaren Gherkin Sprache. PHPspec beschreibt das interne Verhalten der Software, durch das Schreiben von kleinen „specs“.[19]

Entwickelt wurde PHPspec von dem leitenden Entwickler Marcello Duarte und seinem Partner Konstantin Kudryasho.[20] Bei PHPspec wird design orientiert entwickelt, indem das Verhalten der Software in einer sehr aussagekräftigen Weise beschrieben wird.

4.1.1 Warum reicht PHPunit nicht aus?[21]

PHPunit wurde 2006 von Sebastian Bergmann entwickelt. Es ist ein Framework zum Testen von PHP-Skripten. Mittels dieses Programmes können automatisierte Tests durchgeführt werden. Doch wieso stößt PHPunit bei BDD an seine Grenzen? Bei BDD wird das Verhalten von Programmcode formuliert. Dieser Programmcode soll nicht nur getestet werden, ob korrekte Ergebnisse zurückgegeben werden, sondern es geht um die Grundeinstellung, den Programmcode von den Anforderungen her zu modellieren.

---

19 vgl. http://welcometothebundle.com/phpunit-vs-phpspec-theory-on-behaviour-driven-development/

20 vgl. http://marcelloduarte.net/phpspec-2-0-is-out/

21 vgl. http://blog.mayflower.de/3873-PhpSpec.html

4.1.2 PHPspec ist intuitiv

Als Beispiel soll eine Klasse spezifiziert werden:

```
public function let($mailer)
{
    $mailer->beAMockOf('\Swift_Mailer');
    $this->beConstructedWith($mailer);
}
```

Abbildung 5 Einrichtung einer Mailer-Funktionalität für PHPspec

Wenn man nun die Funktion laut vorliest, wird man feststellen, dass dies intuitiv verständlich ist. Die Funktion erledigt genau das, was auch gelesen wird. Dies vereinfacht es enorm eine Spezifikation eines anderen Entwicklers zu lesen.

4.1.3 Funktionsweise von PHPspec

Die Entwicklungsart von BDD Tools legt fest dass, nur ein Bereich pro Spezifikation definiert werden darf.

Wie verhält sich PHPspec? Die Definition von PHPspec beschreibt ebenfalls nur eine einzelne Spezifikation pro Bereich.

Was ist eine Spezifikation? Es ist eine Liste von Beispielen, wie sich das Objekt verhalten soll.

Wie werden Objekte beschrieben?

```
$this->getTitle();
```

Abbildung 6 Beschreibung eines Objektes in PHPspec

Um das Verhalten von Objekten zu beschreiben werden sogenannte „Matcher“ verwendet.

```
$this->getTitle()->shouldReturn("myself");
```

Abbildung 7 Matcher in PHPspec

Das ist möglich, weil PHPspec die geschriebene Spezifikation in ein spezielles Objekt umwandelt - dieses Objekt wird als „Prophet“-Objekt bezeichnet.

Mittels diesem „Prophet“-Objekt ist es nun möglich, Methoden aufzurufen und Annahmen mit „Matchers“ zu definieren, welcher Wert erwartet wird. Zu dem jetzigen Zeitpunkt ist noch kein Programmcode implementiert. Aber wenn man dies ausführt wird kein Fehler zurückgegeben, sondern ein Vorschlag diese Klasse oder Methode generieren zu lassen.

Ebenfalls ist die nachfolgende Aussage möglich, da diese auch als „Prophet“-Objekt bereit gestellt wird:

```
$this->getPayment()->getType()->shouldReturn("visa");
```

Abbildung 8 weiteres Beispiel um ein Objekt zu beschreiben

Dies beschreibt zwar ein Objekt und wird über Annahmen überprüft, ob es selbst konsistent ist, aber Objekte kooperieren normalerweise miteinander.

```
public function payWith(CreditCardInterface $card)
{
    $card->withDraw($this->amount);
}
```

Abbildung 9 Beschreiben von einem Verhalten mit mehreren Objekten

Wie funktioniert dies?

Es existieren 2 Wege, um solch eine Situation anzugehen, eine klassische und eine nachgeahmte (mock) Herangehensweise. Die klassische Herangehensweise arbeitet mit realen bzw. nahezu realen Objekten. Dies bedeutet, dass jedes Mal wenn ein Collaborator gebraucht wird, reale Objekte instanziiert werden. Die „Mock“ Herangehensweise ahmt alle Collaborators nach. PHPspec folgt dem Prinzip: „Nur ein Objekt zu seiner Zeit“. Deswegen funktioniert die klassische Herangehensweise nicht und somit findet bei PHPspec das „Mocking“ statt.

4.1.4 Examples

Das Verhalten von Objekten ist in Beispielen zusammengefasst, den sogenannten „Examples“. Beispiele befinden sich in öffentlichen Methoden und beginnen mit „it_“ oder „its_“. PHPspec durchsucht die Spezifikationen nach diesen Methodenanfängen, um sie auszuführen. Aber wieso werden diese mit einem „Unterstrich“ benannt? Der Grund dafür ist:

weil_es_viel_besser_zu_lesen_ist

als

diesenSatzHierderSchwierigZuLesenIst.

4.1.5 Matchers

Matchers ähneln sehr den Assertions von xUnit, aber sie konzentrieren sich darauf dem Objekt zu sagen, wie es sich verhalten soll, anstatt zu prüfen wie es funktioniert. PHPspec enthält momentan 5 Vergleichsoperatoren, jedes von ihnen besitzt einen Alias damit sie leichter zu lesen sind.

Identität

- return, be, equal, beEqualTo
- ähnelt dem ===

Vergleich

- beLike
- ähnelt dem ==

Throw

- throw, during
- für das Überprüfen von Fehlermeldungen

Typ

- beAnInstanceOf, returnAnInstanceOf, haveType
- prüft den Objekttyp

Objektstatus

- have
- rufe die “is” Methode auf und gibt einen Wert zurück

4.1.6 Vorteile von PHPspec[22]

Der Fokus

Aufgrund des Spezifizierens des Verhaltens hat jede Zeile Code, die entwickelt wurde, einen messbaren Wert. Somit wird keine Kraft bzw. Zeit vergeudet und der Fokus bleibt stehts bei dem erwünschten Verhalten der Software.

