Testautomatisierung

- mit Selenium -

Fakultät für Informatik und Mathematik
der Hochschule München

Seminararbeit

vorgelegt von

Matthias Karl
Matrikel-Nr: 03280712

im Dezember 2014

Prüferin: Prof. Dr. Ulrike Hammerschall

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

Diese Arbeit befasst sich mit der Automatisierung von Testfällen. Die Arbeit gibt einen Überblick über die verschiedenen Bereiche und Möglichkeiten bei der Testautomatisierung und befasst sich im nähren mit dem Automatisierungstool Selenium. Zu Beginn werden zunächst die allgemeinen Grundlagen des Testens geklärt, um dann die verschiedenen Bereiche und Möglichkeiten der Testautomatisierung aufzuzeigen. Auf den Bereich der automatisierten UI-Tests wird im Kontext von Selenium näher eingegangen.

Inhaltsverzeichnis

1 Einleitung

Software hat in der heutigen Zeit eine hohe Verbreitung gefunden. Softwaresysteme werden immer wichtiger sowohl für Unternehmen als auch für jeden Einzelnen persönlich. Die Anforderungen an moderne Software steigen ständig. Die Systeme werden immer größer und komplexer. Das hat zur Folge, dass auch die Anforderungen an die Qualität der Software immer weiter steigen und wichtiger werden. Fehler in Anwendungen verursachen immer wieder einen hohen finanziellen Schaden und können im schlimmsten Fall sogar Menschenleben kosten. Diese Probleme werden immer gravierender, wenn mit der Wichtigkeit, Komplexität und Größe von Applikationen nicht auch gleichzeitig die Qualität steigt. [Bur03]
Softwaretests sind ein weit verbreitetes Mittel um die Qualität einer Software zu überprüfen und sicherzustellen. Komplexere und umfangreichere Software bedeutet daher auch gleichzeitig einen steigenden Testaufwand.
Qualitätssicherungsmaßnahmen, wie z.B. das Testen, machen jetzt schon einen Großteil der Kosten in Softwareprojekten aus. Studien haben gezeigt, dass das Testen für 50% und mehr der gesamten Projektkosten verantwortlich ist. [RW06] Es ist daher nicht verwunderlich, dass es immer wieder Versuche gibt diese Kosten zu reduzieren, ohne dabei den angestrebten Qualitätsstandart zu reduzieren. Einer der Wege die dafür vorgeschlagen wurden, ist das automatisieren von Testfällen. [Har00] Im laufe der Jahre hat die Teatautomatisierung immer mehr an Bedeutung gewonnen. Heute ist sie bereits fester Bestandteil von Entwicklungskonzepten wie Continous Delivery und Continous Integration.
Diese Arbeit befasst sich daher mit der Automatisierung von Testfällen. Die Ausarbeitung soll einen Überblick über die verschiedenen Bereiche der Testautomatisierung geben, um dann den Bereich der Automatisierten UI-Tests mit Selenium näher zu beleuchten. Zu Beginn werden zunächst die allgemeinen Grundlagen des Testens erläutert, um dann den Begriff der Testautomatisierung näher einzugehen. Nachdem die verschiedenen Bereiche der Testautomatisierung aufgezeigt wurden, wird der Fokus auf die Automatisierung von UI-Tests mit Hilfe von Selenium gelegt.

2 Grundlagen

2.1 Softwarequalität

Nach der Norm ISO-25000:2014 4.33 bezeichnet der Begriff der Softwarequalität die Fähigkeit einer Software, die expliziten und impliziten Bedürftnisse von Benutzern, unter den Bedingungen unter der sie benutzt wird, zu befriedigen. [Int14] Softwarequalität hat nach dieser Definition einen subjektiven Charakter. Dieser subjektive Charakter macht den Begriff in der Praxis schwer zu greifen und damit nicht direkt anwendbar. Aus diesem Grund existieren sogenannte Qualitätsmodelle, die Softwarequalität messbar und damit auch überprüfbar machen sollen. Ein solches Qualitätsmodell wird zum Beispiel in der ISO-Norm 9126 vorgestellt. [ISO01] Es werden verschiedene Qualitätsmerkmale definiert, die zur Beurteilung der Gesamtqualität eines Softwareprodukts dienen. Hierunter fallen die Merkmale:

- Funktionalität
- Zuverlässigkeit
- Benutzbarkeit
- Effizienz
- Änderbarkeit
- Übertragbarkeit

Es existieren verschiedene Methoden um sicherzustellen, dass Software, bezogen auf die Qualitätsmerkmale gewissen Anforderungen genügt. Ein Teil der Methoden geht dabei davon aus, dass ein qualitativ hochwertiger Prozess der Produkterstellung die Entstehung von qualitativ hochwertigen Produkten begünstigt. Das Augenmerk wird hierbei also auf die Prozessqualität gelegt. Allgemein fasst man diese Gruppe unter dem Begriff prozessorientiertes Qualitätsmanagement zusammen. Die klassischen Vorgehensmodelle der Softwarentwicklung werden z.B. hier eingeordnet. Worauf sich diese Arbeit jedoch konzentrieren möchte, sind die

Methoden des produktorientierten Qualitätsmanagement [Pro06]. Hierbei wird das Softwareprodukt direkt bezüglich der Qualitätsmerkmale überprüft. Darunter fallen beispielsweise Softwaretests.

2.2 Softwaretest

Das produktorientierte Qualitätsmanagement unterteilt sich weiter in die Bereiche des konstruktiven und analytischen Qualitätsmanagement. Unter dem konstruktiven Qualitätsmanagement versteht man den Einsatz von z.B. Methoden, Werkzeugen oder Standards die dafür sorgen, dass ein (Zwischen-)Produkt bestimmte Forderungen erfüllt. Was man im Allgemeinen aber unter einem Softwaretest versteht, ist im Bereich der prüfenden Verfahren des analytischen Qualitätsmanagement angesiedelt. Unter analytischen Qualitätsmanagement versteht man den Einsatz von analysierenden bzw. prüfenden Verfahren, die Aussagen über die Qualität eines (Zwischen-)Produkts machen. [Pro06]
Die Norm ISO/IEC/IEEE 24765:2010 3.280 definiert das testen von Software als eine dynamische Verifikation des Verhaltens eines Programms, gegen ein erwartetes Verhalten, bei einer endlichen Auswahl an Testfällen. (Im Original: „the dynamic verification of the behavior of a program on a finite set of test cases, suitably selected from the usually infinite executions domain, against the expected behavior.“ [ISO10]) Aufgabe eines Softwaretests ist es dabei nicht, einen Fehler im Code zu Lokalisieren und zu beheben. Das lokalisieren und Beheben des Defekts ist Aufgabe des Softwareentwicklers und wird auch als Debugging (Fehlerbereinigung, Fehlerkorrektur) bezeichnet. Während Debugging das Ziel hat, Defekte bzw. Fehlerzustände zu beheben, ist es Aufgabe des Tests, Fehlerwirkungen (die auf Defekte hinweisen) gezielt und systematisch aufzudecken. [SL07] Dabei dienen definierte Anforderungen als Prüfreferenz, mittels derer ggf. vorhandene Fehler aufgedeckt werden. „Das Testen von Software dient durch die Identifizierung von Defekten und deren anschließenden Beseitigung durch das Debugging zur Steigerung der Softwarequalität.“ [SL07] Als möglicher Rahmen für die Anforderungen können z.B. die in 2.1 bereits beschriebenen Qualitätsmerkmale dienen.

