René Weber

Auslieferungs- und Testmanagement mit .NET unter besonderer Beachtung automatisierter Tests für Web- bzw. Desktopanwendungen

eingereicht als

Bachelorarbeit

an der

HOCHSCHULE MITTWEIDA
UNIVERSITY OF APPLIED SCIENCES

Fachbereich

Mathematik Naturwissenschaften Informatik

Mittweida, 2011

Erstprüfer: Prof. Dr. rer. nat. Günter Werner

Zweitprüfer: Dipl.-Informatiker Jens Wagner

Die Vorgelegte Arbeit wurde verteidigt am: 24.11.2011

Bibliographische Beschreibung

René Weber:

Auslieferungs- und Testmanagement mit .NET unter besonderer Beachtung automatisierter Tests für Web- bzw. Desktopanwendungen

2011 – Mittweida, Hochschule Mittweida, Fachbereich Mathematik Naturwissenschaften Informatik, Bachelorarbeit

Kurzreferat

Diese Arbeit beschäftigt sich hauptsächlich mit Vorgehensweisen und Konzepten zum automatischen Testen von Web- und Desktopanwendungen. Es wird dabei allerdings auch auf das allgemeine Auslieferungs- und Testmanagement von Software eingegangen. Dazu gehören unter anderem der Ablauf des Auslieferungsprozesses, Einführung zum Testen von Software oder verschiedene Testmethoden.

Stichwörter: Software Qualität, Whitebox-Test, Blackbox-Test, Testautomatisierung, Selenium

Abstract

This thesis mainly deals with approaches and concepts for automated testing of web and desktop applications. Besides that an elaboration about general delivery and test management of software is made. This includes the flow of the delivery process, introduction to testing of software, different testing techniques and so on.

Keywords: software quality, whitebox-testing, blackbox-testing, test automation, Selenium

Inhaltsverzeichnis

Abbildungsverzeichnis

Listings

Glossar

(HTTP-)Request	Eine Anfrage an einen Server
(HTTP-)Response	Eine Antwort von einem Server auf eine Anfrage
Capture-Replay	Beschreibt das Aufnehmen von Aktionen die vom Nutzer durchgeführt wurden bzw. das Abspielen dieser
CSS	*Cascading Style Sheets* – Sprache zum Formatieren von strukturierten Dokumenten
CVS	*Concurrent Versions System* - ist ein Versionsverwaltungs-system
DOJO	JavaScript-Framework
Enum	Bezeichnet einen ein Datentyp in Programmier-sprachen zur Aufzählung bestimmter Werte
Exception	Eine Ausnahmesituation in einem Programm
Factory (Entwurfsmuster)	Beschreibt ein Entwurfsmuster, bei dem eine speziell angelegte Methode das Erstellen bestimmter Klassen übernimmt.
GWT	*Google Web-Toolkit* – Toolkit zur Entwicklung von Webanwendungen
HTTP	*Hyper Text Transfer Protocol* – Protokoll zur Übertragung von Daten
JavaScript-Alert	Ein Informationsfenster, welches durch JavaScript aufgerufen wurde
Maven	Tool zum Erstellungs- und Abhängigkeitsmanagement
MSAA	*Microsoft Active Accessibility* – Ein Framework von Microsoft zur Barrierefreiheit
SGML	*Standard Generalized Markup Language* – Metasprache mit die es ermöglicht verschiedene Auszeichnungssprachen zu definieren
SVN	*Subversion* – ist ein Versionsverwaltungssystem
UIA	*User-Interface Automation* – Ein Framework von Microsoft zur Barrierefreiheit
Wrappen von Komponenten	Beschreibt das Erstellen einer Komponente, welche eine andere Komponente umgibt bzw. umhüllt.

Vorwort

Diese Arbeit richtet sich vorrangig an Personen, welche im Bereich der Informatik tätig sind. Dabei ist es weiterhin von Vorteil Kenntnisse im Bereich .NET und C# mitzubringen.

„Program testing can be a very effective way to show the presence of bugs, but is hopelessly inadequate for showing their absence."

Edsger W. Dijkstra, 1972

1 Einleitung

1.1 Motivation

Bei der Auslieferung von Softwareprodukten fallen immer wieder gleichartige Tätigkeiten an. Gerade das immer wiederkehrende Testen ähnlicher bzw. gleicher Funktionalitäten eines Softwaresystems, bei der Auslieferung neuer Versionen, kann sehr aufwendig sein. Deshalb ist mit einer Automatisierung dieser Prozesse eine beträchtliche Optimierung des Auslieferungsvorganges möglich und dadurch kann, über längere Zeit gesehen, eine Reduzierung der Kosten erreicht werden.

