Tests und testgetriebene Entwicklung

Hausarbeit

Arbeitsbereich Wissenschaftliches Rechnen
Fachbereich Informatik
Fakultät für Mathematik, Informatik und Naturwissenschaften
Universität Hamburg

Vorgelegt von:	Antonia Bücklers
E-Mail-Adresse:	3bueckle@informatik.uni-hamburg.de.de
Matrikelnummer:	6524676
Studiengang:	BSc. Informatik

Hamburg, den 02.09.2015

Abstract

Testen ist ein wichtiges Tool zur Qualitätssicherung in der Softwareentwicklung. Seit einiger Zeit wird immer häufiger von testgetriebener Entwicklung Gebrauch gemacht, welches ein Vorgehen ist, bei dem die Softwaretests vor der zu programmierenden Software geschrieben werden. Durch dieses Vorgehen wird sichergestellt, dass die Software nur die Anforderungen des Kunden erfüllt. Es wird noch viel Kritik an dieser relativ neuen Vorgehensweise geäußert, hauptsächlich, da die wenige Erfahrung der Programmierer mit dieser Technik häufig zu einem erhöhten Zeitaufwand bei der Programmierung der benötigten Software führt, und die Einarbeitung auch einer gewissen Zeit benötigt. Es wurden schon einige Experimente zwischen Gruppen, die traditionelle Softwareentwicklung betreiben und welchen die testgetriebene Entwicklung betreiben durchgeführt, um mögliche Vorteile der testgetriebenen Entwicklung herauszufinden. Es wurden drei Studien ausgewertet und das Fazit dieser Studien ist, dass testgetriebene Entwicklung sehr effizient sein kann, da durch dieses Verfahren weniger Defects auftauchen als bei der traditionellen Programmierung. Außerdem haben die Fall Studien gezeigt, dass testgetriebene Entwicklung keine ausschlaggebenden Auswirkungen auf die Produktivität der Entwickler hat. Dennoch liefert eine testgetriebene Entwicklung keine Garantie für eine Fehlerfreiheit des Systems, deshalb sollten zusätzliche Tests für die Sicherung der Qualität durchgeführt werden.

Inhaltsverzeichnis

Kapitel 1

Einleitung

Softwaretests gehören zu jedem gute Programmcode. Durch sie wird die Software auf die Erfüllung der vorher festgelegten spezifischen Anforderungen geprüft und bewertet. Softwaretests messen also die Qualität der zu testenden Software. Es gibt viele verschiedene Softwaretests, diese können nach verschiedenen Kriterien klassifiziert werden. Sie können beispielsweise nach ihrer Prüftechnik oder nach dem Testkriterium klassifiziert werden.

Es gibt verschiedene Stufen, durch welche die Software getestet wird. Diese Einordnung erfolgt dem Entwicklungszustand des Systems. Der Inhalt orientiert sich an den Entwicklungsstufen des V-Modells, wobei jede Teststufe gegen die Systementwürfe und Spezifikationen der zugehörigen Entwicklungsstufe getestet wird. Beim V-Modell erfolgt als erstes die Systemanforderungsanalyse, dann die System-Architektur und der Softwareentwurf. Beim Aufstieg des V's erfolgen dementsprechend die Tests. Der Systementwurf wird mit der Zeit immer detaillierter, bei den Tests ist es genau andersherum, diese fangen mit den Detailliertesten an und werden mit der Zeit immer gröber.

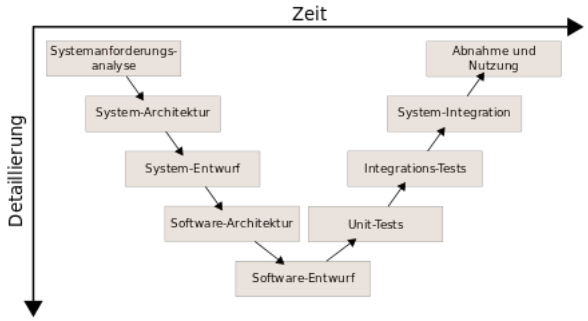


Abbildung 1.1: V-Modell

Quelle:V-Modell.Softwaretest.Wikipedia-DiefreieEnzyklopädie.\http://de.wikipedia.org/wiki/Softwaretest

