Testautomatisierung mit Selenium

- Teilautomatisierte Generierung von Page Objects -

Fakultät für Informatik und Mathematik
der Hochschule München

Masterarbeit

vorgelegt von

Matthias Karl
Matrikel-Nr: 03280712

im <Datum>

Prüfer: Prof. Dr. Ullrich Hafner

Eidesstattliche Erklärung

Hiermit versichere ich, die vorliegende Studienarbeit selbstständig und nur unter Verwendung der von mir angegebenen Quellen und Hilfsmittel verfasst zu haben. Sowohl inhaltlich als auch wörtlich entnommene Inhalte wurden als solche kenntlich gemacht. Die Arbeit hat in dieser oder vergleichbarer Form noch keinem anderem Prüfungsgremium vorgelegen.

Datum: _____ Unterschrift: _____

Zusammenfassung / Abstract

Inhaltsverzeichnis

1 Einleitung

2 Grundlagen

2.1 Software-Qualität

Nahezu jeder Programmierer ist schon einmal mit dem Begriff der Software-Qualität in Berührung gekommen. Diesen Qualitätsbegriff jedoch genau zu fassen erweist sich als schwierig. Die DIN-ISO-Norm 9126 definiert Software-Qualität wie folgt:
„Software-Qualität ist die Gesamtheit der Merkmale und Merkmalswerte eines Software-Produkts, die sich auf dessen Eignung beziehen, festgelegte Erforderniss zu erfüllen."[ISO01] Aus dieser Definition wird deutlich, dass es sich bei dem Begriff der Software-Qualität eine multikausale Größe handelt. Das bedeutet, dass zur Bestimmung der Qualität einer Software nicht ein einzelnes Kriterium existiert. Vielmehr verbergen sich hinter dem Begriff eine ganze Reihe verschiedener Kriterien die je nach den gestellten Anforderungen in ihrer Relevanz variieren.[Hof13, vgl. S.6 ff.] Sammlungen solcher Kriterien werden in sogenannten Qualitätsmodellen zusammengefasst. Die DIN-ISO-Norm 9126 bietet selbst ein solches Qualitätsmodell und definiert damit eine Reihe von wesentlichen Merkmalen, die für die Beurteilung der Software-Qualität eine Rolle spielen. Diese Merkmale sind in der Abbildung 2.1 zusammengefasst. Eine nähere Definition der einzelnen Begriffe des Qualitätsmodells kann beispielsweise dem Buch Software-Qualität von Dirk W. Hoffmann entnommen werden. [Hof13, S.7 ff.] Um die Qualität einer Software zu Steigern, bietet die moderne Software-Qualitätssicherung eine Vielzahl von Methoden und Techniken. Ein Teil der Methoden versucht durch eine Verbesserung des Prozesses der Produkterstellung die Entstehung von qualitativ hochwertigen Produkten begünstigt. Diese Methoden fallen in den Bereich der Prozessqualität. Einen weiteren Bereich bilden die Methoden die zur Verbesserung der Produktqualität dienen. Bei diesen Methoden wird das Softwareprodukt direkt bezüglich der Qualitätsmerkmale überprüft. Dieser Bereich unterteilt sich in die konstruktive und analytische Qualitätssicherung. [Hof13, vgl. S.19 ff.] Unter konstruktiver Qualitätssicherung versteht man den Einsatz von z.B. Methoden, Werkzeugen oder Standards die dafür sorgen, dass ein Produkt bestimmte Forderungen erfüllt. Unter analytische Qualitätssicherung versteht man den Einsatz von analysierenden bzw. prüfenden Verfahren, die Aussagen über die Qualität eines Produkts machen. [Pro06] In

Quelle: [ISO01]

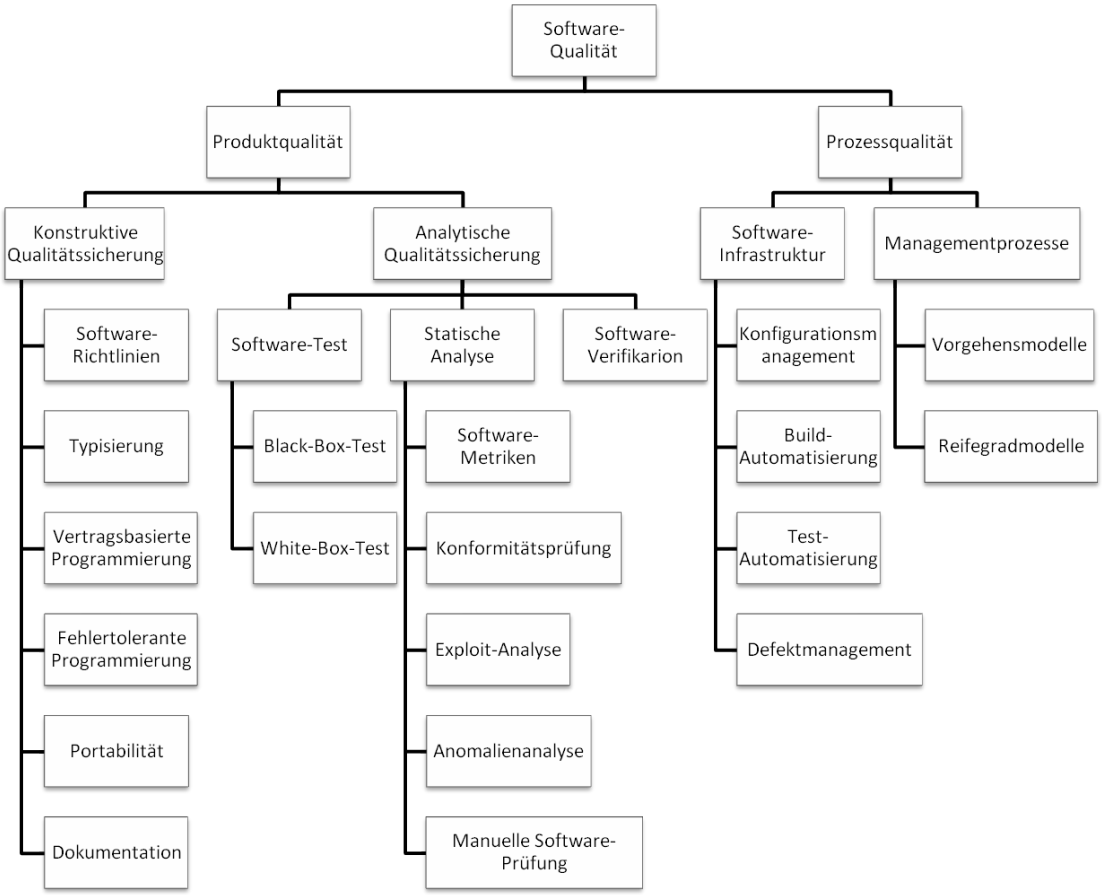
Abbildung 2.1: Qualitätsmerkmale von Softwaresystemen (ISO 9126)