Gutes Design

PHPspec veranlasst den Entwickler über ein gutes Design Gedanken zu machen.

Prophecy

Prophecy ist ein hilfreiches “Mocking”-Framework, welches PHPspec unterstüzt. Durch Prophecy wird das Verhalten von Klassen mithilfe von Collaborators beschrieben. Wenn nun ein Teil eines Programmcodes geprüft wird, das eine Verbindung zu einer externen Klasse besitzt, ist man aufgefordert, zu definieren, was die Erwartungen sind. Welche Methode soll aufgerufen werden? Welche Argumente sollen übergeben werden? Was wird als Rückgabewert erwartet? Durch diese Vorgehensweise erhält man eine gute Sicht darauf, was wirklich passiert.

Entwicklungsgeschwindigkeit

Durch PHPspec wird die Entwicklungsgeschwindigkeit erhöht mittels dem Zyklus der vorher schon beschrieben wurde. Des Weiteren beinhaltet PHPspec viele Codegeneratoren.

---

[22] vgl. http://jamiehannaford.com/code/7-reasons-why-you-should-use-phpspec/

Dokumentation

Die Resultate der Tests sind lesbar und dienen als Dokumentation.

4.2 Bewertung der Resultate anhand der Erstellung einer Funktionalität für einen Onlineshop

In diesem Kapitel werden die Ergebnisse präsentiert und ausgewertet welche die in Kapitel 3 gestellten Kriterien ergaben.

4.2.1 Entwicklungsumgebung

Technische Voraussetzungen

Geprüft wurde die verhaltensgetriebene Methode PHPspec mit der Version 2.1.* dev. Diese Version wird als Basis für die Voraussetzungen verwendet.

Wichtige Voraussetzungen für die Lauffähigkeit von PHPspec sind folgende:

- Eine PHP-Version ab einer Versionsnummer von 5.3 wird benötigt.
- Des Weiteren ist Prophecy in der Version 1.1 von Nöten sowie das gesamte PHP-Framework Symfony in Version 2.4.
- Um PHP-Skripte auszuführen muss Lokal ein Webserver vorinstalliert und konfiguriert sein. In diesem Fall wurde MAMP in der Version 3.0.5 verwendet.

Einrichtung

Als Erstes wird der lokale Webserver installiert. Dies geschieht durch die Installation von MAMP in der Version 3.0.5. In dem „web-ausführbaren“ Verzeichnis „htdocs“ wird der Composer hineinkopiert.

Die Einrichtung bzw. Installation der notwendigen Programme geschieht über den „Composer“. Hierzu ein Ausschnitt:

```
"require-dev": {
    "phpspec/phpspec":"2.0.*@dev"
},
"require": {
    "php": ">=5.3.3",
    "symfony/symfony": "~2.4",
    "doctrine/orm": "~2.2,>=2.2.3",
    "doctrine/doctrine-bundle": "~1.2",
    "twig/extensions": "~1.0",
    "symfony/assetic-bundle": "~2.3",
    "symfony/swiftmailer-bundle": "~2.3",
    "symfony/monolog-bundle": "~2.4",
    "sensio/distribution-bundle": "~2.3",
    "sensio/framework-extra-bundle": "~3.0",
    "sensio/generator-bundle": "~2.3",
    "incenteev/composer-parameter-handler": "~2.0"
},
```

Abbildung 10 Ausschnitt der Composer.json für PHPspec

Abbildung 11 Konsolen-Befehl für die Installation der Bibliotheken, Plugins, Frameworks

Hier zu sehen ist der Konsolen-Befehl um den Composer zu beauftragen, die definierten Programme zu installieren. Als erstes wird der Pfad zu PHP angegeben, um dann darauffolgend den Composer mitzuteilen, dass er den Befehl „install“ ausführen soll.

Fazit

Die Voraussetzungen für PHPspec sind sehr niedrig gehalten. Es werden zwar bestimmte Programme benötigt, allerdings sind diese Kostenlos und ohne großen Aufwand zu installieren. Jedoch gab es 2 Hürden bei der Einrichtung. Die Zusammenstellung der Composer-Konfiguration kann kompliziert sein. Durch die Webseite „Packagist“ lassen sich die notwendigen Konfigurationen für PHPspec herausfinden, aber dennoch harmonieren nicht alle Programme perfekt zusammen. Deswegen war es ein Geduldsspiel, herauszufinden, welche Versionsnummern der Programme funktionieren. Die andere Hürde war die korrekte Ausführbarkeit der Konsolen-Befehle zu garantieren. Das Problem hierbei war, den korrekten Pfad zu PHP herauszufinden und einzubinden. Allerdings ist dies eine Problematik, die beim erneuten Einrichten keine Hürde mehr darstellen sollte.

4.2.2 Teamorientierte Programmierung

Einfluss des Kunden auf die Entwicklungsphase des PHP-Projektes

Wie bereits erwähnt deckt PHPspec das interne Verhalten eines Programmes ab. In den sogenannten „Specs“, also den Dateien, wo die Tests formuliert sind. Dies ist für Kunden ohne Programmierkenntnisse schwer zu verstehen. Somit ist der Einfluss des Kunden nur gering. Nachfolgend wird der Kunde nur geringen Einfluss auf das PHP-Projekt haben, zumindest für das interne Verhalten des Programmes. Aber die Erfahrung zeigt, dass Kunden häufig nicht an das interne Verhalten interessiert sind bzw. es nicht verstehen könnten als Nicht-Entwickler.

Zusammenarbeit von „QS“ zuständigen Mitarbeitern und Programmierern

Durch die intelligente Benennung von Funktionen innerhalb des Specs ist es „QS“ zuständigen Mitarbeitern möglich, das Verhalten zu erahnen und nachfolgend dem Entwickler auf eventuelle Lücken beim Beschreiben des Verhaltens des Programmes hinzuweisen. Somit wird die Zusammenarbeit zwischen Abteilungen gestärkt und die Prüfungs- bzw. Testphase des Projektes kann schon während der Entwicklung beginnen.

Transparenz und Übersicht des Projekt-Prozesses und des Status innerhalb des Teams

Dadurch, dass Tests als erstes beschrieben werden, kann man die einzelnen Funktionalitäten als Meilensteine definieren. Diese Funktionalitäten werden Schrittweise abgearbeitet. Somit ist stets eine gute Übersicht des Projektes gewährleistet sowie auch die Transparenz.