Diese Arbeit soll eine Evaluierung zum Auslieferungs- und Testmanagement, insbesondere dem automatisierten Testen, darstellen. Hierbei sind auf der einen Seite Webapplikationen und auf der anderen Desktopapplikationen zu betrachten.

1.2 Problemstellung

Bei großen Softwareprodukten stellt sich schon immer das Problem eines sehr großen Testaufwandes, um ein möglichst fehlerfreies Funktionieren des Produktes zu gewährleisten. Viele Firmen haben deshalb eigene Testabteilungen, die nur zur Qualitätssicherung des Produktes zur Verfügung stehen und den enormen Testaufwand bewältigen müssen. Durch Hilfswerkezeuge, Paradigmen oder Vorgehensweisen wird versucht, sowohl bei der Entwicklung als auch beim Testen der Software, den Testaufwand zu verringern. Im Laufe der Zeit haben sich dabei verschiedene, mehr oder weniger gut funktionierende Möglichkeiten herausgebildet. Einige dieser Varianten haben sich weitgehend etabliert, sodass man diese als heutigen Standard bezeichnen kann. Dazu zählen unter anderem Unit-Tests, die beim Entwickeln großer Softwareprodukte nicht mehr wegzudenken sind.

Allerdings gibt es auch Bereiche, in denen derartige Werkzeuge nicht in solchem Umfang vorliegen bzw. nicht direkt die gewünschten Funktionalitäten anbieten. Dies kann man unter anderem von den sogenannten Systemtests behaupten. Solche Systemtests sind zum Testen des gesamten Systems auf die Richtigkeit aller beinhalteten Funktionen. Da größere Abschnitte des Systems zusammenhängend getestet werden müssen, ist es notwendig, diese Abläufe komplett im Test nachzustellen. Sowohl im Bereich der Webapplikationen als auch Desktopapplikationen, gibt es Tools, die dies Bewerkstelligen sollen. Dabei besteht allerdings das Problem, dass diese Tools oft nicht als Komplettlösung angesehen werden können, nicht die gestellten Anforderungen erfüllen oder sehr hohe Kosten mit sich bringen. Gerade im Bereich der Desktopapplikationen zeigen sich diese Probleme sehr deutlich. Das Hauptaugenmerk liegt in dieser Arbeit liegt jedoch auf dem automatischen Testen von Webapplikationen. Weiterhin soll auch ein Einblick

in die verschiedenen Möglichkeiten zum automatischen Testen von Desktopanwendungen gegeben werden.

1.3 Aufbau der Arbeit

Nach der Einleitung, wird im zweiten Kapitel der typische Auslieferungsprozess von Softwareprodukten betrachtet. Dabei stehen die technischen Schritte, die Teil des Prozesses sind, im Mittelpunkt der Betrachtung. Zu diesen gehören zum Beispiel das Kompilieren der Software, Erstellen eines benutzerfreundlichen Installationsprogramms oder das Testen des Gesamtsystems. Es wird dabei im Besonderen auf das Testen des auszuliefernden Produktes eingegangen.

Im dritten Kapitel wird das Testen von Software noch näher veranschaulicht, indem die Grundlagen vertieft werden. Dazu gehören unter anderem eine genaue Definition des Begriffs Softwaretests, Ziele des Testens und Vorteile bzw. Nachteile des Testens von Software.

Nachdem die Grundlagen geschaffen sind, werden im vierten Kapitel verschiedene Arten von Softwaretests aufgeführt. Dabei werden Blackbox- und Whiteboxtests betrachtet, die dabei in ihre verschiedenen Einsatzbereiche bzw. Zugehörigkeiten gegliedert werden.

Im fünften Kapitel wird von der allgemeinen Betrachtung auf das automatische Testen von Software übergeleitet. Es werden dabei anfangs wieder die Grundlagen vermittelt. Des Weiteren werden die Unterschiede bzw. die Vor- und Nachteile zum manuellen Testen betrachtet. Weiterhin werden Anwendungsgebiete zum automatischen Testen genannt.

Nachdem auch dieser Bereich ausreichend betrachtet wurde, folgt eine Aufteilung in die zwei großen Kapitel, automatisches Testen von Webanwendungen und automatisches Testen von Desktopanwendungen.