Kapitel 2

Tests

Die Komponenten-Tests, welche auch Unit-Tests genannt werden, testen die einzelnen Module der Software. Getestet wird die Funktionalität innerhalb einzelner abgrenzbarer Teile der Software. Ziel ist die technische Lauffähigkeit und korrekte fachliche Ergebnisse nachzuweisen. Getestet wird dies meist vom Entwickler selbst.
Die Integrationstests testen die Zusammenarbeit voneinander abhängiger Komponenten. Getestet wird vor allen Dingen, ob die Schnittstellen der beteiligten Komponenten korrekte Ergebnisse über komplette Abläufe liefern.
Systemtests testen das gesamte System gegen die gesamten Anforderungen. Dieser Test findet in einer Testumgebung mit Testdaten statt und wird meistens durch die realisierende Organisation durchgeführt.
Abnahmetests, welche auch Verfahrens- oder Akzeptanztests genannt werden, sind das Testen der gelieferten Software durch den Kunden.
Softwaretests können auch nach anderen Kriterien klassifiziert werden. Man kann sie beispielsweise nach ihrer

- Prüftechnik (statisch/dynamisch, nach dem
- Testkriterium (Funktionale, Schnittstellen, Stress...,) nach
- Zeitpunkt der Durchführung, nach
- Testintensität, oder nach
- Informationsstand über die zu testenden Komponenten

einordnen. Bei der letzten Klassifizierungsart handelt es sich um die sogenannten Black- und White Box Test.

2.1 Black- und White-Box Tests

Eine weitere Möglichkeit Software zu verifizieren und zu validieren sind Black- und White-Box Tests. Die Namensgebung der Tests erfolgt nach der Funktion der Tests. Black-Box Tests kann man sich wie eine schwarze Box vorstellen, bei der man das Innere, also die Funktionsweise nicht sehen kann. Bei White-Box Tests kann man sich eine weiße

Box vorstellen, bei der man das Innere sehen kann, man also Wissen über die innere Funktionsweise hat.
Black-Box Tests sind also funktionsorientierte Tests. Sie überprüfen das nach außen sichtbare Verhalten. Ziel der Black-Box Tests ist es die Übereinstimmung eines Softwaresystems mit seiner Spezifikation zu überprüfen.
Bei den White-Box Tests hingegen handelt es sich um Tests, bei denen man Kenntnisse über die innere Funktionsweise des zu testenden Systems hat und diese überprüft. Bei den White-Box Tests werden meist vier Verfahren benutzt:

- Anweisungsüberdeckung: Hierbei wird jede Anweisung des Programms mindestens einmal durchlaufen.
- Zweigüberdeckung: Hier wird jeder Zweig des Programms mindestens einmal durchlaufen. Dies spielt besonders bei der Fallunterscheidung eine wichtige Rolle.
- Pfadüberdeckung: Jeder mögliche Pfad der Anwendung wird mindestens einmal durchlaufen.
- Bedingungsüberdeckung: Jede Teilbedingung und Bedingung wird mindestens einmal durchlaufen und ist einmal 'true' und einmal 'false'

2.1.1 Beispiel Black-Box Test

Nehmen wir an, wir wollen die Methode sqrt(double) mit der folgenden Spezifikation testen:

```
**********************************************************************
/**
* @params a Wert von dem die Wurzel berechnet wird
* @pre a >= 0
* @return returnValue * returnValue = a mit Genauigkeit +/-0.01
* @throws IllegalArgumentException, wenn a < 0
*/
public static double sqrt(double a){
...
}
**********************************************************************
```

Unsere Funktion berechnet also die Wurzel aus unserer Eingabe, welche eine Zahl vom Typ 'double' sein soll, und kann daher keine negativen Werte als Eingabe bekommen. Nun kann man die Wertebereiche sinnvoll einteilen, so dass Äquivalenzklassen entstehen. Eine Sinnvolle Einteilung wäre eine Aufteilung in einen ungültigen Eingabebereich, bei dem a <= 0 gilt, und einen gültigen Eingabebereich. In unserem Fall sollte bei a <= 0 eine Exception geworfen werden. Bei dem gültigen Eingabebereich sollte eine korrekte Wurzel gezogen werden. Ein weiterer wichtiger Bereich der durch Tests abgedeckt werden sollte ist die Randwertanalyse. Werte an den Rändern der Äquivalenzklassen sollten besonders gut auf ihre Zuverlässigkeit überprüft werden.