Fazit

PHPspec bietet einen schlechten Zugang für Kunden, da Programmierkenntisse vorhanden sein müssen um „specs“ erstellen und lesen zu können. Aber die Zusammenarbeit zwischen „QS“ zuständigen Mitarbeitern und Entwicklern ist einfach und klar definiert.

4.2.3 Dokumentation

In PHPspec wird das beschriebene interne Verhalten einer Funktionalität in „specs“ gespeichert. Diese „specs“ enthalten also die Anforderungen an die Funktionalität und dienen somit als gute Dokumentation und Überblick der Funktionalität. Hierzu ein Ausschnitt:

```
function its_createAction_should_save_the_paymentObject_when_form_is_valid(Product $product, $request, $form, $formFactory, $entityManager, $repository)
{
    $entityManager->getRepository(Argument::exact('ShopBundle:Product'))->willReturn($repository);
    $repository->findAll()->willReturn($product);
    $repository->find(1)->willReturn($product);
    $repository->find(2)->willReturn($product);
    $repository->find(3)->willReturn($product);

    $classPayment = Argument::exact('BookStore\ShopBundle\Entity\Payment')->getValue();
    $payment = new $classPayment;
    $classPaymentForm = Argument::exact('Bookstore\ShopBundle\Form\PaymentForm')->getValue();
    $paymentForm = new $classPaymentForm;
    $classOrdered = Argument::exact('Bookstore\ShopBundle\Entity\Ordered')->getValue();
    $ordered = new $classOrdered;
    //$ordered->setPaymentId(Argument::any())->shouldBeCalled();

    $formFactory->create($paymentForm, $payment)->willReturn($form);
    $form->handleRequest($request)->willReturn($form);
    $form->isValid()->willReturn(true);

    $entityManager->persist($ordered)->shouldBeCalled();
    $entityManager->persist($payment)->shouldBeCalled();
    $entityManager->flush()->shouldBeCalled();

    $response = $this->createAction($request);
    $response->shouldBeAnInstanceOf('Symfony\Component\HttpFoundation\RedirectResponse');
}
```

Abbildung 12 Ausschnitt einer Spec-Datei

In diesem Ausschnitt einer Spec-Datei wird eine Funktion aufgerufen. Aufgrund des Funktionsnamens ist sehr leicht festzustellen, welches Verhalten damit beschrieben wird. Des Weiteren kann der Entwickler feststellen, welche Parameter als Voraussetzung vorhanden sein müssen, um dieses Verhalten korrekt durchzuführen. Schließlich wird auch das Resultat des Verhaltens festgehalten. Diese Argumente sprechen eindeutig dafür warum PHPspec auch als gute Dokumentation geeignet ist.

4.2.4 Einarbeitung neuer Mitarbeiter

Ein großes Problem bei der Einarbeitung neuer Mitarbeiter, die zum ersten mal mit BDD in Kontakt kommen, ist, dass sie bereits ein Verständnis über Programmierung und einer Herangehensweise mitbringen. Es ist also wichtig, dass der Entwickler sein Verständnis von Design und Programmierfähigkeiten überdenkt. Verschiedene Studien, wie zum Beispiel der Universität Auckland belegen[23], dass Entwickler in verschiedenen Formen Widerstand zeigten und ihren ursprünglichen Arbeitsablauf wieder

[23] vgl. Rick Mugridge – Challenges in teaching test driven development, Seite 410-413, Springer 2003

annahmen. Tests wurden erst nach der Implementierung geschrieben, anstatt sie vorher zu schreiben, so wie es BDD fordert. Deswegen ist es erforderlich von der „test-last“ Herangehensweise die Entwickler zur „test-first“ Herangehensweise zu bewegen.

Um dies zu verwirklichen arbeiteten Universitäten, wie die aus Auckland, verschiedene Konzepte aus[24]. Es wurde versucht, den Entwicklern das Refaktorisieren durch einfache Aufgaben zugänglicher zu machen. Sie wurden geschult im Umgang mit Testing-Frameworks.

Wenn diese Hürde geschafft ist, ist die Einarbeitung in ein schon vorhandenes Projekt in PHPspec recht einfach. Wie unter dem Punkt „Dokumentation“ besprochen, dient es als sehr guter Überblick über die vorhanden Funktionalitäten der Software und ist somit ein sehr guter Einstieg in das Projekt.

4.2.5 Grenzen von PHPspec

Nicht-funktionale Aspekte, wie zum Beispiel Layout-Defekte oder mangelnde Gebrauchstauglichkeit, lassen sich nicht automatisiert überprüfen. Dies ist technisch nur hinreichend möglich und hat mit PHPspec nichts zu tun, denn PHPspec beschreibt das interne Verhalten von einer Software. Genau da liegt auch die Grenze von PHPspec, da es nur für Klassen, Methoden und Funktionen zuständig ist. Mittels PHPspec ist es nur möglich diese Aspekte zu überprüfen.

4.2.6 Testabdeckung

Da verhaltensgetrieben entwickelt wird, wird somit auch vor der Implementierung das gewünschte Verhalten der Software in Tests beschrieben. Somit erreicht PHPspec als Nebenprodukt 100% Testabdeckung über das interne Verhalten der Software. Der Vorteil davon ist, dass – wenn alle Tests erfolgreich waren – man auch wirklich davon ausgehen kann, dass alles funktioniert.

---

[24] vgl. Rick Mugridge – Challenges in teaching test driven development, Seite 410-413, Springer 2003

5 Behat

5.1 Grundlagen

Behat wurde ursprünglich als Cucumber ähnliches Framework für PHP entwickelt. Anfangs war die Überlegung, Behat als Third-Party Programm für PHPunit zu entwickeln.[25] Allerdings geriet man mit PHPunit schnell an die Grenzen des Möglichen. Deswegen entschied man sich Behat als ein eigenständiges Framework für BDD zu entwickeln. Es ist das erste BDD Programm für PHP Anwendungen und wurde von Konstantin Kudryasho ins Leben gerufen.[26]

5.1.1 Warum reicht PHPunit nicht aus?

PHPunit deckt interne Funktionalitäten ab. Aber die externen Funktionalitäten, die dem Benutzer der Applikation zuerst betreffen, werden häufig per Hand geprüft. Dagegen ist nichts einzuwenden, aber nachdem die Applikation erweitert und geändert wurde, findet meist kein erneuter Test der externen Funktionalitäten statt. Unternehmen sind meist nicht bereit, erneut für Tester zu zahlen, da es beim Erscheinen der Applikation schon funktioniert hat. Genau um dieses Problem zu beheben tritt Behat in Erscheinung und bietet eine automatisierte Testabdeckung der externen Funktionalitäten an.