2.3 Testprozess

Der Begriff des Softwaretests, wie er in 2.2 beschrieben ist, erfordert eine Einordnung in einen größeren Zusammenhang. Ein Softwaretest steht in der Regel nicht für sich alleine, sondern ist Teil eines größeren Prozesses, der den Softwaretest in seinem gesamten Lebenszyklus begleitet. Durch den Testprozess wird die Aufgabe des Testens in kleinere Testaufga-

ben gegliedert. Splinner und Linz fassen diese Testaufgaben im fundamentalen Testprozess zusammen [SL07]
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. Diese Teilaufgaben werden im folgenden kurz näher beschrieben. Als Grundlage dient hierfür die Beschreibung des fundamentalen Testprozesses nach Splinner und Linz. [SL07, S.20ff]

2.3.1 Testplanung und Steuerung

Das Testen von Software stellt eine umfangreiche Aufgabe dar. Um diese zu beweltigen wird eine sorgfältig Planung benötigt. Mit der Planung des Testprozesses wird am Anfang des Softwareentwicklungsprojekts begonnen. Ziel ist es dabei, die Rahmenbedingungen für die Testaktivitäten festzulegen. Nachdem Aufgaben und die Zielsetzung der Tests bestimmt wurden, können die Resourcen die für die Durchführung der Aufgaben benötigt werden geplant werden. Eine weitere Kernaufgabe der Planung ist das Festlegen einer Teststrategie. Da ein vollständiger Test einer Anwendung in der Regel nicht möglich ist, müssen die zu testenden Einheiten nach schwere der Fehlerwirkung priorisiert werden. Je nach schwere der zu erwarteten Auswirkungen, kann dann die Intensität bestimmt werden, mit der ein einzelner Systemteil getestet werden soll. Ziel der Teststrategie ist es also, eine optimale Verteilung der Tests auf die gesamte Software zu erreichen. Dabei sind auch geeignete Testendekriterien festzulegen, um zu entscheiden, ob ein Testprozess abgeschlossen werden kann. Ein weiterer Punkt, der in der Planungsphase berücksichtigt werden muss, ist die Beschaffung von geeigneten Werkzeugen die zur Durchführung und Erstellung der Testfälle benötigt werden. Die in der Planung erarbeiteten Ergebnisse werden in einem Testkonzept festgehalten. Eine mögliche Gliederung bietet z.B. die internationale Norm IEEE 829-2008. [IEE08] Parallel zu den Testaktivitäten muss über den gesamten Testprozess eine Steuerung erfolgen. Der Fortschritt der Tests und des Projekts wird dabei laufend erhoben, geprüft und bewertet.

2.3.2 Testanalyse und Testdesign

In dieser Phase wird die Testbasis überprüft, also die zugrunde liegenden Dokumente, die für die Erstellung der Testfälle benötigt werden. Spezifikationen und Anforderungen müssen vollständig, konsistent und überprüfbar vorliegen. Anhand der in der Planung festgelegten Teststrategie und der Testbasis können nun Testfälle erstellt werden. Die Spezifikation der Testfälle erfolgt dabei in zwei Stufen. Testfälle werden in dieser Phase zunächst recht allgemein definiert. Diese allgemeinen Testfälle können dann später mit tatsächlichen Eingabewerten konkretisiert werden. Zu der Spezifikation eines Testfalls gehören auch etwaige Rand- und Vorbedingungen sowie ein erwartetes Ergebnis. Um letzteres bestimmen zu können, muss ein so genanntes Testorakel befragt werden. Hierbei handelt es sich um eine Quelle, die auf das erwartete Ergebnis schließen lässt. Ein mögliches Testorakel wäre beispielsweise die Spezifikation der Software.

2.3.3 Testrealisierung und Testdurchführung

In diesem Schritt des Testprozesses werden aus den allgemein gehaltenen Testfällen konkrete Testfälle gebildet. Diese Testfälle können dann zusammen mit der Testinfrastruktur im Detail realisiert werden. Logisch zusammengehörige Testfälle werden dabei in Testszenarien gruppiert. Dabei ist die in der Planung festgelegte Priorität der Testfälle zu berücksichtigen. Wenn das Testobjekt zur Verfügung steht, beginnt die Abarbeitung der Testfälle. Jede Durchführung und deren Ergebnisse werden protokolliert. Auf mögliche Abweichungen von den erwarteten Ergebnissen muss entsprechend reagiert werden. Nach der Korrektur des Fehlers ist zu überprüfen, ob der Fehler wirklich beseitigt wurde und bei der Beseitigung keine weiteren Fehlerzustände hinzugekommen sind.

2.3.4 Testauswertung und Bericht

In dieser Phase des Testprozesses wird überprüft, ob die in der Planung festgelegten Testendkriterien erfüllt sind. Dabei kann es zu unterschiedlichen Ergebnissen kommen. Sind ein oder mehrere Kriterien nicht erfüllt, müssen eventuell weitere Tests spezifiziert und durchgeführt werden. Ein Ergebnis kann jedoch auch sein, dass der Aufwand zum Erfüllen der Testendekriterien nicht in einem angemessenem Verhältnis zum Aufwand steht. In diesem Fall kann auch auf weitere Tests verzichtet werden. Bei dieser Entscheidung muss jedoch das damit verbundene Risiko berücksichtigt werden. Ist das Testende erreicht, ist ein zusammenfassender Bericht an die Entscheidungsträger zu erstellen. Je nach dem wie kritisch die

betrachteten Tests sind, kann dieser Bericht mehr oder weniger formal ausfallen.

2.3.5 Abschluss der Testaktivitäten

Am Ende des Testprozesses steht ein kritischer Rückblick auf die durchgeführten Tätigkeiten. Die gemachten Erfahrungen müssen analysiert werden. Probleme, die in diesem Testprozess aufgetreten sind, können so in folgenden Projekten vermieden werden. Auf diese Weise kann der Testprozess ständig verbessert werden. Um im Zeitraum der Wartung die Testfälle erneut durchführen zu können, sollten die verwendeten Tools so wie die eingesetzten Testsysteme und Testrahmen konserviert werden.

2.4 Softwarelebenszyklus

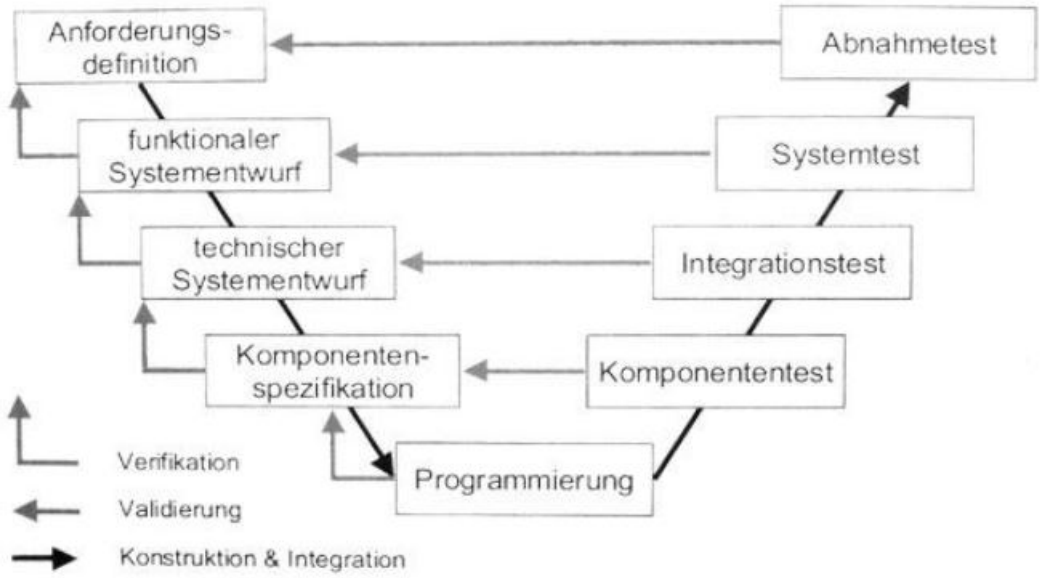
Der Testprozess und seine Aktivitäten sind nur ein Teil des Gesamtprojekts und bilden mit dem Anforderungsmanagement, der Designphase und der Entwicklung den Kern des Projekts. Dazu kommen noch unterstützende Prozesse, wie Projekt-, Release oder Konfigurationsmanagement. Die zeitliche und inhaltliche Ausgestaltung dieser Phasen ergibt ein Softwareentwicklungsmodell bzw. ein Projektvorgehen. [Sei12] Aus Sicht des Testens spielt hier das allgemeine V-Modell eine besondere Rolle.