2.1.2 Beispiel White-Box Test

Nehmen wir an, wir wollen eine Methode f(int a, int b) testen:

```
*****************************
public int f(int a, int b){
    if(a>0){ //if1
    ...
    }
    if(b>0){ //if2
    ...
    }
}
*****************************
```

- Für die Anweisungsüberdeckung benötigen wir nur einen Testfall: f(1,1)
- Für die Zweigüberdeckung benötigen wir zwei Testfälle: f(0,0) und f(1,1)
- Für die Pfadüberdeckung benötigen wir vier Testfälle:
  - if1, if2 mit f(1,1)
  - !if1, if2 mit f(0,2)
  - if1, !if2 mit f(1,0)
  - !if1, !if2 mit f(0,0)
- Für die Bedingungsüberdeckung benötigen wir nur zwei Testfälle:
  - if1, if2 mit f(1,1)
  - !if1, !if2 mit f(0,0)

2.1.3 Vor- und Nachteile

Vorteile des Black-Box Tests gegenüber White-Box Tests sind zum einen eine bessere Verifikation des Gesamtsystems, da der Code gegen die vorher festgelegten Anforderungen getestet wird. Weitere Vorteile sind die Möglichkeit den Code auf semantische Eigenschaften zu testen und die Portabilität von semantisch erstellten Testfällen auf Plattform unabhängigen Implementierungen.
Nachteile des Back-Box Tests sind unter anderem der größere organisatorische Aufwand, dass zusätzlich eingeführte Funktionen bei der Implementierung nur durch Zufall getestet werden und, dass Testsequenzen unbrauchbar sind, wenn die Spezifikation unzureichend ist.
Vorteile von White-Box Tests sind, dass alle Teilkomponenten und die innere Funktionsweise getestet werden. Außerdem ist der organisatorische Aufwand der White-Box Tests geringer und sie können durch die richtigen Tools automatisiert werden.
Nachteile der White-Box Tests sind zum einen, dass sie die Erfüllung der Spezifikation

überprüfen und, dass durch die Orientierung am Quellcode eventuell um Fehler herum getestet wird.
Um also einen möglichst fehlerfreien Code zu gelangen sollten sowohl Black- als auch White-Box Tests an einem System durchgeführt werden.

2.2 Grey-Box Tests

Es gibt auch sogenannte Grey-Box Tests, welche die Vorteile von Black- und White-Box Tests verbinden sollen. Bei diesen Tests wird der Test des zu testenden Systems geschrieben, bevor der zu testende Code implementiert wird. Das Vorgehen unterscheidet sich also grundlegend von dem, der herkömmlichen Programmierung.
Mit den Black-Box Test haben Grey-Box Tests also gemeinsam, dass es bei der Erstellung des Tests keine Kenntnisse über das Interne gibt, da der eigentliche Programmcode noch gar nicht geschrieben wurde.
Die Gemeinsamkeit mit dem White-Box Test besteht darin, dass der Test von dem Entwickler des zu testenden Programms stets selbst geschrieben wird, wodurch der Programmierer weiß, welche Anforderungen der zu implementierende Code erfüllen muss.
Der Ansatz der Grey-Box Tests findet vor allem bei der testgetriebenen Entwicklung Anwendung.

Kapitel 3

Testgetriebene Entwicklung/Testdriven Development (TDD)

Bei der testgetriebenen Entwicklung handelt es sich um ein Vorgehen, bei dem die Softwaretests konsequent vor den zu testenden Komponenten geschrieben werden. Hintergrund dieses Vorgehens ist, dass man bei der traditionellen Programmierung oft nicht die gewünschte und erforderliche Testabdeckung erreicht. Dies liegt oftmals an einer fehlenden oder mangelnden Testbarkeit des Systems, an der Erstellung von Tests unter Zeitdruck oder einfach an der Nachlässigkeit der Programmierer bei der Testerstellung. Das Vorgehen bei der testgetriebenen Entwicklung kann man am Modell der wissenschaftlichen Methode veranschaulichen