Zuerst wird im sechsten Kapitel das automatische Testen von Webanwendungen veranschaulicht. Dabei werden Besonderheiten und Voraussetzungen zum automatischen Testen von Webanwendungen aufgeführt. Danach wird eine grundsätzliche Beschreibung von Webtechnologien, die wichtig für das automatische Testen von Webapplikationen sind, gegeben. Als nächstes werden Vorgehensweisen zum automatischen Testen von Webanwendungen aufgezeigt. Es soll dabei gezeigt werden, wie man ein automatisiertes Testen erreichen kann, indem man unter anderem die genannten Technologien nutzt. Danach erfolgt eine Evaluierung verschiedener Testtools, die die zuvor aufgezeigten Vorgehensweisen nutzen. Dabei wird insbesondere auf das Tool „Selenium“ mit all seinen Komponenten eingegangen. Da Selenium alleine nicht alle Anforderungen an ein automatisches Testtool erfüllen kann, werden dann Konzepte zum Aufbau zur

Eigenimplementierung eines Tools unter Nutzung von Selenium gegeben. Dazu werden unter anderem verschiedene Probleme beim Implementieren eines solchen Tools und dazu passende Lösungsansätze aufgezeigt.

Im siebten Kapitel wird das automatische Testen von Desktopanwendungen unter Windows beschrieben. Zu Beginn stehen die Besonderheiten zum automatischen Testen unter Windows im Fokus. Weiterhin erfolgt eine Betrachtung des Aufbaus von Desktopanwendungen. Dabei wird im Besonderen auf die GUI (Graphical User Interface) eingegangen, da diese in vielen Fällen der Ansatzpunkt für automatische Tests von Desktopanwendungen ist. Allerdings wird auch auf andere Programmstrukturen eingegangen, die einen Einstiegspunkt zum automatischen Testen geben können. Nachdem nun eine Einführung gegeben und der Aufbau von Desktopanwendungen geklärt wurde, werden die Herangehensweisen zum automatischen Testen solcher Anwendungen umfangreich betrachtet. Dafür werden die in dem vorigen Abschnitt erklärten Strukturen von Desktopanwendungen genutzt, um an diesen die Herangehensweisen darzustellen. Anschließend wird eine Vorstellung vorhandener Tools durchgeführt, welche die besprochenen Herangehensweisen nutzen.

Im letzten Kapitel erfolgen noch ein Fazit der Arbeit und ein Ausblick.

2 Der Auslieferungsprozess

Das folgende Kapitel beschäftigt sich mit dem Auslieferungsprozess von Software. Es werden hierbei die typischen Schritte dieses Prozesses erklärt. Dabei werden insbesondere die Schritte, die sich mit dem Testen der Software beschäftigen, betrachtet.

2.1 Einführung

Die Auslieferung beschäftigt sich mit dem Bereitstellen einer neuen Version eines Softwareproduktes. Dabei müssen sowohl fachliche als auch technische Vorgänge berücksichtigt werden. Gerade bei den technischen Schritten wird eine Automatisierung gewünscht, um den Aufwand der Auslieferung zu verringern. Bei komplexen Softwaresystemen wird dies entweder mittels eigens programmierten Auslieferungstools oder mit vorhandenen Tools wie zum Beispiel Maven, Hudson oder Apache Ant realisiert. Die Anforderungen an ein solches Auslieferungstool hängen entscheidend von dem bereitzustellenden Softwareprodukt ab und die Schritte, welche im Auslieferungsprozess unternommen werden müssen, können somit variieren. Im folgenden Abschnitt werden die typischen Schritte, die meist an ein solches Tool zu durchlaufen hat, aufgeführt.

2.2 Die Schritte des Auslieferungsprozesses

Überprüfen der Software und gegebenenfalls Aktualisierung:

Im ersten Schritt erfolgt im Normallfall eine Überprüfung des Quellcodes auf Aktualität. Da an größeren Projekten viele Personen beteiligt sind, kommt es vor, dass auf den Systemen unterschiedliche Versionen vorhanden sind. Deshalb gibt es in den meisten Fällen ein zentrales System zur Verwaltung und Bereitstellung der unterschiedlichen Quellcode-Versionen. Die bekanntesten, sogenannten Versions-verwaltungssysteme sind SVN (Subversion) und CVS (Concurrent Versions System). Soll also eine Bereitstellung erfolgen, so ist es zweckmäßig, die Version, des auszuliefernden Systems, aus dem Versionsverwaltungssystem zu nutzen. Dies ist ein Schritt, der sich relativ einfach automatisieren lässt, da es verschiedene vorgefertigte Möglichkeiten gibt, auf ein solches Versionsverwaltungssystem zuzugreifen. Als Beispiel soll die SharpSVN-Bibliothek dienen. Sie ermöglicht über die Programmiersprache C# einen Zugriff auf das System „SVN“ und kann damit interagieren. So kann mittels des Auslieferungstools eine Aktualisierung des Softwaresystems erfolgen. Auch Maven bietet zum Beispiel über ein Plugin, Zugriff auf Versionsverwaltungssysteme.