2.4.1 V-Modell

Die Grundidee des allgemeinen V-Modells ist, dass Entwicklungsarbeiten und Testarbeiten zueinander korrespondierende, gleichberechtigte Tätigkeiten sind. Bildlich dargestellt, wird dies durch die zwei Äste eines „V“. Der linke Ast steht für die immer detaillierter werdenden Entwicklungstätigkeiten, mit denen das System Schritt für Schritt realisiert wird. Der rechte Ast steht für Integrations- und Testartbeiten, in deren Verlauf elementare Programmbausteine sukzessive zu größeren Teilsystemen zusammengesetzt (integriert) und jeweils auf die richtige Funktion gegen die korrespondierenden Spezifikation des linken Astes geprüft werden. [SL07]
Das V-Modell wird in zahlreichen unterschiedlichen Versionen dargestellt. Je nach Literaturquelle und Interpretation des Anwenders variieren Benennung und Anzahl der Phasen.

Ein bildliches Beispiel für ein allgemeines V-Modell wird in Abbildung 2.1 dargestellt.

Die Aktivitäten des linke Astes lassen sich wie folgt beschreiben: [SL07]



Quelle: [SL07]

Abbildung 2.1: Allgemeines V-Modell

- **Anforderungsdefinition:** Die Wünsche und Anforderung des Auftraggebers oder späteren Systemanwenders werden gesammelt, spezifiziert und verabschiedet. Zweck und gewünschte Leistungsmerkmale des zu erstellenden Softwaresystems liegen damit fest.
- **Funktionaler Systementwurf:** Die Anforderungen werden auf Funktionen und Dialoge des neuen Systems abgebildet.
- **Technischer Systementwurf:** Die technische Realisierung des Systems wird entworfen. Hierzu gehören u.a.: Definition der Schnittstellen zur Systemumwelt und die Zerlegung des Systems in überschaubare Teilsysteme, die möglichst unabhängig voneinander entwickelt werden können.
- **Komponentenspezifikation:** Für jedes Teilsystem werden Aufgaben, Verhalten, innerer Aufbau und Schnittstellen zu anderen Teilsystemen definiert.
- **Programmierung:** Programmierung jedes spezifizierten Bausteins in einer Programmiersprache

Der rechte Ast beschreibt zu den oben aufgeführten konstruktiven Aktivitäten eine korrespondierende Teststufe: [SL07]

- **Komponententest:** Prüft, ob jeder einzelne Softwarebaustein für sich die Vorgaben seiner Spezifikation erfüllt.
- **Integrationstest:** Prüft, ob Gruppen von Komponenten wie im technischen Systementwurf vorgesehen zusammenspielen.
- **Systemtest:** Prüft, ob das System als Ganzes die spezifizierten Anforderungen erfüllt.
- **Abnahmetest:** Prüft, ob das System aus Kundensicht die vertraglich vereinbarten Leistungsmerkmale aufweist.

3 Testautomatisierung

Nach Seidl et al. versteht man unter dem Begriff der Testautomatisierung „die Durchführung von ansonsten manuellen Testtätigkeiten durch Automaten.“ [Sei12, Seite 7] Das Spektrum der Testautomatisierung umfasst alle Tätigkeiten, die dazu dienen, die Qualität einer Software zu überprüfen. Darunter fallen alle Aufgaben, die im Testprozess in den einzelnen Phasen der Softwareentwicklung anfallen. Testautomatisierung ist also nicht nur auf die automatisierte Testdurchführung beschränkt, sondern kann ebenso bei der Testfallerstellung, der Testdatengenerierung, der Testauswertung oder auch der Testumgebungsherstellung und -Wiederherstellung eine Rolle spielen. „Die Grenze der Automatisierung liegt [nach Seidl et al.] darin, dass diese nur die manuellen Tätigkeiten eines Testers übernehmen kann, nicht aber die intellektuelle, kreative und intuitive Dimension dieser Rolle.“ [Sei12, Seite 7] Besonders lohnenswert ist eine Automatisierung bei sich wiederholenden Aufgaben, zu denen besonders die Testdurchführung zählt. Die Automatisierung in diesem Bereich ist bereits weit verbreitet und soll auch den Fokus dieser Arbeit bilden. Die anderen Bereiche der Testautomatisierung sollen nur kurz aufgezeigt werden.

3.1 Warum Testautomatisierung

Studien haben gezeigt, dass das Testen für 50% und mehr der gesamten Projektkosten verantwortlich ist. [RW06] Es gab daher immer wieder Versuche den Bereich des Testens zu optimieren. Testautomatisierung ist ein weit verbreiteter Ansatz die Kosten von manuellen Tests zu reduzieren und dabei die Qualität des Softwaresystems zu sichern. [AGIS14] Man verspricht sich dabei eine Reihe von Vorteilen gegenüber der manuellen Ausführung von Testfällen. Fewster und Graham haben dazu eine Liste aufgestellt, die in Tabelle 3.1 verkürzt dargestellt ist. Ähnliche Vorteile werden auch von Thalle [Tha02, Seite 228] beschrieben.
Die meisten der aufgelisteten Vorteile können mit den Worten, Effizienz und Wiederverwertbarkeit, zusammengefasst werden. Die Testautomatisierung entfaltet ihr volles Potential immer dann wenn Testfälle nicht nur einmal, sondern wiederholt ausgeführt werden. Mittels Automatisierung können Testfälle wiederholt durchlaufen werden ohne dabei einen großen

Führe existierende Regressionstests für eine neue Version eines Programms aus.	Die Möglichkeit bereits erstellte Testfälle ohne Mehraufwand auszuführen macht das Testen effektiver.
Führe mehr Tests öfter aus .	Automatisierung bedeutet schnellere Testausführung. Dadurch lassen sich mehr Testdurchläufe bewerkstelligen. Automatisierung sollte auch das erstellen neuer Testfälle einfacher und schneller machen.
Führe Tests durch, die manuell schwer bis unmöglich wären.	Performancetests sind beispielsweise ohne Automatisierung fast nicht zu bewältigen.
Bessere Verwendung von Resourcen.	Die Automatisierung von sich wiederholenden Aufgaben ermöglicht den Testern die Arbeit an anderen Aufgaben.
Wiederholbarkeit und Konsistenz von Testfällen.	Tests werden immer gleich ausgeführt. Auf diese weise können die Testergebnisse besser verglichen werden.
Wiederverwendbarkeit von Tests.	Vor allem neue Projekte werden durch die Wiederverwendbarkeit von Testfällen beschleunigt.
Frühere Markteinführung.	Das Wiederverwenden und beschleunigen von Testfällen beschläunigt den gesamten Testprozess. Das verkürzt letztendlich auch die Zeit bis zur Markteinführung.
Verbessertes Vertrauen.	Viele Testfälle, die oft und konstant ausgeführt werden können, erhöhen das Vertrauen in die Software und seine Marktreife.