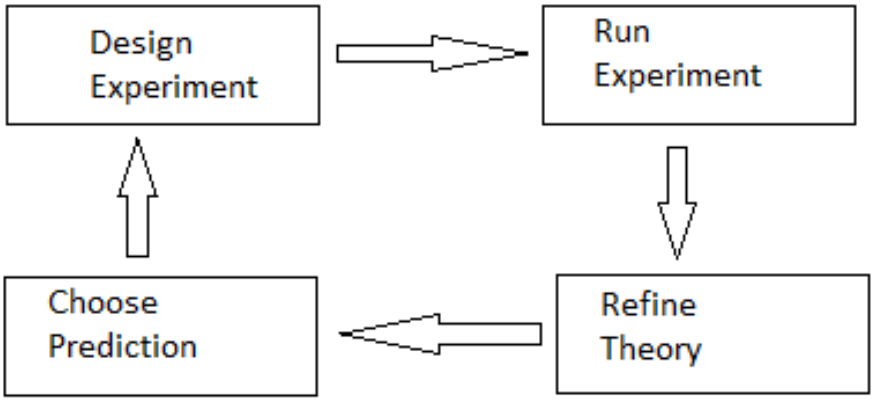


Abbildung 3.1: Modell der wisschenschaftlichen Methode

Quelle:ModellderwissenschaftlichenMethode:TestDrivenDevelopmentandtheScientificMethod.RickMudridge.IEEE.2003

Bei der wissenschaftlichen Methode wird wie folgt vorgegangen:

- Zuerst stell der Wissenschaftler eine Hypothese auf,
- dann entwirft er ein Experiment, welches die Hypothese bestätigen soll.
- Als nächstes führt er das Experiment durch

- und mit Hinblick auf die Ergebnisse des Experiments wird die Theorie verfeinert und weiterentwickelt.
- Daraufhin kann die nächste Hypothese aufgestellt werden und das ganze geht von vorne los.

Bei der testgetriebenen Entwicklung läuft das ganze sehr ähnlich ab. Hierbei nehmen die Tests die Rolle der Experimente ein und das Design die Rolle der Theorie:

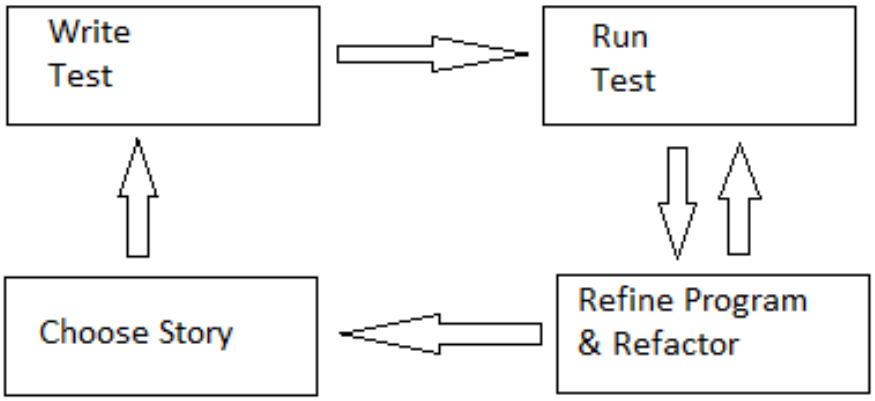
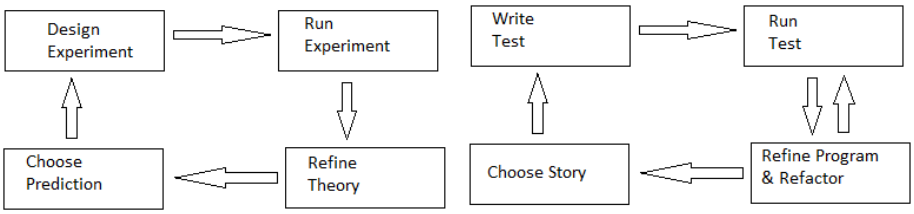