diesem Bereich der Qualitätssicherung befindet sich unter anderem der klassische Software-Test. Eine Übersicht über das gesamte Gebiet der Software-Qualitätssicherung, wie es sich uns gegenwärtig darstellt, ist in Abbildung 2.2 dargestellt.

2.2 Softwaretest

Im laufe der Zeit wurden viele Versuche unternommen um die Qualität von Software zu steigern. Besondere Bedeutung hat dabei der Software-Test erlangt. Der IEEE Std 610.12 definiert den Begriff Test als das ausführen einer Software unter bestimmten Bedingungen mit dem Ziel, die erhaltenen Ergebnisse auszuwerten, also gegen erwartete Werte zu vergleichen. (Im Original: „An activity in which a system or component is executed under specific conditions, the results are observed or recorded, and an evaluation is made of some aspect of the system or component.“ [IEE91]) Bereits zu beginn der Softwareentwicklung hat man versucht Programme vor ihrer Auslieferung zu testen. Der dabei erzielte Erfolg entsprach nicht immer den Erwartungen. Im laufe der Jahre wurde das Testen daher auf eine immer breitere Grundlage gestellt. Es entwickelten sich Unterteilungen des Software-Tests die bis heute bestand haben. Hier wären zu nennen:

- White-Box-Test
- Black-Box-Test und externe Testgruppe



Quelle: [Hof13, vgl. S.20]

Abbildung 2.2: Übersicht über das Gebiet der Software-Qualitätssicherung

- Volume Test, Stress Test und Test auf Systemebene

Jeder dieser Begriffe beschreibt bestimmte Techniken, die bei konsequenter Anwendung dazu führen können Fehler in Softwareprodukten zu identifizieren. [Tha02, vgl. S.18]
Neben der Auswahl der richtigen Techniken für ein bestimmtes Problem spielt in der Praxis die Testkonstruktion eine zentrale Rolle. Bereits für kleine Programme ist es faktisch nicht mehr möglich das Verhalten einer Software für alle möglichen Eingaben zu überprüfen. Es muss sich daher immer auf eine vergleichsweise winzige Auswahl an Testfällen beschränkt werden. Testfälle unterscheiden sich jedoch stark in ihrer Relevanz. Die Auswahl der Testfälle hat daher einen großen Einfluss auf die Anzahl der gefundenen Fahler und damit auch auf die Qualität des Endprodukts. [Hof13, vgl. S.22]
Dennoch ist der Software-Test eines der am meisten verbreiteten Techniken zur Verbesserung der Softwarequalität. Um über lange Sicht gute Software zu produzieren reicht es jedoch nicht aus sich nur auf die Technik des Software-Tests zu stützen. Ein großer Nach-

teil des Softwaretests ist es, dass Fehler erst in einer relativ späten Phase der Entwicklung identifiziert werden. Je später eine Fehler jedoch erkannt wird, desto teurer wird auch seine Beseitigung. Abbildung 2.2 zeigt, dass der Software-Test nur eine von vielen Techniken des Qualitätsmanagement darstellt. Um die Probleme des Software-Tests auszugleichen, ist es daher ratsam, sich bei der Qualitätssicherung möglichst breit aufzustellen und sich nicht nur auf die analytische Qualitätssicherung in Form des Software-Tests zu verlassen. [Tha02, vgl. S.18]

2.3 Testautomatisierung

Das Testen von Software macht in heutigen Projekten einen großen Teil der Projektkosten aus. Studien haben gezeigt, dass das Testen für 50% und mehr der gesamten Projektkosten verantwortlich sein kann. Mit Steigender Komplexität der Softwäre steigen diese Kosten immer weiter an. [RW06] [Har00] Um diese Kosten zu reduzieren haben sich im laufe der Zeit die bestehenden Testmethoden immer weiter entwickelt und auch neue Ansätze herausgebildet. Einer dieser Ansätze ist es Software-Tests möglichst automatisiert durchzuführen. [Har00] Diesen Ansatz fast man mit dem Begriff Testautomatisierung zusammen. Seidl et al. definieren Testautomatisierung als „die Durchführung von ansonsten manuellen Testtätigkeiten durch Automaten.“[Sei12, S.7] Diese Definition zeigt, dass das Spektrum der Testautomatisierung breit gefächert ist. Testautomatisierung beschränkt sich nicht nur auf das automatisierte durchführen von Testfällen sondern erstreckt sich über alle Bereiche des Software-Test. [Die Verschiedenen Möglichkeiten der Testaustomatisierung werden in Kapitel.. dargestellt]. Aus Sicht des Qualitätsmanagement ist die Testautomatisierung sowohl den Methoden zur Steigerung der Produktqualität als auch der Prozessqualität zugeordnet. Ein automatisierter Software-Test hat immer noch den selben Charakter wie ein manueller Software-Test und ist daher ein Teil der analytischen Qualitätssicherung. Allerdings erfordert Testautomatisierung auch immer infrastrukturelle Anpassungen. Automatisierte Testfälle benötigen in der Regel eine Besondere Software-Infrastruktur wie etwa ein Automatisierungsframework. Solche Maßnahmen, die den Programmentwickler aus technischer Sicht in die Lage versetzen, seiner täglichen Arbeit in geregelter und vor allem produktiver Weise nachzugehen, werden den Methoden zur Verbesserung der Prozessqualität zugeordnet. (siehe Abbildung 2.2). [Hof13, vgl. Seite 25]