5.1.2 Was ist nun Behat?

Mittels Behat ist es möglich lesbare „Stories“ in der Gherkin-Sprache zu programmieren, die als Tests für die Applikation dienen. Es kann für „API“ und „Funktionalitäten“ testen verwendet werden.

---

[25] vgl. http://www.methodsandtools.com/tools/behat.php

[26] vgl. http://about.me/everzet

5.1.3 Die Verzeichnisstruktur von Behat[27]

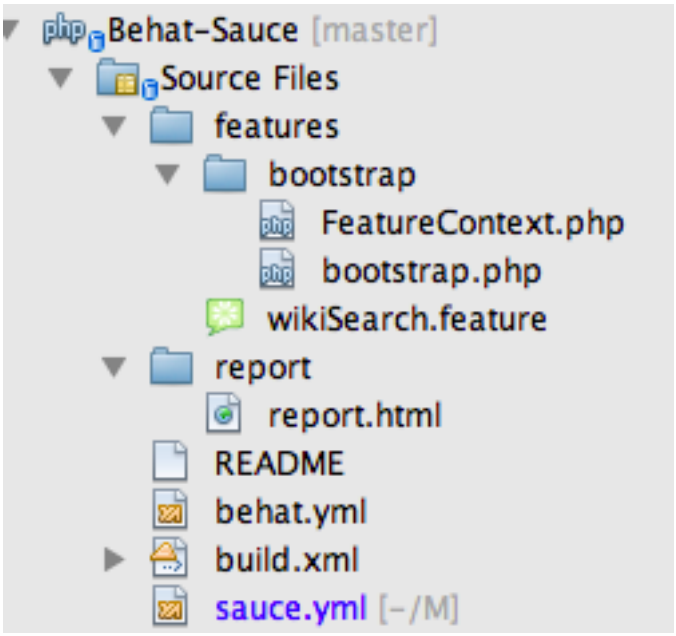

Abbildung 13 Verzeichnisstruktur eines Behat-Projektes[28]

features-Ordner

In diesem Ordner befinden sich alle .feature Dateien. Diese enthalten die Szenarios bzw. das Verhalten der Applikation, die getestet werden sollen. In dem Ordner “bootstrap” befindet sich die FeatureContext.php, welche Verlinkungen zu Third-Party-Libraries beinhaltet sowie Funktionalitäten, die von Szenarien aufgerufen werden können.

behat.yml

Das ist die Standard-Konfigurationsdatei von Behat, welches benutzt wird, um Features auszuführen. Des Weiteren wird darin die Webdriver-Engine festgelegt sowie Profile und Pfade zu der FeatureContext.php.

---

27 vgl. http://sauceio.com/index.php/2012/01/adding-sauce-to-behat/

28 vgl. http://sauceio.com/wp-content/uploads/2012/01/Screen-Shot-2012-03-13-at-7.20.11-PM.png

5.1.4 Gherkin[29]

Gherkin ist eine Sprache und ein „Parser“. Gherkin wurde eingesetzt für Cucumber, um Features zu spezifizieren. Diese Sprache beschreibt die Struktur von einem Feature. Diese Struktur enthält:

- Titel
- Story
- Szenario
- Steps

Titel

Der Titel eines Features beschreibt das Verhalten aus der Sicht des Stakeholders.

Beispiel: Member rents Video

Durch diese Bezeichnung des Titels ist offensichtlich, dass ohne diesen Programmcode, diese Funktionalität nicht enthalten ist. Wenn ein allgemeiner Begriff wie „Rent-Management“ gewählt wird, ist nicht mehr eindeutig, welches Verhalten darin beschrieben wird.

Story

Eine Story hat 3 Komponenten: einen Ausführer, ein Verhalten und einen Grund (welcher den Wert beschreibt für den Stakeholder).

Beispiel: As a video-club member
I want to rent a video
So that i can take it away with me and watch it at home

Dies beschreibt klar, was für Funktionalitäten gefordert sind und hilft dem Team weitere Überlegungen zu treffen.

Szenario

---

[29] vgl. http://docs.behat.org/guides/1.gherkin.html

Jedes Szenario besteht aus einer Liste von Steps. Die Steps starten mit Schlüsselwörtern wie „Given“, „When“, „Then“, „But“ und „And“. Diese Schlüsselwörter spielen keine Rolle bei der Funktionalität. D.h. theoretisch ist es egal welches Schlüsselwort verwendet wird, aber um das Verhalten zu beschreiben bzw. im nachhinein zu erkennen wie sich die Funktionalität verhält, ist es wichtig die Schlüsselwörter sinnvoll auszuwählen.

Beispiel[30]:

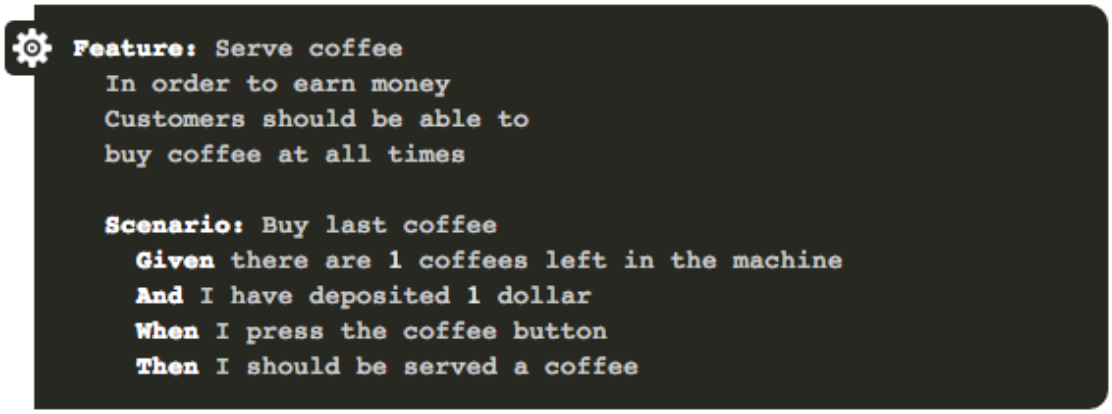

Abbildung 14 Beispiel eines Szenarios

Das Szenario ist ein Kernelement der Gherkin Struktur. Jedes Szenario beginnt mit dem Schlüsselwort „Scenario“ und einem Titel. Jedes Feature kann mehrere Szenarios enthalten.