Abbildung 3.2: Modell der testgetriebenen Entwicklung

Quelle:ModelldertestgetriebenenEntwicklung:TestDrivenDevelopmentandtheScientificMethod.RickMudridge.IEEE.2003

- Zuerst wählt man einen Bereich des Designs oder Aufgabenbereichs, der die Entwicklung am besten vorantreibt.
- Dann schreibt man einen konkreten Test, der so einfach wie möglich sein soll, und
- führt diesen aus um sicher zu gehen, dass der Test zunächst fehlschlägt.
- Nun verändert man das System so, dass es den Test erfüllt, sowie alle vorher geschriebenen Tests auch.
- Möglicherweise muss man noch Refactoring am Programm durchführen, um Redundanzen zu vermeiden. Hierbei muss darauf geachtet werden, dass die Tests auch nach dem Refactoring noch durchlaufen.
- Danach kann man mit dem nächsten Test weiter machen

Experimentelle Reproduktivität wird hierbei durch den fortlaufenden Gebrauch von automatisierten Tests geschaffen, um sicher zu stellen, dass die Spezifikation weiterhin erfüllt wird.



(a) Modell der wisschenschaftlichen Methode

(b) Modell der testgetriebenen Entwicklung

Abbildung 3.3: Im Vergleich

3.1 Kritik

Viele äußern jedoch Kritik an der testgetriebenen Entwicklung. Zum einen muss man sehr konsequent Vorgehen, denn ohne ausreichende Tests wird keine ausreichende Testabdeckung für das Refactoring und die gewünschte Qualität erreicht. Die meisten Programmierer haben bisher nur sehr wenig Erfahrung mit testgetriebener Entwicklung gemacht und die Transition ist sehr schwierig. Daher leidet die Produktivität der Entwickler unter dem hohen Zeitaufwand beim erlernen der neuen Technik. Außerdem wird oft angemerkt, dass testgetriebene Entwicklung keinen Ersatz für weitere Testarten bietet. Fehler, die im Zusammenspiel zwischen verschiedenen Programmteilen entstehen, können beispielsweise eher durch Integrationstests gefunden werde. Testgetriebene Entwicklung bietet also keinesfalls eine Garantie für Fehlerfreiheit. Darüber hinaus gibt es keine feste Test Suite, alle Tests müssen eigenständig und individuell entwickelt werden.

Kapitel 4

Extreme Programmierung (XP)

Testgetriebene Entwicklung findet vor allem bei der Extremem Programmierung Anwendung. Bei der extremen Programmierung steht das Lösen einer Programmieraufgabe stets im Vordergrund. Hierbei wird davon ausgegangen, dass die Anforderungen des Kunden zu Beginn der Projektes noch nicht ganz klar sind. Die Aufgabe ist es also sich an die Anforderungen der Kunden anzunähern. Hierbei ist die aktive Teilnahme des Kunden am Produkt von großer Erforderlichkeit, denn nur dadurch können Anpassungen an die Anforderungen des Kunden kontinuierlich mit einbezogen werden.
Ziel dieses Ansatzes ist es, dass nur das, was für den Kunden einen Nutzen hat verwirklicht wird.

4.1 Vorteile der Extremen Programmierung

Die Extreme Programmierung bietet viele Vorteile gegenüber der traditionellen Programmierung. Zum einen verhindert die kundennahe Entwicklung das Erstellen unerwünschter oder unbrauchbarer Software. Darüber hinaus entstehen durch das vorzeitige Testen und die hohe Testabdeckung weniger Fehler im Ergebnis. Des Weiteren gibt es in der Extremen Programmierung keine strikte Rollentrennung und durch den allgemeinen Wissensaustausch und die stetige Kommunikation wird einem Wissensmonopol vorgebeugt, wodurch eine Kompensation bei Krankheitsfällen erleichtert wird.