2.4 Testprozess

Um Software-Tests effektiv und Strukturiert durchzuführen wird eine verfeinerter Ablaufplan für die einzelnen Testaufgaben benötigt. Diesen Ablaufplan fassen Splinner und Linz im fundamentalen Testprozess zusammen. Die einzelnen Arbeitsschritte die im Lebenszyklus eines Software-Tests anfallen werden dabei verschiedenen Phasen zugeordnet. Durch den Testprozess wird die Aufgabe des Testens so in kleinere Testaufgaben untergliedert.
Testaufgaben, die man dabei unterscheidet sind:

- Testplanung und Steuerung
- Testanalyse und Testdesign
- Testrealisierung und Testdurchführung
- Testauswertung und Bericht
- Abschluss der Testaktivitäten

„Obgleich die Aufgaben in sequenzieller Reihenfolge im Testprozess angegeben sind, können sie sich überschneiden und teilweise auch gleichzeitig durchgeführt werden.“ [SL07, S.19] Auf Grundlage des fundamentalen Testprozesses nach Splinner und Linz werden im folgenden diese Teilaufgaben näher beschrieben. [SL07, S.20ff] Diese Beschreibung wird durch Ausführungen von Seidl et al. erweitert. [Sei12, Seite 9 ff.]

2.4.1 Testplanung und Steuerung

Um dem Umfang und der Komplexität heutiger Software-Tests gerecht zu werden, benötigt man zu Beginn des Testprozesses eine genaue Planung. Ziel dieser Planung ist es, den Rahmen für die weiteren Testaktivitäten festzulegen. Die Aufgaben und die Zielsetzungen der Tests müssen ermittelt wurden. Eine Ressourcenplanung wird benötigt und eine geeignete Teststrategie muss ermittelt werden. In Kapitel 2.2 wurde bereits erwähnt, dass das vollständige Testen einer Anwendung in der Regel nicht möglich ist. Die einzelnen Systemteile müssen daher nach Schwere der zu erwartenden Fehlerwirkung priorisiert werden. Um so schwerwiegender die zu erwartende Fehlerwirkung ist, umso intensiver muss der betrachtete Systemteil auch getestet werden. Ziel der Teststrategie ist also „die optimale Verteilung der Tests auf die »richtigen« Stellen das Softwaresystems.“ [SL07, S.21] Steht das Softwareprojekt unter einem hohen Zeitdruck müssen Testfälle zusätzlich Priorisiert werden. Um zu

verhindern, dass das Testen zu einem endlosen Prozess wird, müssen auch geeignete Testendekriterien festgelegt werden. Anhand dieser Kriterien kann später entschieden werden ob der Testprozess abgeschlossen werden kann.
Bereits zu Beginn des Testprozesses werden auch wichtige Grundsteine für eine spätere Testautomatisierung gelegt. Es muss entschieden werden, in welchen Teststufen und Testbereichen eine Automatisierung eingesetzt werden soll. Vor allem ist die frage zu klären ob und in welchem Ausmaß eine Automatisierung überhaupt sinnvoll ist. Entscheidet man sich für eine Testautomatisierung hat das in der Regel große Auswirkung auf die einzusetzenden Ressourcen und die Zeitliche Planung und Aufwandsschätzung. Oftmals wird im Rahmen der Tests eine besondere Werkzeugunterstützung oder Infrastruktur benötigt. Derartige Punkte müssen auch bereits in der frühen Planungsphase berücksichtigt werden. Wie Kapitel 2.2 gezeigt hat, ist das im besonderen der Fall wenn in der Planung beschlossen wird, dass die Testfälle automatisiert durchgeführt werden sollen.
Die Gesamten erarbeiteten Rahmenbedingungen werden in einem Testkonzept dokumentiert. Eine mögliche Vorlage für dieses Dokument bietet die internationale Norm IEEE 829-2008 [IEE08]. Neben der frühzeitigen Planung der Tests muss während des gesamten Testprozesses eine Steuerung erfolgen. Hierfür werden die Ergebnisse und Fortschritte der Tests und des Projekts laufend erhoben, geprüft und bewertet. Werden Probleme erkannt, kann so rechtzeitig gegengesteuert werden.

2.4.2 Testanalyse und Testdesign

In dieser Phase wird zunächst die Qualität der Testbasis überprüft. Alle Dokumente die für die Erstellung der Testfälle benötigt werden müssen in ausreichendem Detailgrad vorhanden sind. Mit Hilfe der qualitätsgesicherten Dokumente kann die eigentliche Testfallerstellung beginnen. Anhand der Informationen aus dem Testkonzept und den Spezifikationen werden mittels strukturierter Testfallerstellungsmethoden logische Testfälle erstellt. Diese logischen Testfälle können dann in einer späteren Phase konkretisiert werden, indem ihnen z.B. tatsächliche Eingabewerte zugeordnet werden. Für jeden dieser Testfälle müssen die möglichen Rand- und Vorbedingungen so wie ein erwartetes Ergebnis bestimmt werden. In dieser Phase beginnt auch die Umsetzung der Testfälle in der Testautomatisierung. Abgestimmt auf die ausgewählten Automatisierungswerkzeuge und die zu testende Software muss die Umgebung für die Testautomatisierung vorbereitet werden. Anhand der Vorgaben des Testkonzeptes können dann jene Testfälle und Testabläufe ausgewählt werden die im Zuge der Testautomatisierung implementiert werden sollen. Hierbei wird noch einmal die technische

Umsetzbarkeit der ausgewählten Testfälle geprüft. Bei der Auswahl der Testfälle sollte zu beginn eine möglichst breite Testabdeckung angestrebt werden. Problemfelder können dann später durch weitere Testfälle in in der Tiefe getestet werden.