Steps

Features bestehen aus Steps, also aus „Given“, „When“, „Then“, „And“ und „But“.

- Given
    - Der Sinn hinter dem Schlüsselwort “Given” ist, dem System einen Status zu geben.
    - Beispiel:
        - Given there are no users on site
        - Given the database is clean
        - Given I am logged in as “Lukas Vorberg”

[30] vgl. http://docs.behat.org/guides/1.gherkin.html

- When
    - „When“ wird verwendet um eine Useraktion zu beschreiben.
    - Beispiel:
        - When I am on “/some/page”
        - When I fill “username” with “Lukas”
        - When I fill “password” with “123456”
        - When I press button “login”
- Then
    - „Then“ beschreibt das Resultat der Useraktion. Dieses Resultat sollte der Story-Beschreibung ähneln.
    - Beispiel:
        - When I call “echo Hello”
        - Then the output should be “Hello”
    - Das Schlüsselwort „Then“ beschreibt nur Ausgaben, die dem Benutzer ebenfalls sichtbar sind. Datenbank-Dateien werden damit nicht überprüft.
- And, But
    - Wenn mehrere „Given“-, „When“- oder „Then“-Steps existieren:
        - Given one thing
        - Given another thing
        - Given yet another thing
        - When I open my eyes
        - Then I see something
        - Then I don’t see something else
    - Diese Schreibweise ist korrekt, behindert jedoch den Lesefluss. Daher wird “And” und “But” verwendet:
        - Given one thing
        - And another thing
        - And yet another thing
        - When I open my eyes

- Then I should see something
- But I don’t see something else

5.1.5 FeatureContext.php

Die FeatureContext.php beherbergt Klassen und Methoden, die letztendlich die Steps ausführen. Jeder Step, dessen Anfang mit dem Schlüsselwort „Given“, „When“, „Then“, „And“ oder „But“ definiert wurde, wird in der FeatureContext.php gesucht. Die Methoden in der FeatureContext.php besitzen sogenannte „Assertations“. Diese stellen die Beziehung zwischen Step und der dazugehörigen Methode her.

```
<?php
/**
* @Given /^I am in a directory "([^"]*)"$/
*/
public function iAmInADirectory($dir)
{
    if (!file_exists($dir)) {
        mkdir($dir);
    }
    if (!is_dir($dir)) {
      throw new Exception('Directory not found');
    }
    chdir($dir);
}
?>
```

Abbildung 15 Auszug einer FeatureContext.php

Die Methode iAmInADirectory enthält die Assertation: I am in a directory „([^“]*)“
Diese Assertation definiert den Step, den es benötigt, damit diese Methode aufgerufen werden kann. Der Platzhalter „([^“]*)“ beschreibt, dass der Wert, der an dieser Stelle im Step steht, als Variable für die Funktion verwendet wird. Dadurch ist es möglich, eine Methode für mehrere Steps zu definieren.

5.1.6 Mink[31]

Der Browser spielt eine große Rolle im Web-Bereich. Der Browser ist das Fenster durch welchen der Benutzer mit einer Webseite interagieren kann. Um nun Web-Applikationen korrekt zu testen, muss die Interaktion zwischen Browser, Webseite und dem Benutzer geprüft werden. Mittels der Extension Mink ist dies möglich. Mink benutzt hierfür sogenannte Browser-Emulatoren. Es existieren 2 komplett verschiedene Typen von Emulatoren. Die Headless Browser Emulatoren sowie die Browser Controller.

“Headless-Browser” emulieren einen realen HTTP-Request gegen eine Applikation und analysieren den Inhalt der Antwort. Diese Methodik ist einfach auszuführen und zu konfigurieren, weil der Emulator in jeder beliebigen Programmsprache erstellt werden kann und per Konsole auf dem Server ohne grafische Benutzeroberfläche lauffähig ist. Eine bekannte Engine für den Headless-Browser ist GoutteDriver. Vorteile dieser Emulatoren ist die Einfachheit, Schnelligkeit sowie die Lauffähigkeit ohne einen realen Browser. Nachteilig dabei ist allerdings, dass keine Unterstützung für JavaScript und Ajax vorhanden ist.

“Browser-Controller” zielen darauf ab, einen realen Browser zu steuern. Dieser Emulator simuliert Benutzerinteraktionen in einem Browser und fragt die Informationen des aktuellen Browserfensters ab. Bekannte Engines hierfür sind: SahiDriver, ZombieDriver, SeleniumDriver oder auch Selenium2Driver. Vorteilig ist die Unterstützung von JavaScript und Ajax. Nachteile sind u.a., dass es einen vorinstallierten Browser benötigt und es gegenüber dem Headless-Browser wesentlich langsamer ist.

Jeder Emulator-Typ hat seine Vorzüge. Im realen Gebrauch ist die Nutzung beider Typen wichtig.

---

[31] vgl. http://mink.behat.org/#understanding-the-mink

5.2 Bewertung der Resultate anhand der Erstellung einer Funktionalität für einen Onlineshop

In diesem Kapitel werden die Ergebnisse präsentiert und ausgewertet welche die in Kapitel 3 gestellten Kriterien ergaben.

5.2.1 Entwicklungsumgebung

Technische Voraussetzungen

Behat wird mit der Version 3.0.x-dev getestet und geprüft. Diese Version wird als Basis für die nachfolgenden Überprüfungen verwendet.