Kapitel 5

Experimente

5.1 Annahmen

Es wurden einige Fall Studien durchgeführt, um die testgetriebene Entwicklung auf mögliche Vorteile gegenüber der traditionellen Programmierung zu testen.
Ein möglicher Vorteil ist Effizienz, da man durch das Vorgehen kontinuierlich Feedback zu seinem Code bekommt. Dadurch werden Fehler und Defects früh erkannt und die Quelle des Problems ist leichter erkennbar.
Des Weiteren haben die Wissenschaftler vermutet, dass sich die zusätzliche Zeit, die beim Testen aufgebracht wird, mit der Zeit ausgleicht, die ansonsten benötigt wird um Fehler zu finden. Es soll also entgegen der Kritik keine Zeiteinbuße geben.
Ein weiterer Vorteil ist das Ausnutzen der Vorteile von Tests. Programmierer die testgetriebene Entwicklung betreiben sollen dazu neigen Code zu schreiben, der durch automatisierte Tests geprüft werden kann. Dies führt zu einem zuverlässigerem System, zur Verbesserung der Qualität des Testaufwandes, zu einer Reduzierung des Testaufwandes und zu einer Minimierung des Zeitplans.
Außerdem soll testgetriebene Entwicklung natürlich zu einer Reduzierung von Programmfehlern führen. Die Tests bei der testgetriebenen Entwicklung können als Regressionstests angesehen werden. Regressionstests bezeichnen die Wiederholung von Testfällen, um sicher zu stellen, dass Modifikationen in bereits getesteten Teilen der Software keine neuen Fehler verursachen. Ich werde mich im folgenden auf das Paper 'Test-Driven Development as a Defect-Reduction Practice' von L.Williams, E.M.Maximilien und M.Vouk beziehen. In diesem Paper wird davon ausgegangen, dass alle Tests, die während der testgetriebenen Entwicklung geschrieben werden, automatisierte Tests sind.

5.2 1.Experiment: Studienabsolventen

In dem Paper von L.Williams, E.M.Maximilien und M.Vouk wird zunächst auf ein Experiment von Muller und Hagner eingegangen. 19 Studienabsolventen, also noch relativ unerfahrene Programmierer, wurden in zwei Gruppen geteilt. Eine Gruppe sollte testgetriebene Entwicklung betreiben und die andere Gruppe sollte zuerst den Code implementieren und danach automatisierte Tests erstellen. Beide Gruppen bekamen dieselbe Aufgabe. Sie sollten jeweils ein Programm mit gegebener Spezifikation und

gegebenen Design mit Methodendeklarationen vervollständigen. Es gab vier Phasen des Experimentes:

1. Initiale Implementierung
2. Evaluierung, hier wurden Tests durchgeführt und Feedback gegeben
3. Akzeptanzphase
4. Post-Experimentelle Analyse

Es wurden drei Kriterien untersucht:

- die Entwicklungszeit
- die Zuverlässigkeit und
- die Verständlichkeit des Systems

Die Auswertung der Ergebnisse führte zu keinem eindeutigen Vorteil der testgetriebenen Entwicklung gegenüber der traditionellen Programmierung. Es gab keinen Unterschied in der Entwicklungszeit. Außerdem wies die testgetriebene Entwicklung weniger Zuverlässigkeit nach der Implementierungsphase, aber mehr Zuverlässigkeit nach der Akzeptanzphase nach. Die testgetriebene Entwicklung schien also weder schneller zu sein, noch eine bessere Qualität der Software vorweise zu können. Aus der Auswertung geht jedoch nicht hervor, wie genau die Zuverlässigkeit getestet wurde.

5.3 2.Experiment: Professionelle Programmierer

Muller und Hagner führten ein weiteres Experiment durch. Dieses Mal wurden 24 Professionelle Programmierer untersucht. Diese teilten sie wieder in zwei Gruppen, eine Gruppe würde testgetriebene Entwicklung und die andere, die Kontroll-Gruppe, eine Wasserfall ähnliche Entwicklung durchführen.