2.4.3 Testrealisierung und Testdurchführung

In diesem Schritt des Testprozesses werden aus den Logischen Testfällen der vorangegangenen Phase konkrete Testfälle gebildet. Diese Testfälle werden anhand ihrer fachlichen und technischen Zusammengehörigkeit zu Testszenarien gruppiert und anhand der Vorgaben aus dem Testkonzept priorisiert. Sobald die zu testende Software zur Verfügung steht, kann mit der Abarbeitung der Testfälle begonnen werden. Die dabei erhaltenen Ergebnisse werden vollständig protokolliert. Werden im Zuge der Durchführung Fehler aufgedeckt, muss darauf in geeigneter Weise reagiert werden. Es könnte beispielsweise ein zuvor definierter Fehlerprozess gestartet werden. Korrekturen und nachgehende Veränderungen am Testobjekt werden durch eine Wiederholung der Testläufe abgedeckt. Aus Sicht der Testautomatisierung beginnt in dieser Phase die technische Umsetzung der Testfälle. In vielen Fällen bedeutet das Programmiertätigkeit. Diese Programmiertätigkeiten sind wiederum anfällig für eigene Fehler und müssen daher in angemessener Weise qualitätsgesichert werden. Auch bei der Testautomatisierung ist eine Zusammenfassung von Testfällen sinnvoll. Auf diese weise kann man funktionalen und logischen Abhängigkeiten zwischen den Testfällen gerecht werden. Nach der Implementierung können die geplanten Testfälle durchgeführt werden. Gerade bei der Automatisierung ist eine genaue Protokollierung der Ergebnisse besonders wichtig. Nur dadurch ist es später möglich, aufgetretene Fehler überhaupt zu lokalisieren.

2.4.4 Testauswertung und Bericht

In dieser Phase des Prozesses wird geprüft, ob die im Testkonzept definierten Testendekriterien erreicht wurden. Sind alle Forderungen erfüllt, kann es zu einem Abschluss der Testaktivitäten kommen. Kommt es zu Abweichungen im Bezug auf diese Kriterien, muss darauf entsprechend reagiert werden. Es können Fehlerkorrekturen durchgeführt werden oder neue Testfälle erstellt werden. Aber auch der umgekehrte Fall ist möglich. Es kann dazu kommen, dass Endekriterien nur mit unverhältnismäßig hohem Aufwand erreicht werden und daher bestimmte Testfälle entfallen oder Kriterien überdacht werden müssen.
Für die Testautomatisierung ist wesentliche Aufgabe dieser Phase die Auswertung und Aufarbeitung der erhaltenen Ergebnisse. Automatisierte Tests generieren oftmals eine Fülle an Log-Dateien und Protokollen. Um aus diesen Ergebnissen die richtigen Schlüsse zu ziehen

und sie für dritte zugänglich zu machen müssen sie in eine lesbare Form gebracht werden. In jedem Fall muss über die erhaltenen Ergebnisse und das daraus resultierende Vorgehen ein Testbericht erstellt werden. Je nach Umfang und Phase der Test kann dieser mehr oder weniger formal ausfallen. Für einen Komponententests reicht beispielsweise eine formlose Mitteilung. Höhere Teststufen erfordern allerdings einen formaleren Bericht.

2.4.5 Abschluss der Testaktivitäten

Sind die Testaktivitäten abgeschlossen sollten zum Schluss alle im laufe des Testprozesses gemachten Erfahrungen analysiert werden. So können die gewonnenen Erkenntnisse für spätere Projekte genutzt werden. Dadurch kann eine stetige Verbesserung des Testprozesses erreicht werden. Die während des Prozesses erstellte Testware sollte archiviert werden. Auf diese weise steht sie für folgende Regressionstests zur Verfügung. Die Kosten in Wartung und Pflege der Software können damit gesenkt werden. Bei der Testautomatisierung bedeutet das, die Wiederverstellbarkeit der Testumgebung und des Sourcecodes der Tests sicherzustellen.

Abschließend ist zu sagen, dass sich die Testautomatisierung in der Regel gut in einen bereits bestehenden Testprozess integrieren lasst. Sie wird allerdings „den Prozess nicht verbessern oder gerade richten, sondern nur unterstützen.“ [Sei12, S.21] Ist der Testprozess schon vor einführung einer Automatisierung schlecht gelaufen, wird er es nach der Einführung auch noch. Die Testautomatisierung ist also nicht als Heilmittel für schlecht laufende Prozesse gedacht sondern als Möglichkeit sich einen bereits gut laufenden Prozess effizienter zu gestalten.

2.5 Vorgehensmodelle

Der in Kapitel 2.4 beschriebene Testprozess ist nicht als losgelöster, eigenständiger Prozess zu betrachten. Vielmehr ist der Testprozess immer ein Teil eines größeren Entwicklungsablaufes bei der Erstellung eines Softwareprodukts. Einen solchen Entwicklungsablauf versucht man mit Hilfe von sogenannten Softwareentwicklungsmodellen, auch Vorgehensmodelle genannt, abzubilden. Ein Projekt wird dazu in einzelne Phasen untergliedert an deren Ende ein gewisses Ziel bzw. Ergebnis steht. Auf gröbster ebene können dabei vier Hauptphasen unterschieden werden, die auch so von Seidl et al. verwendet werden [Sei12, S.21 ff.]. Diese Phasen finden sich mehr oder weniger Ausgeprägt in den meisten der gängigen Vorgehensmodelle wieder:

- Spezifikation

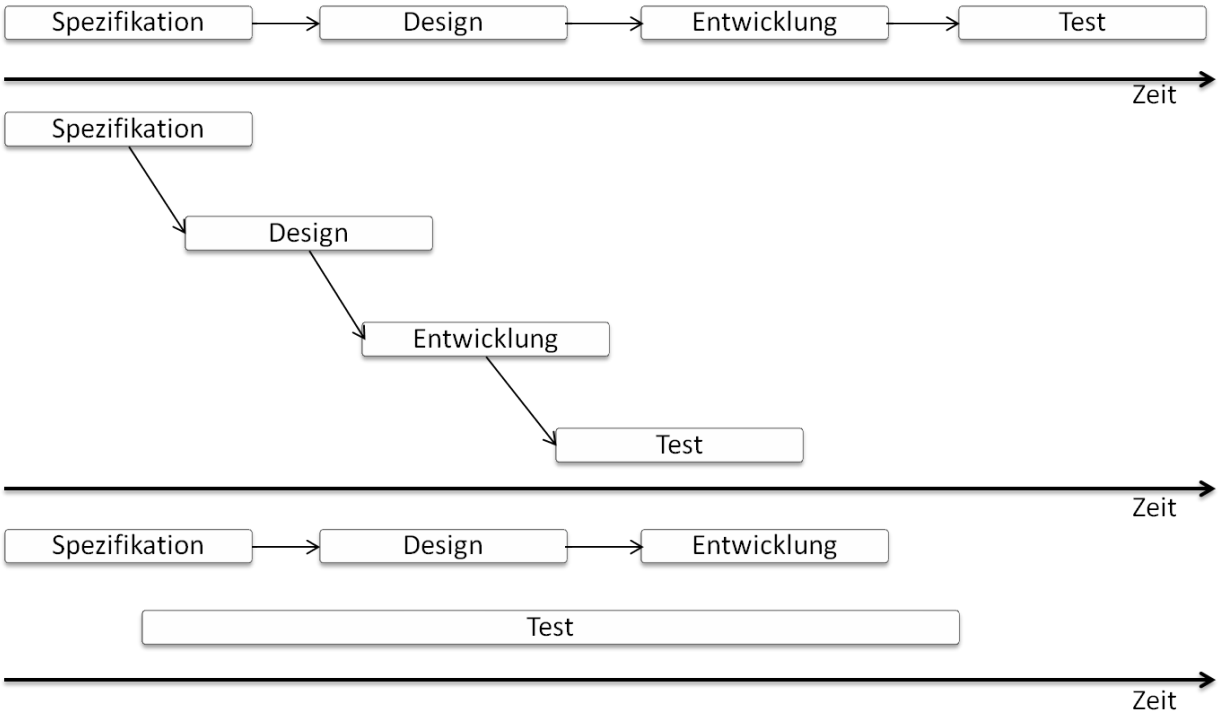
- Design
- Entwicklung
- Test

Das Testen, bzw. der Testprozess ist also eine von mehreren Phasen in solch einem Entwicklungsmodell. Es gibt eine Vielzahl von unterschiedlichen Softwareentwicklungsmodellen. Der Hauptunterschied liegt meist in der zeitlichen Koppelung und der inhaltlichen Ausprägung der einzelnen Phasen. Die einzelnen Phasen können sich innerhalb eines Vorgehensmodells überschneiden und wiederholen und müssen auch nicht immer wie in der Auflistung angegeben sequentiell abgearbeitet werden. Aus der Sicht der Testautomatisierung ist eine Einteilung der verschiedenen Vorgehensmodelle in zwei Gruppen sinnvoll: [Sei12, vgl. S.21 ff.]

- Klassische Entwicklungsmodelle, die eher sequentiell ausgerichtet sind
- Iterative und agile Entwicklungsmodelle, die sich durch Parallelisierung und kurze Iterationen auszeichnen.

2.5.1 Klassische Entwicklungsmodelle

Die hier als Klassische Entwicklungsmodelle betitelten Vorgehensmodelle zeichnen sich vor allem dadurch aus, dass die einzelnen Phasen sequentiell ausgeführt werden. Der bekannteste Vertreter dieser Vorgehensmodelle ist das Wasserfallmodell [Roy87]. In diesem Modell sind alle Phasen strickt voneinander getrennt. Eine neue Phase kann erst begonnen werden, wenn eine vorangegangene Phase abgeschlossen wurde. Rücksprünge in vorangegangene Phasen sind unerwünscht. In der Praxis wird diese vorgehen jedoch oft nicht ganz so strickt umgesetzt. Es kommt zu Mischformen, bei denen die einzelnen Phasen nicht mehr voll sequentiell abgearbeitet werden sondern sich teilweise überlagern. Vor allem im Bereich des Testens geht man oft dazu über, dieses strikte Vorgehensmodell aufzuweichen. Das Testen ist meist keine getrennte Phase am Ende des Entwicklungsprozesses sondern erstreckt sich über den gesamten Prozess ausgehend von der frühen Spezifikationsphase.
In Projekten die ein solch sequentielles Vorgehen wählen, muss bereits in der Planungsphase des Testprozesses genau abgewägt werden, ob eine Automatisierung der Testfälle überhaupt sinnvoll ist. Wenn zu Beginn des Projektes schon klar ist, dass die Testfälle nur ein einziges mal am Ende des Entwicklungsprozesses ausgeführt werden, steht eine Automatisierung oft