Konfigurieren:

Auch das Projekt selbst bedarf im Normallfall einer Konfiguration, das heißt, es müssen verschiedene Einstellungen getroffen werden, die für den Produktivbetrieb notwendig sind. Auch dies ist automatisierbar indem in den Konfigurationsdateien die Werte entsprechend angepasst werden.

Kompilieren der Software:

In diesem Schritt erfolgt die Kompilierung der Software. Eine Kompilierung ist allerdings nicht bei allen Programmiersprachen notwendig. So gibt es Programmiersprachen wie PHP die nur interpretiert und nicht kompiliert werden. Gerade wenn eine Aktualisierung der Software erfolgte, muss gegebenenfalls eine Neu-Kompilierung stattfinden. Die Kompilierung kann unter Umständen einen sehr großen Aufwand darstellen, wenn viele Abhängigkeiten zum Projekt vorhanden sind. Diese Abhängigkeiten können durch verschiedene externe Bibliotheken, Frameworks etc. entstehen. Wenn also die eigentliche Software bereitgestellt werden soll, muss auch sichergestellt sein, dass alle Abhängigkeiten vorhanden sind und vor allem auch für eine Bereitstellung konfiguriert sind. Um nicht den Überblick über diese Abhängigkeiten und deren Konfigurationen zu verlieren, gibt es Tools, die auch schon bei der Entwicklungsphase helfen können. Solche Tools zum Abhängigkeits-Management werden in der Praxis bei Großprojekten sehr oft eingesetzt. Als typischer Vertreter wäre auch hier Maven zu nennen, welches eine solche Verwaltung unter Java ermöglicht. [Apa11]

Entfernen von Testdaten:

Meist werden während der Entwicklung der Software verschiedene Testdaten genutzt, um ein realistisches Testen zu ermöglichen. So werden zum Beispiel in Datenbanken oft Dummy-Einträge eingefügt, damit eine möglichst realitätsnahe Simulation ermöglicht wird. Natürlich muss darauf geachtet werden, dass diese vorgefertigten Einträge beim Ausliefern entfernt werden. Um dies zu automatisieren, muss ein Zugriff auf die jeweiligen Informationsquellen erfolgen, um dort die vorhandenen Testdaten zu entfernen.

Packen der Software:

Das fertige Produkt sollte am Ende für den Kunden möglichst einfach zu installieren sein. Deshalb sollte die Konfiguration der Software auf dem Zielsystem möglichst automatisch und ohne den Eingriff von außen geschehen. Dafür kann man beispielsweise das Produkt in ein Installationsarchiv verpacken. Dieses installiert und konfiguriert die Software dann so, dass keine oder nur geringfügige Eingriffe von Nöten sind.

Abschließender Test:

Im Zuge der Auslieferung, sollte ein abschließender Test das gesamte System nach Fehlern überprüfen. Dies zeigt vor allem, ob die verschiedenen Bestandteile des Systems erfolgreich miteinander arbeiten oder eben Probleme verursachen. Im nächsten Kapitel wird noch tiefgehender auf die Bedeutung dieser Tests eingegangen.

2.3 Bedeutung von Softwaretests im Auslieferungsprozess

Die Tests während der Auslieferung sind für die Entwickler der Software die letzten Tests vor der Übergabe des Produktes an den Kunden. Sie stellen deshalb auch nochmals eine Qualitätssicherung des Produktes dar. Daher ist es generell angebracht, dass diese Tests möglichst das gesamte System durchlaufen und eine Überprüfung aller Komponenten stattfindet. Die Tests sind also sehr wichtig, um dem Kunden ein Produkt zu liefern, welches den geforderten Qualitäts-anforderungen entspricht. Sie sind aber auch mit die aufwendigsten Tests, da sie nicht nur die Funktionen im Einzelnen, sondern das gesamte Softwaresystem auf Fehlerfreiheit überprüfen sollen.

Im nächsten Kapitel soll vorerst eine Einleitung zum Testen von Software gegeben werden, welches die Grundlagen schafft, um dann auf die verschiedenen speziellen Testarten eingehen zu können.

3 Grundlagen zum Testen von Software

Dieses Kapitel beschäftigt sich mit den Grundlagen von Softwaretests. Am Beginn wird der Begriff des Softwaretests definiert um ein besseres Verständnis zu diesem Begriff zu schaffen. Weiterhin werden die Ziele von Softwaretests definiert. Auch die Bedeutung vom Testen in der Softwaretechnik wird näher betrachtet.