Tabelle 3.1: Vorteile der Testautomatisierung nach Fewster und Graham [FG99]

Mehraufwand zu erzeugen. Regressionstests eignen sich daher beispielsweise besonders gut für eine Automatisierung. Tester können von sich wiederholenden Tätigkeiten entlastet werden und sich anderen Aufgaben widmen. Andere Bereiche, wie zum Beispiel Stresstests, wären ohne Automatisierung schwer bis gar nicht zu realisieren. Ein Stresstest mit ca. 200 gleichzeitigen Benutzern ist auf manuelle Weise nicht umsetzbar. Die Eingaben von 200 Benutzern können hingegen recht einfach, mittels automatisierten Tests, simuliert werden. Mit Hilfe der Testautomatisierung ist es daher möglich die Zeit, die das Testen benötigt zu verringern und dabei die Softwarequalität zu erhöhen.
Die genannten Vorteile lassen die Testautomatisierung sehr attraktiv erscheinen. Diese Versprechen sind in der Praxis jedoch nicht leicht zu erreichen. Wird die Automatisierung nicht gut umgesetzt, kann sie schnell zu einer größeren Belastung werden, als dass sie einen Nutzen bringt. Fewster und Graham haben auch hierzu eine Liste mit bekannten Problemen zusammengestellt, [FG99] die in Tabelle 3.2 zusammengefasst ist. Das Hauptproblem ist dabei meist, dass der Aufwand der Testautomatisierung unterschätzt wird. In jedem etwas größeren Softwarevorhaben muss die Testautomatisierung als eigenes Projekt gesehen werden. Jedes Projekt erfordert eine genaue Planung und muss gewissen Prozessen folgen. Wird diese Planung vernachlässigt oder die Prozesse missachtet, kann ein Testautomatisierungsprojekt schnell in die falsche Richtung laufen. Vor allem die Planung ist hier besonders wichtig. Nicht alles was automatisiert werden kann, sollte auch automatisiert werden. Amannejad et al. [AGIS14] widmen sich in einem Paper der Frage, welche Teile eines Testobjekts in einer automatisierten Art und Weise getestet werden sollten. Sie kommen zu eben diesem Ergebnis. Die Vorteile durch die Testautomatisierung überwiegen nicht immer und sind gegen den zu erwartenden Aufwende zu prüfen. Erst wenn die zu erwartenden Einsparungen die Kosten überwiegen, ist die Testautomatisierung sinnvoll.

3.2 Bereiche der Testautomatisierung

Die Vor- und Nachteile der Testautomatisierung, wie sie in Kapitel 3.1 beschreiben sind, beziehen sich hauptsächlich auf die automatisierte Durchführung von Testfällen. Hierauf soll auch der Fokus dieser Arbeit liegen. Wie aber bereits eingangs in Kapitel 3 erwähnt, erstreckt sich die Möglichkeit zur Testautomatisierung über den gesamten Bereich des Testprozesses. Eine dem Testprozess recht ähnliche, jedoch leicht andere Unterteilung für die Bereiche der Testautomatisierung, schlägt Amannejad et al. [AGIS14] vor. Er unterteilt die Testautomatisierung im Testprozess in vier Aufgaben:

Unrealistische Erwartungen.	Manager erwarten oft, dass die Testautomatisierung alle Probleme löst und sofort die Qualität der Software verbessert wird.
Schlechte Testpraxis.	Wenn die Testpraktiken bereits schlecht sind, ist es besser diese zunächst zu verbessern, bevor mit einer Automatisierung begonnen wird.
Erwartung, dass automatisierte Tests viele neue Fehler findet.	Wenn automatisierte Tests einmal erfolgreich ausgeführt wurden, ist es nicht sehr wahrscheinlich, dass sie in folgenden Testläufen noch viele weitere Fehler finden werden.
Falsche Vorstellung von Sicherheit.	Ein Testreport ohne Fehler bedeutet nicht, dass das Testobjekt fehlerlos ist.
Wartung.	Wenn das Testobjekt geändert wird, müssen meist auch die automatisierten Testfälle angepasst werden. Wenn die Wartung mehr Zeit verschlingt als durch die Automatisierung eingespart werden kann, ist der Nutzen fraglich.
Technische Probleme.	Testfälle zu automatisieren ist keine einfache Aufgabe. Es ist zu erwarten, dass dabei eine Reihe von Problemen gelöst werden müssen.
Organisatorische Fragen.	Eine erfolgreiche Testautomatisierung erfordert einen hohen Grad an technischer Kompetenz und Unterstützung des Managments. Testautomatisierung hat drüber hinaus Einfluss auf die Organisation und erfordert oft Änderungen in den etablierten Prozessen.

Tabelle 3.2: Nachteile der Testautomatisierung nach Fewster und Graham [FG99]

- Testdesign: Erstellen einer Liste von Testfällen um gewisse Akzeptanzkriterien zu prüfen.
- Testcodeerstellung: Erstellen von automatisiertem Testcode.
- Testdurchführung: Ausführen von Testfällen und aufzeichnen der Ergebnisse.
- Testauswertung: Auswerten der aufgezeichneten Testergebnisse.

3.2.1 Testdesign

Unter Testdesign versteht man das erstellen von Testfällen, um gewisse Akzeptanzkriterien zu prüfen. Darunter fällt auch das identifizieren von Testdaten wie z.B. mögliche Eingabewerte und erwartete Ergebnisse. Hierfür gibt es zahlreiche Ansätze die manuell, aber auch toolgestützt und damit automatisiert durchgeführt werden können. Unter dem Oberbegriff der Kombinatorik existieren einige Testfallentwurfsmethoden die vor allem darauf abzielen eine sinnvolle Auswahl an Eingabedaten zu ermitteln. Nach Seid et al. fallen darunter: [Sei12, Seite 27]

- Äquivalenzklassenbildung.
- Grenzwertanalyse.
- Klassifikationsbaummethode.

Fewster und Graham [FG99, Seite 19 ff.] beschreiben eine weitere Möglichkeit, bei der Tools direkt den Code und die Interfaces des zu testenden Systems verwenden, um Eingabedaten abzuleiten. Ein weiterer Weg zum automatisierten Design von Testfällen, den Seidel et al. aufzeigen, [Sei12, Seite 33] sind modellbasierte Techniken. In modellbasierten Techniken wird das zu testende System in einem hohen Detailgrad durch Modelle abgebildet. Mit Hilfe dieser Modelle ist es möglich Testfälle abzuleiten. Viele weitere Ansätze sind möglich. So beschreiben Last et al. beispielsweise einen Ansatz, der data mining verwendet. [LFK03] Memon et al. beschreiben einen weiteren Ansatz zum Erzeugen von Testfällen für grafische Benutzeroberflächen. [MPS99] Eine weitere Möglichkeit, die an den Anforderungen der Software ansetzt, wird von Tahat et al. beschrieben. [TVKB01]

3.2.2 Testcodeerstellung

Beim manuellen Testen versteht man unter der Testcodeerstellung das Dokumentieren von Testfällen, die in der vorangegangenen Phase gefunden wurden. Im Bereich der Testautomatisierung

versteht man darunter das Erzeugen von wirklichem Testcode. In der Regel handelt es sich bei der Testcodeerstellung um einen manuellen Prozess, bei dem ein Softwareentwickler die zuvor designeten Testfälle als Code entwickelt. Weit verbreitet sind in diesem Bereich aber auch halb automatisierte Methoden. Diese basieren auf sogenannten „recorde-and-playback“ Tools. Die Tools zeichnen die Interaktionen eines Benutzers mit der grafischen Oberfläche der zu testenden Anwendung auf und generieren daraus Testskripte. Im Anschluss können die aufgezeichneten Abläufe beliebig oft wiederholt werden. So erzeugte Skripte können dann beispielsweise angepasst und verwendet werden, um in einem daten getriebenen Ansatz eine ganze Reihe an Testfällen abzuarbeiten. Die in der Designphase durch kombinatorische Methoden erzeugten Eingabedaten können hierfür zum Beispiel als Datenbasis dienen. Neben halbautomatischen Methoden ist es auch möglich die Erstellung von Testcode voll zu automatisieren. Über modellbasierte Techniken ist es nicht nur möglich Testfalldesigns abzuleiten. Bei entsprechend hohen Detailgrad der Modelle können Testskripte auch voll automatisiert erzeugt werden. Bouquet et al. [BGLP08] beschreiben beispielsweise ein modellbasiertes Framework, welches das automatisierte designen, generieren, managen und ausführen von Testfällen unterstützt.