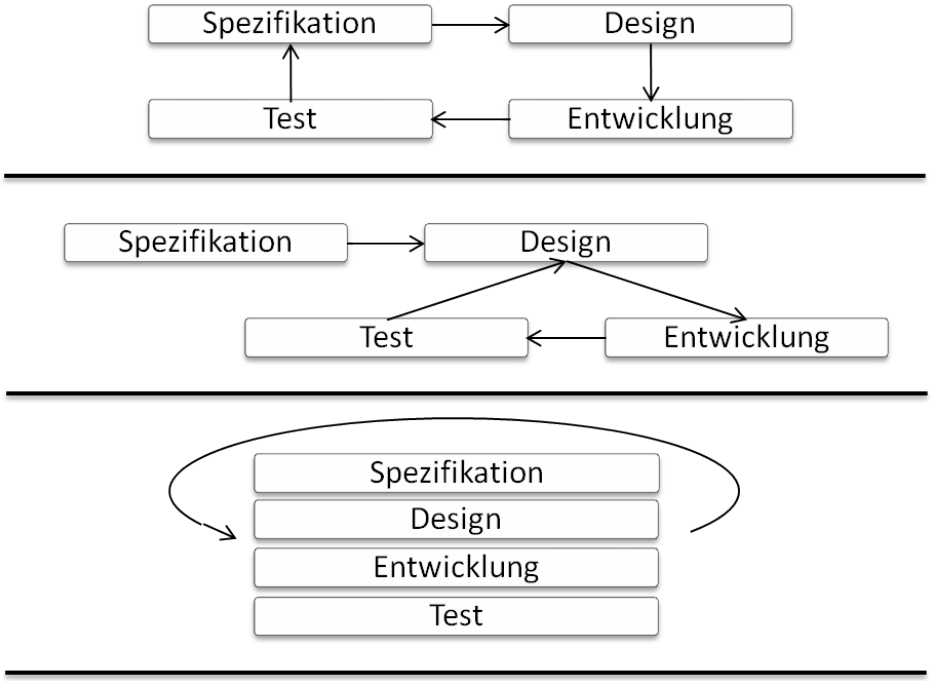
Quelle: [Sei12, vgl. S.22]

Abbildung 2.3: Verschiedene Ausprägungen klassischer Entwicklungsmodelle

nicht in Relation zu den erhöhten Kosten die bei der Erstellung der Testfälle anfallen würden. Bei dieser Entscheidung ist allerdings zu beachten, dass Software meist mit dem Ende eines Projektes nicht seinen finalen Stand erreicht hat. Fehler so wie geänderte Anforderungen führen meist dazu, dass sich Softwareprodukte ständig Weiterentwickeln. Diese Weiterentwicklung ist zwangsläufig mit Codeänderungen verbunden die wiederum zu Fehlern in bereits bestehenden Code führen kann. Um solche Fehler zu entdecken müssen im Rahmen von Regressionstests auch Testfälle wiederholt werden die bereits erfolgreich abgeschlossen wurden. Solche Regressionstests lassen sich bei einer vorhandenen Testautomatisierung besonders leicht durchführen. Sind also in der Software nach Projektabschluss größere Änderungen zu erwarten kann sich eine Automatisierung über längere Sicht durchaus lohnen. Neben Regressionstests kann auch die Notwendigkeit einer höheren Testtiefe oder einer breiteren Testabdeckung ein Faktor sein sich für eine Automatisierung zu entscheiden. In manchen Fällen, wie beispielsweise Lasttests mit mehreren hundert Usern, kann eine Automatisierung auch unabdingbar werden.

2.5.2 Iterative und agile Entwicklungsmodelle

Im Gegensatz zu den klassischen Entwicklungsmodellen sin in iterativen Modellen Rücksprünge in vorangegangene Phasen explizit erlaubt. Eine oder alle Phasen werden in diesen Modellen wiederholt durchlaufen. Auf diese weise kann das Softwareprodukt inkrementell wachsen. Durch ein derartiges Vorgehen ist es einfacher möglich auf den Umstand zu reagieren, dass sich Anforderungen in Softwareprojekten häufig ändern. Auch agile Vorgehensmodelle leben von solch einem iterativen Vorgehen. Die einzelnen Phasen werden in kleinen Zyklen viele male durchlaufen. Ein bekannter Vertreter der agilen Methoden ist Scrum [Sch02]. In Scrum wird ein Softwareprodukt in kurzen sogenannten Sprints realisiert. Innerhalb eines solchen Sprints wählt das Team selbständig eine Teilaufgabe des Projekts aus. Diese Teilaufgabe wird spezifiziert, designet, entwickelt und getestet. Am Ende eines Sprints steht ein Softwareprodukt, welches um ein weiteren Baustein ergänzt wurde. Der Sprint ist das zentrale Element dieses Prozessmodelles und kennzeichnet eine Iteration. Für das Tes-



Quelle: [Sei12, vgl. S.24]

Abbildung 2.4: Verschiedene Ausprägungen iterativer und agiler Entwicklungsmodelle

ten stellen diese kurzen Iterationen ein Problem dar. Jeder Entwicklungszyklus bringt neue Features hervor die mit Testfällen abgedeckt werden müssen. Der Agile Charakter in diesen Vorgehensmodellen bedingt, dass sich Anforderungen ständig ändern und somit auch bereits fertiger Code oft angepasst werden muss. Darüber hinaus ist nicht ausgeschlossen, dass

neue Features Auswirkungen auf alten Code haben können. Neben den neu implementierten Teilen muss daher zum Ende einer jeden Iteration auch sämtlicher alter Code getestet werden. Das bedingt einen enormen Testaufwand am Ende einer jeden Iteration. In agilen Vorgehensmodellen wie Scrum ist der Testaufwand nach wenigen Sprints bereits so hoch, dass ein Testdurchlauf zusammen mit allen Regressionstests nicht mehr zu bewältigen ist. Gerade in Projekten, die einem derartigen Vorgehensmodell folgen, ist es daher sinnvoll Testautomatisierung einzusetzen. Einmal implementierte Testfälle können zum Ende einer jeden Iteration erneut ausgeführt werden. Die höheren Kosten, die bei der Automatisierung entstehen sind so schnell amortisiert. Mit Hilfe von Ansätzen wie Test-driven development versucht man die hohen Anforderungen an die Testautomatisierung in agilen Vorgehensmodellen gerecht zu werden. Automatisierte Testfälle werden dazu bereits vor den zu testenden Komponenten implementiert. So kann eine ausreichend hohe Testabdeckung gewährleistet werden. Das sich ständig ändernde Testobjekt bedingt nicht nur die Notwendigkeit von automatisierten Testfällen, es erhöht gleichzeitig auch die Anforderungen an die Qualität der Testfälle. Häufige Änderungen am zu testenden Code lassen einmal implementierte Testfälle schnell veralten. Es muss daher bei der Erstellung der automatisierten Tests besonders auf die Wartbarkeit geachtet werden. Testfälle sollten möglichst Robust gewählt werden und nicht schon durch kleine Änderungen am Testobjekt zerstört werden. Änderungen am Testobjekt sind in agilen Projekten unvermeidbar. Unter diesem Gesichtspunkt sollte daher auch das Design der Testfälle erfolgen. Automatisierte Testfälle sollten ähnliche Qualitätsstandards wie der Code des eigentlichen Projektes verfolgen. Anpassungen an den Testfällen werden sonst schnell zu zeitaufwändig. Die Pflege der bereits implementierten Testfälle wird dann nicht mehr tragbar und die Akzeptanz der Tests im Projekt sinkt.