3.1 Definition zu Softwaretests

Die IEEE definiert das Testen von Software wie folgt: [IEE90]:

> *„Test: (1) An activity in which a system or component is executed under specified conditions, the results are observed or recorded, and an evaluation is made of some aspect of the system or component."*

Dieses Zitat besagt frei übersetzt:

„Testen ist eine Aktivität, in welcher ein System oder eine Komponente unter festgelegten Bedingungen ausgeführt wird, die Resultate überwacht oder aufgenommen werden und eine Evaluierung eines bestimmten Teils des Systems bzw. der Komponente durchgeführt wird."

Diese Definition soll nun etwas erläutert werden:

Im ersten Abschnitt wird erklärt, dass ein Test in einem System bzw. einer Komponente mit festgelegten Bedingungen ausgeführt wird. In das System werden also feste Werte gegeben, ein fester Ablauf des Systems ist vorgegeben, um damit nicht zuletzt feste, vorhersehbare Werte vom System geliefert zu bekommen.

Im Weiteren wird erläutert, dass die Resultate des Tests überwacht bzw. aufgenommen werden. Natürlich müssen die Ergebnisse der Tests in irgendeiner Form festgehalten werden, um letztendlich zu erkennen ob Probleme vorhanden sind oder das System reibungslos läuft.

Über die aufgezeichneten Resultate sollte nun eine Untersuchung des betroffenen Teils des Systems bzw. der Komponenten stattfinden, um diese zu verbessern, wie der abschließende Teil des Zitates besagt.

3.2 Ziele von Softwaretests

Das Ziel eines Softwaretests ist es wie schon erwähnt, Fehler im System zu entdecken, damit diese behoben werden können. Im Größeren bedeutet dies eine Verbesserung der Qualität der Software. In der Softwaretechnik haben sich einige Qualitätsmerkmale für Software etabliert. Diese sollen hier in Anlehnung an die ISO/IEC 9126 [Int01] kurz vorgestellt werden.

Funktionalität: Eines der entscheidenden Kriterien der Qualität einer Softwareproduktes ist die Funktionalität. Sie besagt, in welchem Umfang die gewünschten Funktionen umgesetzt wurden. Beispielsweise kann ein Fehler im System dazu führen, dass eine bestimmte Funktion nicht wie gewünscht zur Verfügung steht.

Zuverlässigkeit: Die Zuverlässigkeit von Software ist je nach Einsatzgebiet ein sehr schwerwiegender Faktor. Gerade wenn die Software durch Fehler die Gesundheit der Menschen, die sie Nutzen beeinträchtigen kann, so ist die Zuverlässigkeit von höchster Wichtigkeit. In anderen Bereichen ist dieser Faktor nicht immer so kritisch, da kleinere Unzulänglichkeiten zum Teil eher ertragen werden können.

Effizienz: Die Effizienz eines Programms beschreibt einerseits die Geschwindigkeit der Ausführung, als auch den Ressourcenverbrauch. Auch hier gibt es je nach Bereich eine hohe bzw. niedrigere Anforderung an diesen Faktor.

Benutzbarkeit: Je nach Anwender muss die Software eine bestimmte Benutzerfreundlichkeit gewährleisten. Soll eine Anwendung für die breite Masse erstellt werden, so ist ein hohes Maß an Benutzbarkeit notwendig. Bei Spezialanwendungen ist dies nicht unbedingt so ein schwerwiegender Faktor.

Übertragbarkeit: Die Übertragbarkeit beschreibt die Möglichkeit, ein vorhandenes System in eine neue Umgebung zu integrieren. Dies kann zum Beispiel die Portierung von einem Betriebssystem zum einem anderen, von einem Server zu einem neuen Server oder die Portierung in eine neue Programmiersprache sein.

Änderbarkeit: Dieses Software-Qualitätsmerkmal besagt, welchen Aufwand die Änderung von Teilen eines Systems bedeutet. Ist ein Softwareprodukt ausgeliefert worden, so muss das Produkt in den meisten Fällen im Nachhinein weiter gewartet werden, um es an neue Technologien anzupassen, um im Nachhinein aufgedeckte Fehler zu beheben oder um neue Features zu integrieren. Je nachdem wie gut strukturiert und abstrahiert der Quellcode ist, lässt sich dies mehr oder weniger einfach bewerkstelligen.

Testbarkeit: Die Testbarkeit beschreibt, wie gut ein System zu überprüfen ist. Dies kann zum Beispiel eine Überprüfung auf Fehler oder Performance sein. Eine gute Testbarkeit eines Systems ist jedoch schwer zu realisieren, da die Komplexität der zu testenden Komponenten oft sehr schnell ansteigt.

Transparenz: Erfüllt ein Softwaresystem alle an ihn gestellten Funktionalitäten, so heißt das noch lange nicht, dass der darunterliegende Quellcode gut strukturiert und aufgebaut ist. Die Transparenz beschreibt, in welchem Maße die Oberfläche und die Funktionalitäten die innere Struktur des Systems reflektiert.