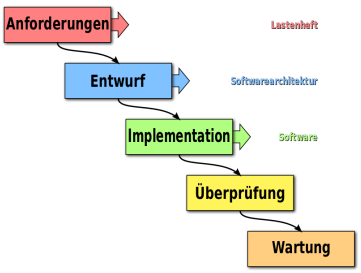


Abbildung 5.1: Wasserfallmodell

Quelle:Wasserfall-Modell:Wasserfallmodell..Wikipedia-DiefreieEnzyklopädie.\https://de.wikipedia.org/wiki/Wasserfallmodell#/media/File:Waterfall_model-de.svg

Alle Programmierer führten Paar-Programmierung durch.
Die Auswertung der Ergebnisse zeigte, dass bei der testgetriebenen Entwicklung 18% mehr erfolgreiche Black-Box Tests durchgeführt wurden als bei der Kontroll-Gruppe. Des weiteren wies die Gruppe der testgetriebenen Entwicklung ein einfacheres Design nach. Auf der anderen Seite stellte sich heraus, dass die testgetriebene Entwicklung tatsächlich mehr Zeit in Anspruch nahm. Insgesamt benötigte diese Gruppe 16% mehr Zeit zum programmieren als die Kontrollgruppe welche traditionelle Entwicklung durchführte.
Leider entwickelte die Kontrollgruppe jedoch keine rentablen automatisierten Tests, was den Vergleich der unterschiedlichen Vorgehensweisen erheblich erschwert, da die Entwicklung der testgetriebenen Entwicklung auf automatisierten Tests basiert.

5.4 3.Experiment: IBM Entwickler

Ein Experiment von L.Williams, E.M.Maximilien und M.Vouk von IBM Softwareentwicklern sollte Aufschluss liefern.
Die IBM Gruppe arbeitet seit Jahren an der Entwicklung von Gerätetreibern. In dem Experiment verglichen die Forscher eine erfahrene Gruppe von Softwareentwicklern, die an der 7. Veröffentlichung einer alten Plattform arbeitet mit einer relativ unerfahrenen Gruppe, die an der 1. Veröffentlichung einer neuen Plattform arbeitet. Diese zweite Gruppe verwendete das Verfahren der testgetriebenen Entwicklung. Es wurde vor allem die Fehler-Reduktion durch testgetriebene Entwicklung untersucht. Bei diesem Experiment wurde die testgetriebene Entwicklung stets nach dem UML-Design Prozess durchgeführt. Der Programmentwurf erfolgte also vor der Testentwicklung. Nachdem der Großteil des Codes implementiert war, wurden die Gerätetreiber zu einem anderen Team geschickt, welche auf diesen funktionale Verifikations-Tests durchführten. Diese hatten Black-Box Tests geschrieben, welche die System-Spezifikation überprüfen sollten.
Die Auswertung der Ergebnisse lieferte Folgendes:

- Bei der testgetriebenen Entwicklung wurden insgesamt 40% weniger Programmfehler gefunden.
- Die Anzahl der positiven funktionalen Verifikations-Tests war bei beiden Gruppen fast identisch.

Man muss für den Vergleich aber auch beachten, dass das alte Produkt auf zwei verschiedenen und das Neue nur auf einer Plattform laufen muss. Da das alte Produkt auf mehr Hardware Plattformen funktionieren muss, müssen die Testfälle auch für jede Plattform wiederholt werden. Es müssen also deutlich mehr Tests durchgeführt werden, als bei dem neuen Produkt.

5.5 Auswertung

Es tauchten 40% weniger Programmfehler bei der testgetriebenen Entwicklung auf. Der Kritikpunkt, dass testgetriebene Entwicklung sich negativ auf die Produktivität der

Programmiere auswirkt wurde für dieses Experiment also widerlegt. Die testgetriebene Entwicklung wirkte sich nur minimal auf die Produktivität der Entwickler aus. Darüber hinaus sind die automatisierten Tests die bei der testgetriebenen Entwicklung implementiert wurden wiederverwendbar und erweiterbar, was einen deutlichen Vorteil gegenüber anderen Tests liefert.
Die vielen Tests bei der testgetriebenen Entwicklung dienen als Basis für Qualitätssicherung und als Vertrag zwischen allen Mitgliedern des Teams. Zudem können die automatisierten Tests als Regressionstests dienen und somit die Qualität der Software sicherstellen.
Nach dem Experiment gaben die Programmierer der IBM Gruppe an der testgetriebenen Entwicklung sehr positiv gegenüber zu stehen und führten die Verwendung des Verfahrens fort. Obwohl sich Ergebnisse der unterschiedlichen Vorgehensweisen in diesem Experiment nur relativ Vergleichen lassen, liefert die testgetriebene Entwicklung, obwohl diese von unerfahreneren Programmieren durchgeführt wurde, sehr positive Ergebnisse gegenüber der Wasserfall ähnlichen Vorgehensweise.