3.2.3 Testdurchführung

Unter Testdurchführung versteht man das Ausführen der zuvor erstellten Testskripte, so wie das aufzeichnen der Testergebnisse. Die Testausführung ist der Bereich, der nach Erfahrungen von Amannejad et al. [AGIS14] am engsten mit der Testautomatisierung verbunden wird. Ähnlich wie in den vorangegangenen Phasen ist auch hier ein manuelles, halb automatisiertes und vollautomatisiertes Vorgehen möglich. Für welches Vorgehen man sich entscheidet, hängt jedoch stark von den Entscheidungen in den vorangegangenen Phasen ab. Hat man sich in den vorangegangenen Phasen für ein manuelles Testen entschieden, ist es nicht mehr möglich die Ausführung automatisiert durchzuführen. Liegen die Testfälle jedoch bereits als automatische Testskripte vor, kann auch die Ausführung der Skripte automatisch erfolgen.

3.2.4 Testauswertung

Nachdem ein Testfall ausgeführt worden ist, muss das erhaltenen Ergebnis gegen einen erwarteten Wert geprüft werden. Diese Überprüfung kann manuell durch einen Tester erfolgen. In einer automatisierten Umgebung werden die erwarteten Ergebnisse jedoch meist fest in den Testcode geschrieben und dienen dort als feste Prüfpunkte. Um die erwarteten Ergebnisse eines Testfalls zu prüfen, können jedoch auch Testorakel benutzt werden. Unter einem

Testorakel versteht man eine Quelle, die Auskunft über einen zu erwartendes Ergebnis in einem Testfall gibt. Es gibt zahlreiche Ansätze diese Testorakel in unterschiedlichen Bereichen wie Spezifikation und GUI zu automatisieren. [MPS00] [RAO92] [SKMH09] Solche automatisierten Testorakel können in automatisierten Testfällen dann zur Testauswertung dienen.

3.3 Möglichkeiten zur Testautomatisierung in Erstellung und Durchführung

Der vorangegangene Abschnitt sollte gezeigt haben, dass die Bereiche und die Möglichkeiten der Testautomatisierung breit gefächert sind. Dieser Abschnitt konzentriert sich auf die Bereiche der Testcodeerstellung und Testdurchführung und soll die gängigen Möglichkeiten, die hier in der Automatisierung gegeben sind aufzeigen. Meszaros [Mes03] stellt dazu eine dreidimensionale Matrix auf. Die drei Dimensionen sind:

- Die Granularität des zu testenden Systems. Meszaros orientiert sich dabei stark an den Testphasen des V-Modells und unterscheidet in Units (z.B. ein Modul, eine Klasse oder eine Methode), integrierte Komponenten und das gesamte System.
- Testcodeerstellung. Meszaros nennt hier das manuelle erstellen von Testskripten und das halbautomatisierte erzeugen über „recorde-and-playback“ Tools.
- Die Schnittstelle zum System. Testfälle können sich direkt der API der Software bedienen oder über die Benutzeroberfläche (UI) arbeiten.

Diese Matrix ergiebt 2x2x3 mögliche Kombinationen, wie in Abbildung 3.1 dargestellt.
Dabei lassen sich vier Hauptbereiche identifizieren in denen jeweils ein unterschiedliches Vorgehen bei der Testautomatisierung benötigt wird. Diese vier Bereiche bilden die Vorderseite des Würfels. Innerhalb der dritten Dimension, der Granulatitätsstufen, unterscheidet sich das Vorgehen bei der Testautomatisierung nur gering. Aus diesem Grund ist eine Unterteilung wie sie das V-Modell beim Testen trifft für eine Unterteilung innerhalb der Testautomatisierung alleine nicht sinnvoll. Die Methoden und Tools die in einzelnen Test-Phasen des V-Modells verwendet werden könne durchaus identisch sein.

3.3.1 Rechte obere Quadranten - XUnit

Die Methoden der Testautomatisierung der drei oberen rechten Quadranten bilden die modernen „XUnit-Frameworks“, die nahezu in jeder Programmiersprache existieren. Der Begriff

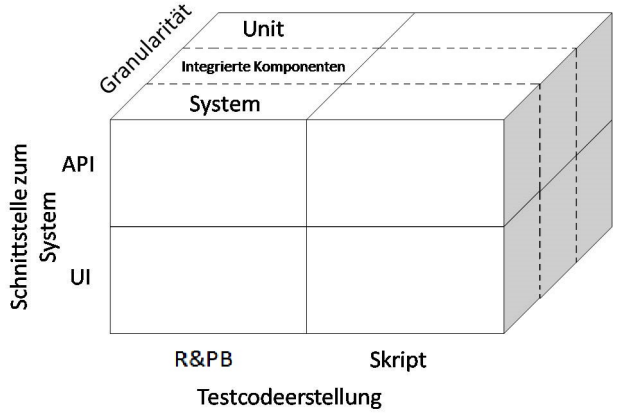


Abbildung 3.1: Die drei Dimensionen der Testautomatisierung

„XUnit-Framework “ ist als reiner Name zu sehen und bedeutet nicht, dass sich diese Frameworks nur mit Testfällen der Granularitätsstufe „Unit“ befassen. Mit dieser Art von Framework ist es möglich, Tests auf allen Granularitätsebenen umzusetzen. Ein weit verbreitetes Beispiel für ein solches Framework wäre beispielsweise JUnit. Bei reinen Unittests werden alle Abhängigkeiten mit Hilfe von zum Beispiel Mockups aufgelöst. Bei Integrationstests werden diese Abhängigkeiten mehr und mehr mit den richtigen Implementierungen gefüllt. Mit Hilfe von weiteren Aufsätzen für das Unit-Framework ist es sogar möglich Systemtests durchzuführen.

3.3.2 Rechte untere Quadranten - Geskriptete UI-Tests

Die drei rechten unteren Quadranten befassen sich mit Testfällen, die von Hand geskriptet wurden und als Schnittstelle zum System die Benutzeroberfläche verwenden. In der Regel handelt es sich dabei um eine Erweiterung der modernen XUnit-Frameworks. Beispiele wären hierfür Selenium [Sela] oder HttpUnit [Htt], die zur Teststeuerung ein XUnit-Framework benutzen und dieses um Methoden erweitern, mit deren Hilfe mit der Benutzeroberfläche des Systems kommuniziert werden kann. HttpUnit generiert dazu eigene Requests und macht die Antworten des Servers über eine Objektstruktur erreichbar. Selenium bedient sich der Funktionalität eines Browsers, um mit dem Webserver zu kommunizieren. Das hat den Vorteil, dass hierbei auch gleich das Verhalten von verschiedenen Webbrowsern gegen die Anwendung getestet werden kann. In den meisten Fällen handelt es sich bei diesen Testfällen um Testfälle aus dem Bereich der Granularitätsstufe „System“. Es ist jedoch auch möglich auf diese

Weise Testfälle für die anderen Granularitätsstufen zu entwickeln. Wird beispielsweise die Fachlogik aus dem System entfernt, ist es möglich Testfälle nur für die Benutzeroberfläche als Komponente zu entwickeln.

3.3.3 Linke untere Quadranten - Robot User

Die linken unteren Quadranten fasst Meszaros unter dem Begriff „Robot User“ [Mes03] zusammen. Bei diesem Vorgehen interagiert ein Tester manuell mit dem Benutzerinterface des Systems. Alle Interaktionen werden dabei aufgezeichnet und gespeichert. Nachfolgend können die so aufgezeichneten Testfälle immer wieder abgespielt werden. Diesen Ansatz verfolgen die meisten kommerziellen Tools für Testautomatisierung und wird hauptsächlich für das Testen auf der Granularitätsstufe „System“ verwendet. Wie schon bei geskripteten UI-Tests können aber auch hier die anderen Granularitätsstufen abgedeckt werden, wenn gewisse Teile des Systems mit Hilfe von zum Beispiel Mockups entfernt werden. Ein bekanntes Open Source-Tool, welches neben den geskripteten UI-Tests auch diesen Ansatz unterstütz,t wäre Selenium.