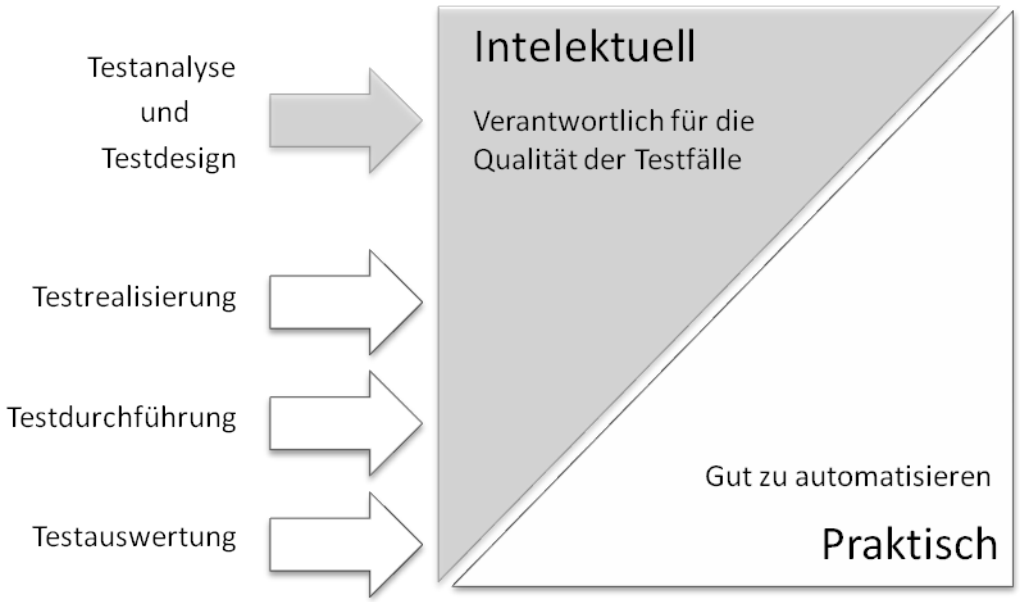
3 Testautomatisierung

In Kapitel 2.3 wurde der Begriff Testautomatisierung bereits eingeführt. Die darin gewählte Definition hat gezeigt, dass man unter Testautomatisierung weit mehr versteht, als das automatisierte ausführen von Testfällen auch wenn das der wohl am weitesten verbreiteten Bereich der Automatisierung ist. Testautomatisierung ist in allen Bereichen des Entwicklungs- bzw. Testprozesses möglich. „Das Spektrum umfasst alle Tätigkeiten zur überprüfung der Softwarequalität im Entwicklungsprozess, in den unterschiedlichen Entwicklungsphasen und Teststufen so wie die entsprechenden Aktivitäten von Entwicklern, Testern, Analytikern oder auch der in die Entwicklung eingebundenen Anwender. Die Grenzen der Automatisierung liegen darin, dass diese nur die manuellen Tätigkeiten eines Testers übernehmen kann, nicht aber die intellektuelle, krative und intuitive Dimension dieser Rolle.“ [Sei12, S.7] Diese intellektuelle Dimension ist vor allem in den frühen Phasen des Testprozesses gefordert. Diese Phasen sind maßgeblich für die Spätere Qualität der einzelnen Testfälle. Testautomatisierung wird daher nie die arbeiten eines guten Testanalysten voll ersetzen können. Um so weiter der Testprozess voranschreitet, um so p raktischer werden auch die zu erledigenden Aufgaben. Das Potential für eine Automatisierung steigt. Fewster et al. stellen diesen Zusammenhang in einer Grafik bildlich dar. [FG99, vgl. S.18] Abbildung 3.1 greift diese Darstellung auf und passt sie auf den in Kapitel 2.4 vorgestellten Testprozess an. Die verschiedenen Möglichkeiten der Testautomatisierung werden in Kapitel 3.2 geklärt. Zunächst soll jedoch die Frage beantwortet werden warum eine Automatisierung von Testfällen überhaupt Sinn macht.

3.1 Warum Testautomatisierung

Richtig durchgeführt kann Testautomatisierung eine Reihe von Vorteilen bringen. Dustin et al. stellen drei bedeutende Vorteile der Testautomatisierung fest. [Dus01, S.44 ff.]

1. Erstellung eines zuverlässigen Systems
2. Verbesserung der Testqualität und Testtiefe
3. Verringerung des Testaufwands und Reduzierung des Zeitplans



Quelle: [FG99, vgl. S.18]