Wie zu sehen ist, decken Softwaretests in erster Linie Probleme in den Funktionalitäten des Programmsauf. Allerdings können auch Lasttests durchgeführt werden um Effizienzprobleme anzuzeigen oder Oberflächentests um Probleme in der Benutzbarkeit zu erkennen. Es können also durch die verschiedenen Test-Methoden die Qualitätsmerkmale eines Softwareproduktes überprüft werden, um damit Probleme aufzudecken und diese dann zu beheben.

Die in dieser Arbeit vorgestellten Vorgehensweisen und Konzepte sollen hauptsächlich eine Testautomatisierung zum Testen von Funktionaltäten, also zum Überprüfen der Richtigkeit der Funktionen, ermöglichen.

3.3 Einordnung in die Softwaretechnik

In der Softwaretechnik gibt es eine große Menge an Vorgehensmodellen und in allen diesen gibt es Phasen, in denen Softwaretests durchgeführt werden sollen. Es gibt sogar Vorgehensmodelle, deren Grundlage das Testen von Software ist, wie zum Beispiel Test-Driven Development. Daher sollen die wichtigsten Vorgehensmodelle kurz vorgestellt werden. Anhand dieser sollen im Besonderen die Testphasen erläutert werden, um die zeitliche Einordnung und die Wichtigkeit des Testens während der Entwicklung und Wartung von Software aufzuzeigen.

3.3.1 Wasserfallmodell

Das Wasserfallmodell ist wohl das Bekannteste, wenn auch in der Praxis nicht sehr häufig angewendete, Vorgehensmodell der Softwaretechnik. Das Wasserfallmodell ist in vier bis sieben Phasen (je nach Variante) unterteilt, die sequentiell ausgeführt werden.

Dabei befindet sich nach der Phase der Implementierung des Programms eine Phase für Tests. Diese wiederum ist der Vorläufer der Phase „Installation“. Wie also einzusehen ist, sind in diesem Vorgehensmodell die Tests vor bzw. mit der Auslieferung des Produktes durchzuführen. Diese sind insbesondere System- und Integrationstests. Nachfolgenden ein Diagramm einer Variante eines Wasserfallmodells.

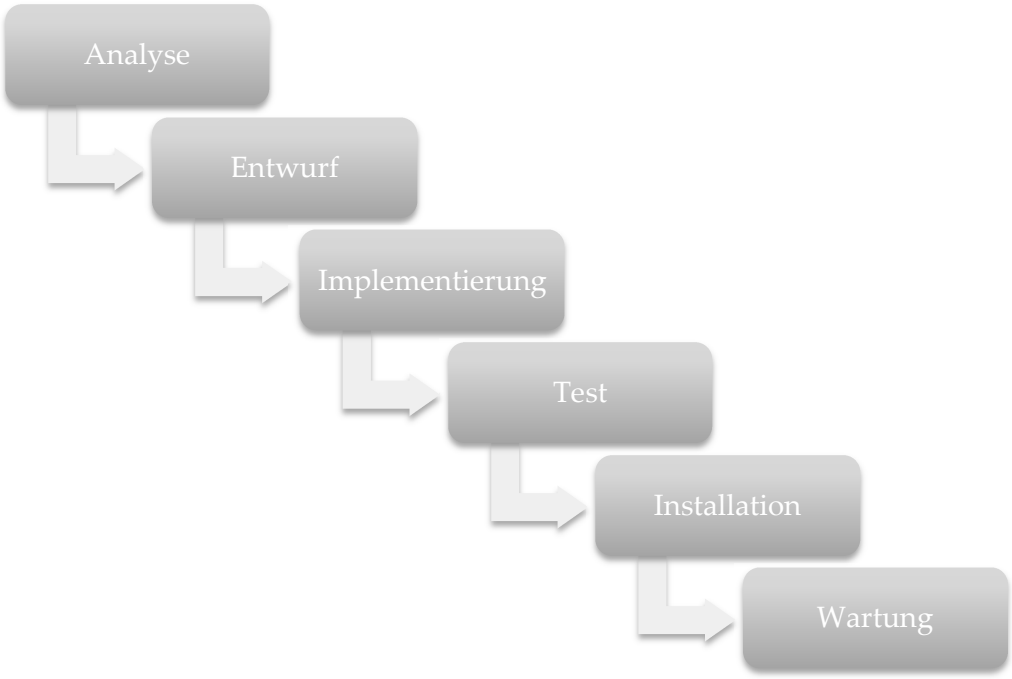


Abbildung 3-1: 6-stufiges Wasserfallmodell

3.3.2 Spiralmodell

Das Spiralmodell soll an dieser Stelle als typischer Vertreter für agile Vorgehensmodelle kurz vorgestellt werden.