3.3.4 Rechte obere Quadranten - Internes R&PB

Das Erstellen von Testfällen in diesem Bereich erfordert das Implementieren einer „recorde-and-playback“ -API die Unterhalb der Ebene des Benutzerinterfaces arbeitet. Mit Hilfe dieser API können dann alle Aufrufe, die den Zustand des Systems beeinflussen, aufgezeichnet werden. Diese Aufzeichnungen können dann wieder als Eingabe für das erneute Abspielen des Workflows verwendet werden. „Recorde-and-playback“ -Metoden, bei welchen nicht das Benutzerinterface sondern die API als Schnittstelle zum System verwendet wird, sind jedoch nicht sehr weit verbreitet.

4 Testautomatisierung mit Selenium

Die Benutzeroberfläche ist als Schnittstelle für die Testautomatisierung weit verbreitet. Hierfür gibt es mehrere Gründe. Die Benutzeroberfläche ist sowohl für Tester, als auch für den Entwickler leicht greifbar und sehr anschaulich. Testfälle die über die Benutzeroberfläche arbeiten, kommen der realen Verwendung der Anwendung sehr nahe und die Dokumentation ist auf dieser Ebene meist am vollständigsten. Wird die Benutzeroberfläche für die Automatisierung verwendet, kommt das Vorgehen dem von klassischen Systemtests sehr nahe, da diese zumeist über die selbe Schnittstelle durchgeführt werden. [Sei12, vgl. Seite 48] Das fördert vor allem die Akzeptanz des Testautomatisierungsprojektes auf der Fachseite, da schnell sichtbare Erfolge erzielt werden können. Ein weit verbreitetes Tool für die Automatisierung von Tests gegen die Benutzeroberfläche ist Selenium.

4.1 Selenium

Selenium [Sela] ist eines der am weitesten verbreiteten Open-Source-Automatisierungswerkzeuge für Webapplikationen. Ordnet man das Tool gegen die in Kapitel 3.3 beschriebene Unterteilung der Testautomatisierung ein, befindet sich Selenium in den unteren beiden Quadranten. Selenium ist also ein Tool, das sowohl das manuelle Skripten von UI Tests, wie auch das halbautomatische „recorde-and-playback“ unterstützt. Selenium ist genaugenommen aber kein einzelnes Tool. Der Begriff steht eher für eine Reihe von Komponenten mit unterschiedlicher Funktionalität, die der Testautomatisierung dienen. Dabei können folgende Teile unterschieden werden:

- **Selenium IDE** (Integrated Development Environment), eine Firefox-Extension, welche eine „recorde-and-playback “ - Funktionalität beinhaltet. Mit der Selenium IDE ist es möglich Browserinteraktionen aufzuzeichen und die so erstellten Skripte zu editieren.
- **Selenium Core** enthält die komplette Basisfunktionalität von Selenium, also das Testbefehl-API und den Test-Runner. Mit Hilfe des Selenium Core können aufgenommene Skripte später wieder abgespielt werden.

- **Selenium WebDriver** ermöglicht es, Selenium aus Skripten und Programmiersprachen zu verwenden. Der WebDriver bietet dazu eine API, die es ermöglicht mit unterschiedlichen Browsern zu kommunizieren.
- **Selenium Grid** für die Parallelisierung von Testdurchläufen.

Was Selenium also bietet, ist eine Reihe von Komponenten, die der Testautomatisierung in den Bereichen Testcodeerstellung 3.2.2 und Testdurchführung 3.2.3 dienen.

4.2 Testdesign mit Selenium

Selenium bietet kein Werkzeug, dass den Tester in der Designphase des Testprozesse unterstützt. Es ist mit Selenium also nicht möglich automatisiert Testfälle generieren zu lassen. Entscheidet man sich für Selenium als Testautomatisierungslösung, werden die Testfälle in der Regel wie in Kapitel 2.3.2 Testanalyse und Design beschrieben, manuell erstellt. Selenium unterstützt zwar nicht den Desingprozess von Testfällen, versperrt jedoch auch nicht eine mögliche Automatisierung. Werden Testfälle, wie in Kapitel 3.2.1 beschrieben automatisch erstellt, könne sie durchaus später mit Hilfe von Selenium automatisiert werden. Beispielsweise wäre ein datengetriebener Automatisierungsansatz mit zuvor generierten Testeingaben durchaus denkbar. Das Zusammenspiel von beiden Phasen, also das automatische designen von Testfällen in Kombination mit einer automatischen Testfallgenerierung für die auch gleich automatisiert die Testskripte generiert werden, wie es beispielsweise modellbasierte Ansätze ermöglichen, ist jedoch ohne weiteres nicht möglich.

4.3 Testcodeerstellung mit Selenium

Im Bereich von GUI-Applikationen stellen Webanwendungen einen Spezialfall dar. Anders als in den meisten Desktopanwendungen ist die Kernfunktionalität der Anwendung serverseitig realisiert. Als Kommunikationsschnittstelle zwischen Clients und Server dient HTML. Die eigentliche Benutzeroberfläche wird erst auf Clientseite durch den Browser aus dem gelieferten HTML-Dokument erzeugt. Das bietet für Testtools eine gute Basis, um auf die Elemente der Benutzeroberfläche zuzugreifen. [Sei12, vgl. Seite 59] UI-Testtools für Desktopanwendungen haben oft mit großen Herausforderungen zu kämpfen, um die einzelenen Grafikelemente einer Anwendung zu identifizieren. Selenium kann hier einfach auf Browserfunktionalitäten, bzw. auf die Struktur der HTML-Seite (Document Object Model) zurückgreifen. Über das Document Object Model können Tester die einzelnen Elemente einer Seite identifizieren

und Interaktionen durchführen. Auf diese Weise kann der Workflow eines Testfalls in der Anwendung nachgebildet werden.
Selenium bietet dafür zwei unterschiedliche Möglichkeiten. Der Testcode zum Abbilden des Workflows kann halb automatisiert über eine „recorde-and-playback„ - Funktionalität oder manuell erstellt werden.

4.3.1 Recorde-and-playback

Das automatisierte Aufnahmen von Testfällen ist eine der wichtigsten Funktionen der Selenium IDE. Befindet sich die IDE im Aufnahmemodus, werden alle Interaktionen des Benutzers mit dem Browser gespeichert. Das so erstellte Testskript wird in einer Selenium eigenen Sprache gespeichert. Diese Sprache wird Selenese genannt. Die einzelnen Selenese-Kommandos werden in einer HTML-Tabellenstruktur abgelegt. Diese Tabellen mit Selenese-Kommandos können später wieder verwendet werden, um den gespeicherten Workflow erneut zu durchlaufen. Selenium leidet jedoch unter den selben Problemen, unter denen die meisten „recorde-and-playback„ -Tools leiden. Aufgenommene Testfälle sind in der Regel instabil. Um die einzelnen Elemente der Website zu identifizieren, benutzt Selenium das Document Object Model der Website. Schon kleine Änderungen an der Oberfläche oder leicht veränderte Testdaten können die Struktur der Seite so stark verändern, dass die Testfälle die gespeicherten Elemente nicht mehr identifizieren können. Die über die „recorde-and-playback „-Funktionalität erstellte Testfälle erfordern daher eine vorgelagerte Planungs- und nachgelagerte Nachbearbeitungsphase, um die Testskripte auch für nachfolgende Durchläufe robust zu machen. Dieses Vorgehen widerspricht aber dem eigentlichen Grundgedanken von „recorde-and-playback„ der darauf abzielt, die Testskripterstellung möglichst einfach und schnell zu gestalten.
Benutzt man für die Testskrtipterstellung die „recorde-and-playback„ -Funktionalität der Selenium IDE, ist man nicht an die Sprace Selenese gebunden. Die IDE bietet die Funktionalität, die in Selenese erstellten Testskripte in eine Reihe von verschiedenen Programmiersprachen zu exportieren. Die so erstellten Testskripte sind in ihrem Aufbau zu den Testskripten identisch die in Selenes generiert wurden und basieren immer auf einem xUnit-Framework der ausgewählten Sprache. Sie leiden daher auch unter der selben Instabilität, unter der bereits die Selenese Testfälle leiden. Darüber hinaus ist der generierte Testcode wenig gekapselt und schwer wiederverwendbar. Das macht die generierten Skripte sehr Wartungsaufwändig und schlecht lesbar. Diese Funktionalität sollte daher eher als Prototyping von Skripten verstanden werden.