Kapitel 6

Fazit

Fassen wir alles noch einmal zusammen.
Softwaretests sind sehr wichtig um die Qualität der Software zu sichern. Bei der testgetriebenen Entwicklung werden Test vor dem zu testenden Programmcode geschrieben. Dadurch entsteht eine sehr hohe Testabdeckung. Da die meisten Programmierer wenig Erfahrung mit testgetriebener Entwicklung haben und das Umstellen sehr schwer sein kann benötigen sie viel Übung was einen erhöhten Zeitaufwand erfordert.
Außerdem ist zu beachten, dass testgetriebene Entwicklung keine Fehlerfreiheit des Programms garantiert und dass zusätzliche Tests zur Sicherung der Fehlerfreiheit durchgeführt werden sollten.
Die Fall Studien haben gezeigt, dass testgetriebene Entwicklung sehr effizient sein kann, da durch dieses Verfahren weniger Defects auftauchen als bei der traditionellen Programmierung. Außerdem haben die Fall Studien gezeigt, dass testgetriebene Entwicklung keine ausschlaggebenden Auswirkungen auf die Produktivität der Entwicklung hat. Inzwischen wird in vielen Unternehmen viel Hoffnung in die testgetriebene Entwicklung gesetzt.
Meiner Meinung nach ist testgetriebene Entwicklung ein sehr guter Ansatz, da sie eine sehr hohe Testabdeckung bietet, und sollte von jedem Entwickler im Laufe seiner Karriere zumindest einmal ausprobiert werden. Mit ein wenig Übung gibt es dann auch keine beziehungsweise nur sehr wenig Zeiteinbuße bei der Entwicklung. Die Studien weisen jedoch erhebliche Unstimmigkeiten auf und sollten nur mit Bedacht als Anhaltspunkt für einen Vergleich zwischen traditioneller und testgetriebener Entwicklung dienen.

Kapitel 7

Literaturverzeichnis

- Test Driven Development and the Scientific Method. Rick Mudridge. IEEE.2003
- Test-Driven Development as a Defect-Reduction Practice. Laurie Williams, E.Michael Maximilien, Mladen Vouk.IEEE.2003
- Black-Box-Test. Wikipedia – Die freie Enzyklopädie. http://de.wikipedia.org/wiki/BlackBox-Test
- Extreme Programmierung. Wikipedia – Die freie Enzyklopädie. http://de.wikipedia.org/wiki/Extreme_Programming
- Grey-Box-Test. Wikipedia – Die freie Enzyklopädie. http://de.wikipedia.org/wiki/GreyBox-Test
- Softwaretest.Wikipedia - Die freie Enzyklopädie. http://de.wikipedia.org/wiki/Softwaretest
- Testautomatisierung.Wikipedia – Die freie Enzyklopädie. http://de.wikipedia.org/wiki/Testautomatisierung
- Testgetriebene Entwicklung. Frank Westphal. Extreme Programmer. Ruby on Rails Freelancer. Web 2.0 Technologist. 06.01.2002 http://www.frankwestphal.de/TestgetriebeneEntwicklung.html
- Testgetriebene Entwicklung. Wikipedia – Die freie Enzyklopädie. http://de.wikipedia.org/wiki/Testgetriebene_Entwicklung
- White-Box-Test. Wikipedia – Die freie Enzyklopädie. http://de.wikipedia.org/wiki/WhiteBox-Test

7.1 Abbildungsverzeichnis

- V-Modell: Softwaretest.Wikipedia - Die freie Enzyklopädie. http://de.wikipedia.org/wiki/Softwaretest
- Modell der wissenschaftlichen Methode: Test Driven Development and the Scientific Method. Rick Mudridge. IEEE.2003

- Modell der testgetriebenen Entwicklung: Test Driven Development and the Scientific Method. Rick Mudridge. IEEE.2003
- Wasserfall-Modell: Wasserfallmodell..Wikipedia - Die freie Enzyklopädie. https://de.wikipedia.org/wiki/Wasserfallmodell#/media/File:Waterfall_model-de.svg