Voraussetzungen für die Lauffähigkeit von Behat sind folgende:

- Eine PHP Version mit der Versionnummer 5.3.3 oder höher wird benötigt.
- Das Framework Symfony mit der Version 2.4 muss vorhanden sein.
- Desweiteren ist es wichtig das Gherkin 4.3, Mink 1.6.x-dev sowie die verschiedenen Browser-Engines vorkonfiguriert sind. Dies wären folgende: Selenium 1.2.x-dev, Goutte 1.1.x-dev.

Einrichtung

Wie bei PHPspec schon berichtet, ist ein lokaler Webserver oder Server notwendig. Hierbei wurde ebenfalls MAMP in der Version 3.0.5 verwendet. Um Behat und die nötigen Funktionalitäten zu installieren wird ebenfalls ein Composer verwendet, der sich in dem „web-ausführbaren“ Verzeichnis „htdocs“ befindet. Nachfolgend ein Ausschnitt der Composer.json Datei:

```
"require-dev": {
    "phpspec/phpspec":"2.0.*@dev",
    "behat/behat": "~2.5",
    "behat/mink": "~1.5",
    "behat/mink-extension": "~1.2",
    "behat/mink-goutte-driver": "~1.0",
    "behat/mink-selenium2-driver": "*",
    "behat/symfony2-extension": "~1.1"
},
"require": {
    "php": ">=5.3.3",
    "symfony/symfony": "~2.4",
```

Abbildung 16 Ausschnitt der Composer.json für Behat

Um die Konfiguration der Composer.json auszuführen und zu installieren wird dieser Konsolenbefehl ausgeführt:

```
lukasvorberg$ /Applications/MAMP/bin/php/php5.5.10/bin/php composer.phar install
```

Abbildung 17 Konsolen-Befehl für die Installation der Bibliotheken, Plug-Ins, Frameworks

Fazit

Um Behat zu benutzen, müssen nur wenige Voraussetzungen erfüllt werden bzw. der Zeitaufwand der Einrichtung ist sehr gering gehalten. Wie auch schon bei PHPspec, hat auch Behat mit den folgenden zwei Herausforderungen zu kämpfen: die richtige Zusammenstellung der Konfigurationen in der Composer.json mit den korrekten Versionsnummern kann eine gewisse Zeit beanspruchen sowie die Ausführbarkeit der Konsolenbefehle funktionsfähig zu machen.

5.2.2 Teamorientierte Programmierung

Einfluss des Kunden auf die Entwicklungsphase des PHP-Projektes

Behat verwendet den Parser Gherkin. Wie bereits in den Grundlagen erwähnt, besteht es aus Szenarien und die wiederum aus Titel und Steps. Gherkin wurde speziell dafür entwickelt, dass Personen mit wenig bis zu gar keinen Programmierkenntnissen lesen und verstehen können.

Dies bietet den Vorteil, dass der Kunde beim Beschreiben des Verhaltens der Software mit involviert sein kann. Somit kann der Kunde nicht nur Kriterien definieren, sondern kann auch ausführlich und detailliert beschreiben, wie sich das gewünschte Produkt

verhalten soll. Deswegen sind Missverständnisse bei der Absprache zwischen Entwickler und Kunde sehr gering.

Erforderlich dafür sind allerdings Englisch-Grundkenntnisse vom Kunden, da Gherkin Englisch als Sprache verwendet.

Der Kunde legt dabei den Fokus mehr auf die Funktionsfähigkeit des Produktes nach seinen vorher definierten Vorstellungen, als and die interne Architektur des Programmes.

Zusammenfassend gesehen ist Behat eine sehr gute Möglichkeit, eine kundennahe Entwicklung zu gewährleisten. Der Kunde kann am Verhalten der Software aktiv mitarbeiten, was letztendlich zu einem besseren und valideren Endpunkt führt.

Zusammenarbeit von „QS“ zuständigen Mitarbeitern und Programmierern

Die Zusammenarbeit zwischen den „QS“ Mitarbeiter die für die Qualität zuständig sind und den Entwickeln geschieht ähnlich wie mit einem Kunden. „QS“ zuständige Mitarbeiter können sich ganz auf das Beschreiben des Verhaltens konzentrieren, während die Entwickler Schritt für Schritt dieses beschriebene Verhalten entwickeln. Die Aufgabenbereiche sind also ganz klar getrennt. Zu welchen Zeitpunkt die Mitarbeiter in das Projekt eingreifen können ist auch ganz klar definiert. Sobald das Verhalten der Software in Gherkin von den „QS“ zuständigen Mitarbeitern bzw. in Zusammenarbeit mit dem Kunden definiert wurde, können die Entwickler die Entwicklungsphase einläuten.

Transparenz und Übersicht des Projekt-Prozesses und des Status innerhalb des Teams

Der Projektfortschritt lässt sich hervorragend anhand der Szenarien überprüfen. Wie bereits erwähnt, wird das Verhalten Schritt für Schritt entwickelt. D.h. um herauszufinden wie weit der Fortschritt ist, führt man Behat aus. Anschließend wird in der Konsole in Echtzeit dokumentiert, welche Szenarien geprüft werden und wo es zu Fehlern kam. Somit wird durch die fehlgeschlagenen Tests erkennbar, was noch entwickelt werden muss. Anhand dieser Möglichkeit ist jederzeit ein Überblick über den Entwicklungsstand möglich.

Fazit

Behat bietet dem Kunden eine sehr gute Möglichkeit in die frühe Entwicklungsphase mit teilzunehmen. Des Weiteren ermöglicht es eine gute Zusammenarbeit zwischen „QS“ zuständigen Mitarbeitern und den Entwicklern. Voraussetzung sind dabei grundlegende Englischkenntnisse sowie eine kurze Einführungsphase in den Sprachge-

brauch von Gherkin. Dies stellt aber eine nicht allzu große Herausforderung dar, da Englisch in der Geschäftswelt weitestgehend gefordert ist.

5.2.3 Dokumentation

Wenn ein neuer Entwickler ein vorhandenes Projekt übernimmt bedarf es immer an gewisser Einarbeitungszeit. Genauso verhält es sich auch, wenn ein älteres Projekt wieder aufgenommen wird. Aber durch die Dokumentation, was im Grunde genommen die Szenarien sind, verkürzt sich die Einarbeitungszeit sehr stark - zumindest was das Erfassen des Verhaltens der Software angeht. Wie im nachfolgenden Ausschnitt einer Szenario-Datei zu entnehmen, ist dies gut gegliedert. Durch die benutzte Syntax ist dies im „eigentlichen“ Sinne bereits eine Dokumentation die fasst problemlos als Solche in den Akten mit abgeheftet werden könnte.