4.3.2 Manuell

Eine andere Möglichkeit ist das manuelle programmieren der Testskripte. Der manuell erstellte Testcode bedient sich den selben Tools, die auch die generierten und exportierten Testfälle aus der „recorde-and-playback„ -Variante verwenden um mit der Anwendung zu kommunizieren. Das macht die Testfallerstellung zu Beginn, im Vergleich zur halbautomatischen Variante aufwendig. Dieser Mehraufwand kann allerdings durch deutlich bessere Wartbarkeit und Wiederverwendbarkeit schnell eingespart werden. Im Gegensatz du den generierten Testfällen kann, bei einem manuellen Ansatz von Anfang an, auf eine wartbare und wiederverwendbare Struktur in den Testfällen geachtet werden. Als best practice hat sich hierfür das Page Object Design Pattern [Selb] durchgesetzt. Jede Maske der Anwendung wird als eigene Klasse umgesetzt. Diese Klassen beinhalten als einziger Ort in den Testskripten die Informationen über Funktionalität und Aufbau der Maske. So können redundanter Code und redundante Informationshaltung vermieden werden. Die Wartbarkeit und Robustheit der Testskripte steigt.

4.4 Testdurchführung mit Selenium

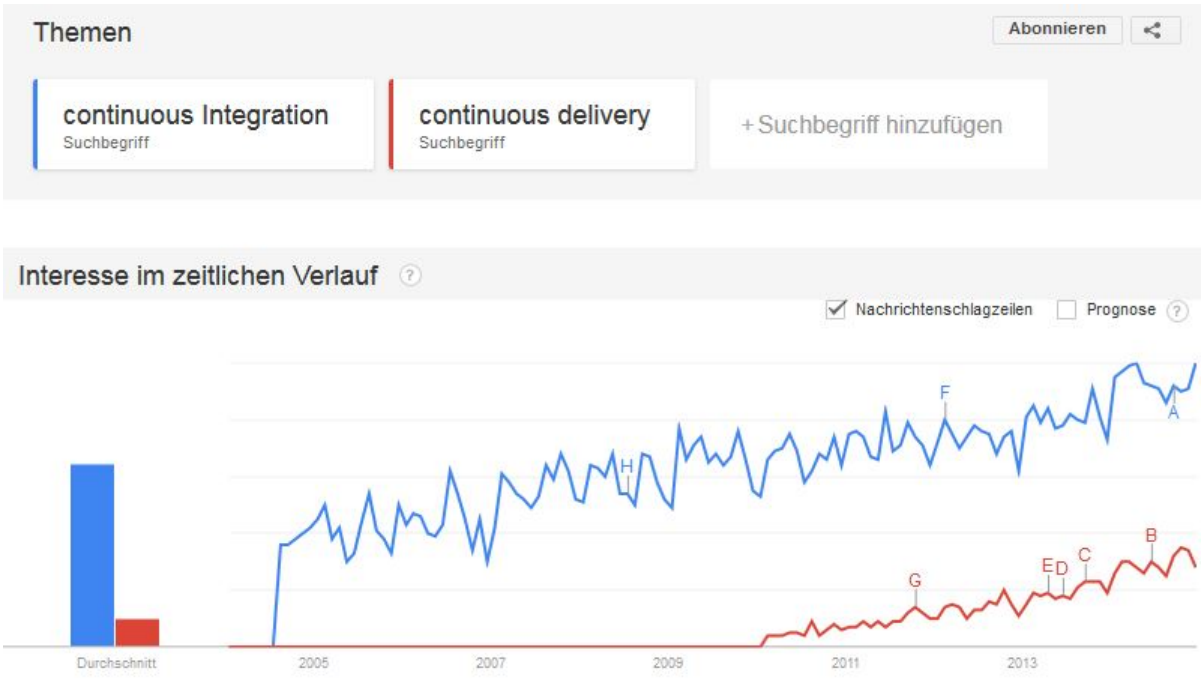
Testfälle die mit Hilfe der „recorde-and-playback„ -Funktionalität aufgenommen wurden, können direkt über die Selenium IDE oder auch in Selenium Core, einem HTML-basierten Durchführungswerkzeug automatisiert ausgeführt werden. Manuell erstellte bzw. exportierte Testfälle bedienen sich zur Ausführung dem entsprechenden xUnit Framework der benutzten Programmiersprache. Im Fall von Java wäre das beispielsweise JUnit oder TestNG. Selenium setzt für die Ausführung also auf gut verbreitete Frameworks, die im Bereich der Unit bzw. Integrationstests gegen die API-Schnittstelle bereits als Standartframeworks verwendet werden. Diese Frameworks sind den meisten Entwicklern bereits bekannt und gut in den Entwicklungsprozess integriert. In Java wird beispielsweise JUnit in allen gängigen IDEs durch Plugins unterstützt. Noch viel wichtiger ist jedoch, die bereits vorhandene Integration in den Buildprozess. Hält man sich an Standarttools, wie Maven oder Gradle zum bauen seiner Projekte, können Testfälle die auf den gängigen xUnit-Frameworks basieren, direkt in den Buildprozess eingebunden werden. Die automatisierte Ausführung der UI-Tests mit Selenium ist daher besonders einfach, da sie mit Standartmitteln möglich ist.

4.5 Testabdeckung mit Selenium

Nach Abschluss der Tests stellt sich häufig die Frage: Wie gut sind meine Tests? Habe ich genug Testfälle, oder benötige ich noch weitere? Um diese Frage zu beantworten wird als Metrik häufig die Codeabdeckung durch die vorhandenen Testfälle herangezogen. Bei diesem Vorgehen wird gemessen, wie viel Prozent der Codezeilen der zu testenden Anwendung bei der Ausführung der vorhandenen Testfällen durchlaufen wird. In herkömmlichen Testfällen, welche die API der Anwendung als Schnittstelle benutzen, ist das recht einfach möglich. Möchte man die Codeabdeckung seiner Unittests messen, können in Java beispielsweise Tools, wie EclEmma oder JaCoCo verwendet werden. Selenium verwendet als Schnittstelle allerdings nicht die API der Anwendung, sondern das UI. Das messen der Codeabdeckung ist in diesem Fall deutlich schwieriger. In der Regel wird gegen eine fertig gebaute Anwendung getestet. Es muss also darauf geachtet werden, dass der durchlaufene Code in der Anwendung und nicht im Testprojekt gemessen wird. Das erweist sich meist als deutlich schwierigere Aufgabe, als das Messen beim Verwenden der API. Das Vorgehen, um eine Messung der Abdeckung in einem Existierenden Projekt zu erreichen, ist stark von der Programmiersprache, in der die zu testende Anwendung entwickelt wurde und von den verwendeten Testtools abhängig. Im Java ist hierfür beispielsweise eine Bytcode Manipulation der zu testenden Anwendung notwendig, bei der jeder Klasse, in der die Codeabdeckung gemessen werden soll, die entsprechenden Anweisungen injiziert werden. Das messen der Codeabdeckung in ausgelagerten Projekten ist also prinzipiell möglich und wird auch von Tools wie EclEmma unterstützt, erweist sich aber als deutlich schwieriger als beim Testen gegen die API.