Das Modell besteht grundsätzlich aus vier Phasen die immer wieder durchlaufen werden. Diese Phasen sind: Bestimmung der Ziele, Beurteilen von Alternativen, Entwicklung und Test und Planung des nächsten Zyklus. In der dritten Phase finden also mit der Entwicklung auch die Tests statt. In den ersten Zyklen bestehen diese Tests eher aus Verifikationen bzw. Validierungen der erstellten Konzepte und sind damit keine Softwaretests. Erst in den Zyklen, in welchen der eigentliche Quellcode implementiert wird, kann natürlich ein Testen dieses Quellcodes bzw. des Programms als solches stattfinden. Im Folgenden soll noch eine vereinfachte Version des Spiralmodells gegeben werden. Normalerweise werden sicherlich mehr als nur zwei Zyklen durchlaufen.

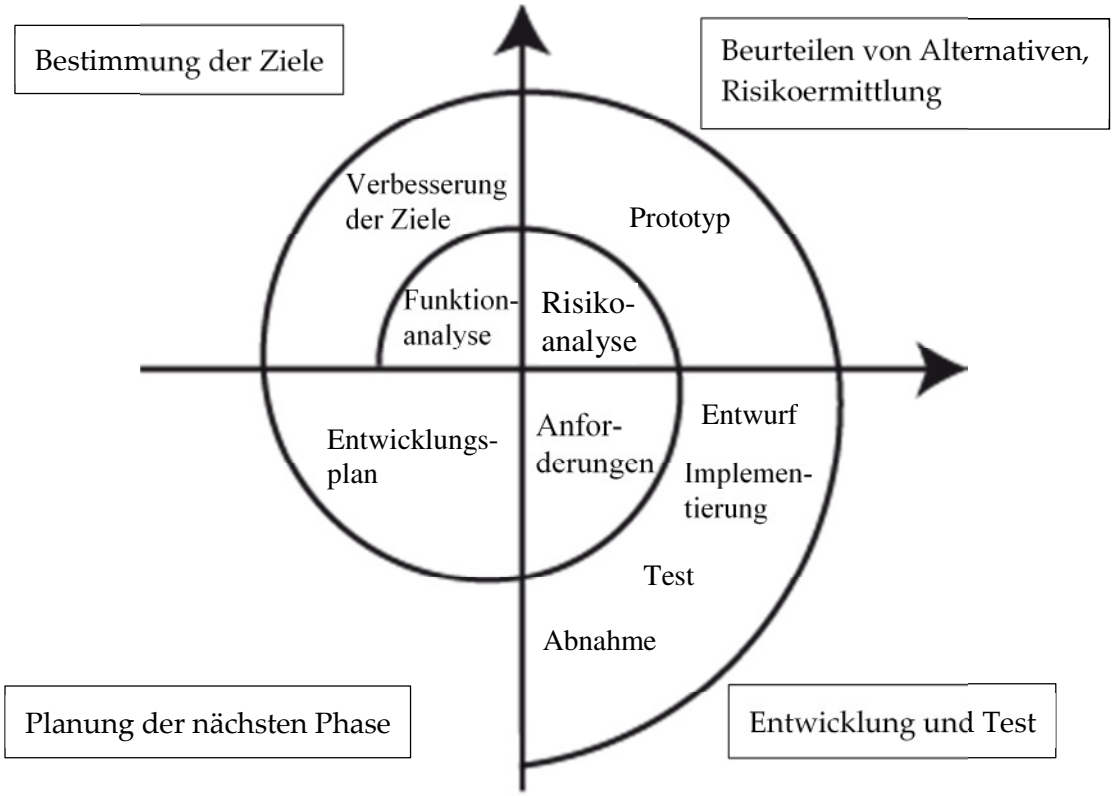


Abbildung 3-2: Vereinfachtes Spiralmodell

Auch hier ist zu sehen, dass im Prinzip mit der Auslieferung die abschließenden Tests erfolgen.

3.3.3 V-Modell

Das V-Modell ist ein Vorgehensmodell welches neben den normalen Entwicklungsphasen auch Phasen zur Qualitätssicherung vorgibt und diese den Entwicklungsphasen zuordnet.