Beispiel[32]:

```
Feature: Serve coffee
  In order to earn money
  Customers should be able to
  buy coffee at all times

  Scenario: Buy last coffee
    Given there are 1 coffees left in the machine
    And I have deposited 1 dollar
    When I press the coffee button
    Then I should be served a coffee
```

Abbildung 18 Beispiel eines Szenarios

5.2.4 Einarbeitung neuer Mitarbeiter

Der Punkt 4.2.4 behandelt die Einarbeitung neuer Mitarbeiter in PHPspec. Fast identisch verhält es sich auch in Behat. Natürlich unterscheidet sich die Technik, wie entwickelt wird, aber in Behat existieren genau dieselben Problematiken wie in PHPspec. Es muss ein Umdenken in der Art und Weise wie entwickelt wird stattfinden. Behat ist ebenfalls eine „test-first“-Herangehensweise, was jedoch außerhalb der verhaltens- und testgetriebenen Entwicklung unüblich ist. Deswegen ist auch hier zu empfehlen die

[32] vgl. http://docs.behat.org/guides/1.gherkin.html

Konzepte der Universitäten, wie die aus Auckland, anzuwenden. Sie versuchen mit einfachen und aufgeteilten Aufgaben die Mitarbeiter an die neue Methode heranzuführen und ihnen Sicherheit im Umgang damit zu vermitteln.

5.2.5 Grenzen von Behat

Behat testet vorwiegend das externe Verhalten einer Software. Mit Behat kann die komplette Webseite überprüft werden - also alles was einem normalen Nutzer ebenfalls möglich wäre. Des Weiteren ist es möglich, dass Behat auch jQuery und Ajax verarbeiten kann. Somit sind selbst komplexe Nutzer-Interkationen mit der Webseite prüfbar. Ebenfalls kann Behat mit Datenbanken interagieren, um bestimmte Situationen zu erzeugen um dann ein Verhalten abzuprüfen. Aber Behat hat auch seine Grenzen. Wie erwähnt ist es nicht möglich Klassen, Methoden und interne Funktionalitäten zu überprüfen. Nur deren Funktionsweise, die als Ausgabe auf der Webseite sichtbar wird.

5.2.6 Testabdeckung

Mittels Behat wird ein Verhalten beschrieben und nicht einfach ein Stück Programmcode überprüft. Aufgrund seiner Natur wird nur Programmcode entwickelt und implementiert, welcher vorher in einem Szenario beschrieben wurde. Dementsprechend wird es keinen überflüssigen Programmcode geben und automatisch findet eine 100% Testabdeckung des externen Verhaltens statt.

6 Fazit

Das erklärte Ziel dieser Bachelorarbeit ist es „zu überprüfen wie verhaltensgetriebene Entwicklung mit PHPspec und Behat funktioniert und ob diese Art Softwarelösungen zu entwickeln heutzutage seine Berechtigung hat“. Dieses Ziel konnte erreicht werden.

In Kapitel 2 „Grundlagen“ wurde ein Fundament geschaffen in der die allgemeine Funktionsweise von BDD präzise erklärt wird, vor allem auch die Unterschiede zu TDD.

In Kapitel 3 „Kritierien“ wurden die Kriterien festgelegt nach welchen PHPspec und Behat sowie generell BDD bewertet wurden.

In Kapitel 4 „PHPspec“ wurden die Grundlagen von PHPspec beschrieben. Es wurde so detailliert darauf eingegangen, damit es als gute Einleitung in diese Thematik dienen kann und ebenfalls auch als Einstieg in die Programmierung mit PHPspec benutzt werden kann. Des Weiteren wurden wichtige Kriterien an dieses Programm überprüft und bewertet. Die gewonnenen Erkenntnisse zeigen deutlich, dass PHPspec ein wertvolles Programm für Unternehmen sein kann. Gerade in einem Team hilft PHPspec enger zusammen zu arbeiten und zeitsparender, schneller und qualitativ hochwertiger zu entwickeln.

In dem Kapitel 5 wurde das Programm „Behat“ unter die Lupe genommen. Als erstes wurde die Funktionsweise erläutert sowie die Vorteile von Behat. Die zuvor gestellten Kriterien wurden geprüft und ausgewertet. Die Ergebnisse zeigen, dass Behat eine gute Schnittstelle zwischen Entwickler und Kunde sein kann. Durch Behat wird anhand der benutzerfreundlichen Sprache Gherkin dem Kunden ermöglicht, dass dieser über den gesamten Entwicklungsprozess beteiligt sein kann. Um das externe Verhalten einer Software zu beschreiben eignet sich Behat hervorragend.

Zusammenfassend muss man sagen, dass durch PHPspec und Behat die Softwarequalität deutlich erhöht wird. Das schließt sauberen Programmcode, gute Erweiterbarkeit und einfache Einarbeitung in ein vorhandenes Projekt mit ein. Diese beiden Programme könnten gemeinsam unter dem Begriff „BDD“ kategorisiert werden. Allerdings arbeiteten diese beiden auf unterschiedlichen Ebenen. Behat kann das externe Nutzerverhalten beschreiben und PHPspec das interne Verhalten der Klassen und Methoden. Um das größtmögliche Potenzial zu gewinnen, müssen beide Programme für das Entwickeln von Softwarelösungen verwendet werden.

7 Ausblick

Mit dieser Arbeit wurde eine Einführung in „verhaltensgetriebene Entwicklung“ erarbeitet. Diese Einführung kann als Muster für Entwickler und Unternehmen dienen, die sich mit dieser Thematik auseinander setzen müssen.