5 Ausblick

Das Thema Testautomatisierung wird auch in den kommenden Jahren ein wichtiges Thema im Bereich der Softwareentwicklung bleiben. Vor allem Continous Integration und Continous Delivery werden hier weiter eine treibende Kraft sein. Diese zwei Bereiche sind als Entwicklungskonzepte derzeit sehr stark im Trend, was auch der Verlauf der Suchanfragen, in den letzten Jahren, zeigt. Die Testautomatisierung ist ein integraler Bestandteil dieser

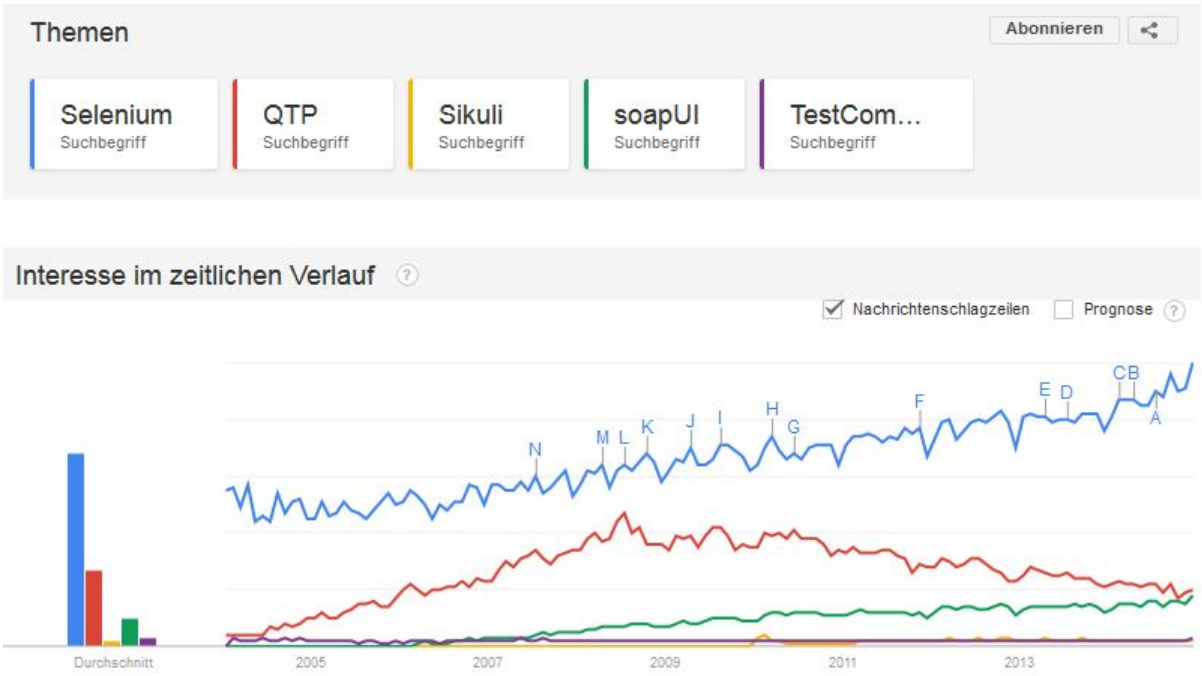


Quelle: [Goo14]

Abbildung 5.1: Trend der Suchanfragen nach dem Begriffen „continous integration“ und „continous delivery“ seit 2006

Entwicklungskonzepte und wird daher von diesem Trend mitgetragen. Unittests bzw. Integrationstests mit den gängigen XUnit-Frameworks, haben sich bereits in vielen Projekten durchgesetzt. Für eine sinvolle Continous Integration bzw. Continous Delivery müssen jedoch alle Bereiche, von Unit- bis Systemtest, durch eine Automatisierung abgedeckt werden. Es

ist daher zu erwarten, dass auch die Automatisierung in diesen Bereichen für viele Projekte an Interesse gewinnen wird.
Im Bereich der Automatisierungstools wird Selenium auch in der nächsten Zeit eine weit verbreitete Lösung darstellen. Der Verlauf der Suchanfragen in den letzten Jahren zeigt, dass Selenium im Vergleich zu anderen gängigen Lösungen immer noch sehr beliebt ist.



Quelle: [Goo14]

Abbildung 5.2: Trend der Suchanfragen nach gängigen Automatisierungstools

Abbildungsverzeichnis

Tabellenverzeichnis

Literaturverzeichnis

[AGIS14] Amannejad, Y. ; Garousi, V. ; Irving, R. ; Sahaf, Z.: A Search-Based Approach for Cost-Effective Software Test Automation Decision Support and an Industrial Case Study. In: *2014 IEEE Seventh International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, 2014, S. 302–311

[BGLP08] Bouquet, Fabrice ; Grandpierre, Christophe ; Legeard, Bruno ; Peureux, Fabien: A Test Generation Solution to Automate Software Testing. In: *Proceedings of the 3rd International Workshop on Automation of Software Test*. New York, NY, USA : ACM, 2008 (AST '08). – ISBN 978–1–60558–030–2, 45–48

[Bur03] Burnstein, Ilene: *Practical Software Testing: A Process-Oriented Approach*. Auflage: 2003. New York : Springer, 2003. – ISBN 9780387951317

[FG99] Fewster, Mark ; Graham, Dorothy: *Software Test Automation Effective use of test execution tools*. Addison-Wesley, 1999. – ISBN 0201331403

[Goo14] Google: *Google Trends - Websuche-Interesse - Weltweit, 2004 - heute*. `http://www.google.de/trends/explore`. Version: 2014

[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

[Htt] HttpUnit: *HttpUnit Home*. `http://httpunit.sourceforge.net/`

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

[Int14] International Organization for Standardization (ISO): *SO/IEC 25000:2014, Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — Guide to SQuaRE*. März 2014

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

[ISO10] ISO/IEC/IEEE: Systems and software engineering – Vocabulary. In: *ISO/IEC/IEEE 24765:2010(E)* (2010), Dezember, S. 1–418. `http://dx.doi.org/10.1109/IEEESTD.2010.5733835`. – DOI 10.1109/IEEESTD.2010.5733835

[LFK03] Last, Mark ; Friedman, Menahem ; Kandel, Abraham: *The Data Mining Approach to Automated Software Testing.* 2003

[Mes03] Meszaros, Gerard: Agile Regression Testing Using Record & Playback. In: *Companion of the 18th Annual ACM SIGPLAN Conference on Object-oriented Programming, Systems, Languages, and Applications.* New York, NY, USA : ACM, 2003 (OOPSLA '03). – ISBN 1–58113–751–6, 353–360

[MPS99] Memon, Atif M. ; Pollack, Martha E. ; Soffa, Mary L.: Using a Goal-driven Approach to Generate Test Cases for GUIs. In: *Proceedings of the 21st International Conference on Software Engineering.* New York, NY, USA : ACM, 1999 (ICSE '99). – ISBN 1–58113–074–0, 257–266

[MPS00] Memon, Atif M. ; Pollack, Martha E. ; Soffa, Mary L.: Automated Test Oracles for GUIs. In: *Proceedings of the 8th ACM SIGSOFT International Symposium on Foundations of Software Engineering: Twenty-first Century Applications.* New York, NY, USA : ACM, 2000 (SIGSOFT '00/FSE-8). – ISBN 1–58113–205–0, 30–39

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

[RAO92] Richardson, D.J. ; Aha, S.L. ; O'Malley, T.O.: Specification-based test oracles for reactive systems. In: *International Conference on Software Engineering, 1992*, 1992, S. 105–118

[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

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

[Sela] Selenium: *Selenium - Web Browser Automation*. `http://www.seleniumhq.org/`

[Selb] Selenium: *Test Design Considerations — Selenium Documentation*. `http://docs.seleniumhq.org/docs/06_test_design_considerations.jsp#page-object-design-pattern`

[SKMH09] Shahamiri, S.R. ; Kadir, W.M.N.W. ; Mohd-Hashim, S.Z.: A Comparative Study on Automated Software Test Oracle Methods. In: *Fourth International Conference on Software Engineering Advances, 2009. ICSEA '09*, 2009, S. 140–145

[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

[TVKB01] Tahat, L.H. ; Vaysburg, B. ; Korel, B. ; Bader, A.J.: Requirement-based automated black-box test generation. In: *Computer Software and Applications Conference, 2001. COMPSAC 2001. 25th Annual International*, 2001, S. 489–495