Abbildung 3.1: Grenzen und Möglichkeiten der Testautomatisierung

Diese Vorteile stehen in enger Abhängigkeit zueinander. Eine Verringerung des Testaufwands für einzelne Tests schafft mehr Zeit die in bessere und breiter angelegte Tests investiert werden kann. Neben der direkten Beeinflussung führt Testautomatisierung also zusätzlich auch indirekt zu einer höhere Testqualität und Testtiefe. Diese bedingt dann wiederum dass mehr Fehler im System aufgedeckt werde können. So kann eine höhere Qualität des Endproduktes erreicht werden die sich in einem zuverlässigeren System zeigt.
In der Literatur gibt es zahlreiche Listen von Vorteilen der Testautomatisierung die bereits sehr fein gegliedert sind. [FG99] [Tha02] Gleicht man diese Vorteile mit den von Dustin et al. gewählten Oberpunkten ab zeigt sich, dass vor allem die Punkte 2 und 3 also Qualität und Effezienz gut durch diese repräsentiert werden. Sie lassen sich leicht mit den Vorteilen anderer Autoren unterfüttern. Die Erstellung eines zuverlässigen Systems ist hingegen nur schwer direkt zu beeinflussen. In der Regel wird dieser Punkt wie oben beschrieben indirekt beeinflusst. Im folgenden wird daher dieser Punkt nicht weiter betrachtet. Die feiner gegliederten Vorteile wie sie Fewster et al. beschreiben werden Verbesserung der Testqualität und Testtiefe so wie der Verringerung des Testaufwands und Reduzierung des Zeitplans zugeordnet.
Während das zuverlässige System eher eine Folge aus höherer Effizienz und Qualität im Testen sind, können der Qualitätsgewinn und die Effizienzsteigerung durch feinere Vorteile unterfüttert werden.

3.2 Möglichkeiten der Testautomatisierung im Testprozess

Einteilung nach testprozess nicht perfekt. Dahre eine einteilung die der automatisierung besser passt: A Search-Based Approach for Cost-Effective Software Test Automation Decision Support and an Industrial Case Study

3.2.1 Testdesign

3.2.2 Testcodeerstellung

c and r

3.2.3 Testdurchführung

3.2.4 Testauswertung

3.3 Schnittstellen der Testautomatisierung zum System

Jeder testfall muss in irgendeiner weise mit dem zu testenden objekt interagieren. Analog zu manuellen tests gibt es hierfür verschiedene möglichkeiten.

3.3.1 API

3.3.1.1 JUnit

3.3.2 GUI

Web

4 Testautomatisierung mit Selenium

4.1 Selenium

4.2 Testdurchführung mit Selenium

4.3 Testcodeerstellung mit Selenium

4.3.1 Recorde-and-playback

4.3.1.1 Vorteile von Recorde-and-playback

4.3.1.2 Probleme von Recorde-and-playback

4.3.2 Manuell

4.3.3 Page Object Pattern

4.3.3.1 Vorteile des Page Object Pattern

4.3.3.2 Probleme des Page Object Pattern

5 Teilautomatisierte Generierung von Page Objects

5.1 übersicht über die Idee

5.2 einordnung des Testharness und gui in die Geamtstruktur (Deploymentdiagramm)

5.3 übersicht über Aufbau des Systems

5.3.1 pro modul ein kapitel

5.4 Vorteile und Probleme

5.5 Anwendung

Abbildungsverzeichnis

Tabellenverzeichnis

Literaturverzeichnis

[Dus01] Dustin, Elfriede: *Software automatisch testen*. Berlin [u.a.] : Springer, 2001 (Xpert.press). `http://fhmoz3.bib-bvb.de/InfoGuideClient.fhmsis/start.do?Login=wofhmze&Query=540="3-540-67639-2"`. – ISBN 978–3–540–67639–3

[FG99] Fewster, Mark ; Graham, Dorothy: *Software Test Automation Effective use of test execution tools*. Addison-Wesley, 1999. – ISBN 0–201–33140–3

[Har00] Harrold, Mary J.: Testing: A Roadmap. In: *Proceedings of the Conference on The Future of Software Engineering*. New York, NY, USA : ACM, 2000 (ICSE '00). – ISBN 1–58113–253–0, 61–72

[Hof13] Hoffmann, Dirk W.: *Software-Qualität*. 2013. Berlin : Springer, 2013. – ISBN 978–3–540–76322–2

[IEE91] IEEE: IEEE Standard Computer Dictionary: A Compilation of IEEE Standard Computer Glossaries. In: *IEEE Std 610* (1991), Januar, S. 1–217. `http://dx.doi.org/10.1109/IEEESTD.1991.106963`. – DOI 10.1109/IEEESTD.1991.106963

[IEE08] IEEE: *IEEE Std 829-2008 IEEE Standard for Software and System Test Documentation*. Juli 2008

[ISO01] ISO/IEC: *ISO/IEC 9126. Software engineering – Product quality*. 2001. ISO/IEC, 2001

[Pro06] Prof. Dr. R. Lindermeier: *Projekt- und Qualitätsmanagement Softwarequalität und Softwareprüfung*. 2006

[Roy87] Royce, W. W.: Managing the Development of Large Software Systems: Concepts and Techniques. In: *Proceedings of the 9th International Conference on Software Engineering*. Los Alamitos, CA, USA : IEEE Computer Society Press, 1987 (ICSE '87). – ISBN 978–0–89791–216–7, 328–338

[RW06] Ramler, Rudolf ; Wolfmaier, Klaus: Economic Perspectives in Test Automation: Balancing Automated and Manual Testing with Opportunity Cost. In: *Proceedings of the 2006 International Workshop on Automation of Software Test*. New York, NY, USA : ACM, 2006 (AST '06). – ISBN 1–59593–408–1, 85–91

[Sch02] Schwaber, Ken: *Agile software development with Scrum*. Pearson Internat. Ed. Upper Saddle River, NJ : Pearson Education International, 2002 (Series in agile software development). `http://fhmoz3.bib-bvb.de/InfoGuideClient.fhmsis/start.do?Login=wofhmze&Query=540="0-13-207489-3"`. – ISBN 978–0–13–207489–6

[Sei12] Seidl, Richard: *Basiswissen Testautomatisierung / Richard Seidl ; Manfred Baumgartner ; Thomas Bucsics*. 1. Aufl. Heidelberg : dpunkt-Verl., 2012. – ISBN 978–3–89864–724–3

[SL07] Spillner, Andreas ; Linz, Tilo: *Basiswissen Softwaretest*. 3. Aufl. Heidelberg : dpunkt-Verl., 2007. – ISBN 3–89864–358–1

[Tha02] Thaller, Georg E.: *Software-Test*. 2., aktualisierte und erw. Aufl. Hannover : Heise, 2002. – ISBN 3–88229–198–2