Während der Bearbeitung der Kriterien kamen folgende Probleme zum Vorschein, bei welchen es seitens der Entwickler der Programme Verbesserungen geben sollte. Der Selenium 2 Driver für das Programm Behat ist recht langsam, gerade wenn jQuery und Ajax überprüft und ausgeführt werden. Selenium 2 ist zwar ein Alleskönner, aber durch die geringe Geschwindigkeit wird ebenfalls die Arbeitsgeschwindigkeit des Programmierers geringer. In diesem Punkt muss Behat bzw. deren Community dringend nachbessern bzw. neue Driver bereitstellen. Ein weiteres Problem ist die nicht ausführliche Dokumentation von PHPspec. Dieses Programm ist relativ neu, aber momentan ist es doch sehr aufwändig sich in die Funktionsweise von PHPspec einzulesen und zu entwickeln. Durch die steigende Popularität von PHPspec wird dies in den nächsten 6 Monaten auch gelöst sein.

Diese Bachelorarbeit beschreibt das Entwickeln in der Entwicklungsumgebung von Symfony. Es existieren schon diverse Funktionalitäten, um dies auf CMS-Systeme zu erweitern. Vor allem Drupal und deren Community ist sehr aktiv darin. Es wäre für die Zukunft wünschenswert, wenn man für die Plattformen Typo3, Joomla und Wordpress diese Programme ebenfalls nutzen könnte.

Wie den Kriterien zu entnehmen ist, wurden auch unternehmerisch typische Situationen geprüft. Dies bietet den Vorteil, dass die gewonnen Erkenntnisse auch für Unternehmen von Interesse sein können. Gerade wenn das Unternehmen sich mit „verhaltensgetriebener Entwicklung“ auseinander setzen möchte um den Mehrwert abzuschätzen.

Literaturverzeichnis

Ayuso, G. (kein Datum). *Methods and Tools*. Abgerufen am 14. Juli 2014 von Behat: http://www.methodsandtools.com/tools/behat.php

Beck, K. (2012). *Test-Driven Development by Example.*Boston: Seventeenth printing.

Behat. (kein Datum). Abgerufen am 14. Juli 2014 von Gherkin: http://docs.behat.org/guides/1.gherkin.html

Behat. (kein Datum). Abgerufen am 14. Juli 2014 von Understanding The Mink: http://mink.behat.org/#understanding-the-mink

David Chelimsky, D. A. (2010). *The RSpec Book - Behaviour-Driven Development with RSpec, Cucumber, and Friends.*

Donato, G. D. (05. Januar 2013). *Welcome To The Bundle*. Abgerufen am 14. Juli 2014 von PHPunit vs PHPspec - Theory on Behaviour Driven Development: http://welcometothebundle.com/phpunit-vs-phpspec-theory-on-behaviour-driven-development/

Duarte, M. (19. März 2014). *Marcello Duarte*. Abgerufen am 14. Juli 2014 von PHPspec 2.0 is out: http://marcelloduarte.net/phpspec-2-0-is-out/

Hannaford, J. (08. April 2014). *Jamie Hannaford*. Abgerufen am 14. Juli 2014 von 7 Reasons why you should use PHPspec: http://jamiehannaford.com/code/7-reasons-why-you-should-use-phpspec/

Herrmann, E. (07. September 2012). *PHP Magazin*. Abgerufen am 14. Juli 2014 von Zend Framework 2 vs. Symfony 2.1 und der Kampf um die Aufmerksamkeit: http://phpmagazin.de/Zend-Framework-2-vs.-Symfony-2.1-und-der-Kampf-um-die-Aufmerksamkeit-064623.html

it-agile. (kein Datum). Abgerufen am 14. Juli 2014 von Testgetriebene Entwicklung TDD: http://www.it-agile.de/wissen/praktiken/agiles-testen/testgetriebene-entwicklung-tdd/

Khalili, M. (15. Februar 2011). *Mehdi-Khalili*. Abgerufen am 14. Juli 2014 von BDD to the rescue: http://www.mehdi-khalili.com/bdd-to-the-rescue

Kiss, F. (21. November 2012). *ymc*. Abgerufen am 14. Juli 2014 von Behavior Driven Development with Behat und co - Mehr als nur testen: http://www.ymc.ch/behavior-driven-development-with-behat-co-mehr-als-nur-testen

Kudryashov, K. (kein Datum). *About Me*. Abgerufen am 14. Juli 2014 von Everzet: http://about.me/everzet

Meier, R. (01. Mai 2010). *Daraff*. Abgerufen am 14. Juli 2014 von Einführung Test Driven Development: http://daraff.ch/2010/01/einfuhrung-test-driven-development/

Mugridge, R. (2003). *Challenges in teaching test driven development.* Springer.

Neely, J. (01. Januar 2009). *Stackoverflow*. Abgerufen am 14. Juli 2014 von How do we define code quality: http://stackoverflow.com/questions/405243/how-do-we-define-code-quality

North, D. (kein Datum). *dannorth*. Abgerufen am 14. Juli 2014 von Introducing BDD: http://dannorth.net/introducing-bdd/

Saud. (kein Datum). *2 Levels Above*. Abgerufen am 14. Juli 2014 von Symfony - The Mother Of All PHP Frameworks: http://2levelsabove.com/symfony-the-mother-of-all-php-frameworks/

Sebastian Bergmann, S. P. (2013). *Softwarequalität in PHP Projekten.* München: Carl Hanser Verlag.

Symfony. (kein Datum). Abgerufen am 14. Juli 2014 von Why use a Framework: http://symfony.com/why-use-a-framework

Trieba, M. (03. Juni 2013). *Mayflower*. Abgerufen am 14. Juli 2014 von PHPspec: http://blog.mayflower.de/3873-PhpSpec.html

Wikipedia. (kein Datum). Abgerufen am 14. Juli 2014 von Sapir Whorf Hypothese: http://de.wikipedia.org/wiki/Sapir-Whorf-Hypothese

Wikipedia. (kein Datum). Abgerufen am 14. Juli 2014 von Codequalität: http://de.wikipedia.org/wiki/Codequalit%C3%A4t

Wikipedia. (kein Datum). Abgerufen am 14. Juli 2014 von Symfony: http://de.wikipedia.org/wiki/Symfony

Eigenständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und nur unter Verwendung der angegebenen Literatur und Hilfsmittel angefertigt habe. Stellen, die wörtlich oder sinngemäß aus Quellen entnommen wurden, sind als solche kenntlich gemacht. Diese Arbeit wurde in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegt.

Ort, Datum Vorname Nachname