In der ersten Phase des V-Modells steht die Systemanforderungsanalyse, dieser wird die letzte Phase „Abnahme und Nutzung“ zugeordnet. Als nächste Phase folgt die System-Architektur und dieser wird die System-Integration zugeordnet. Weiterhin gibt es den Systementwurf, welchem der Integrations-Test zugehörig ist. Danach folgt die Software-Architektur, welchem die Unit-Test-Phase einberechnet wird. Als letzte Entwicklungsphase ist der Software-Entwurf zu nennen, diese besitzt keine zugeordnete Phase zur Qualitätssicherung. Die Phasen der Qualitätssicherung werden dabei quasi umgekehrt zu den Entwicklungsphasen durchgeführt. Während die Phase „Software-Architektur“ einer der letzten Phasen der Entwicklung ist, so ist der Unit-Test die erste Phase der Qualitätssicherung. Bei diesem Vorgehensmodell ist also das Testen der Software ein wesentlicher Bestandteil, um die Qualität sicherzustellen. Nachfolgend soll ein Diagramm des V-Modells gegeben werden.

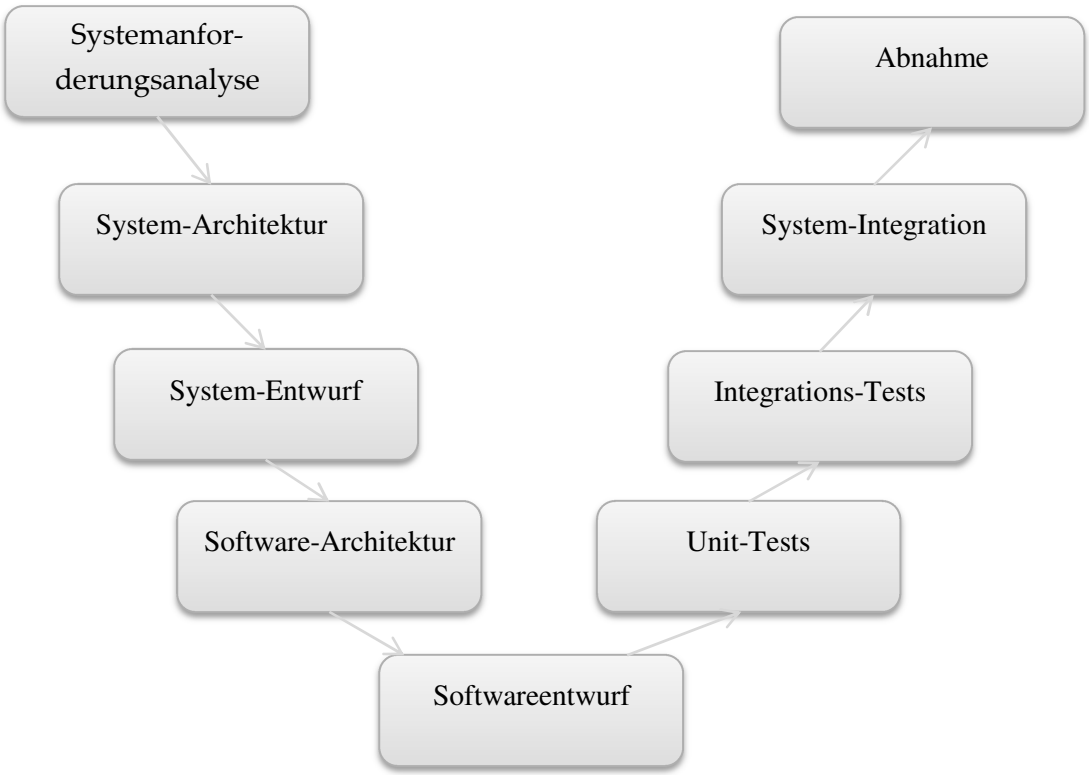


Abbildung 3-3: V-Modell

3.4 Vorteile und Nachteile von Softwaretests

In diesem Abschnitt sollen noch einmal zusammenfassend alle Vor- und Nachteile von Softwaretests aufgeführt werden.

Der größte Vorteil, der durch Softwaretests erreicht wird, ist die Verbesserung der Qualität der Software. Dabei können Qualitätsmerkmale wie Funktionalität, Effizienz, Benutzbarkeit betroffen sein. Die großen Nachteile von Tests sind einerseits der Zeitaufwand und andererseits der Kostenaufwand der entsteht. Es ist daher auch immer abzuwägen, wie viel Kosten und Zeit in die Qualität der Software gesteckt werden können. Weiterhin kann die Quantität, also der Umfang der Software eine Rolle spielen. Soll beispielsweise die Quantität, also der Umfang, eines Softwareproduktes erhöht werden, so muss entweder mehr Zeit oder mehr Kosten eingeplant werden. Der Zusammenhang, der einzelnen Faktoren, lässt sich mittels des sogenannten Teufelsquadrates sehr gut darstellen.

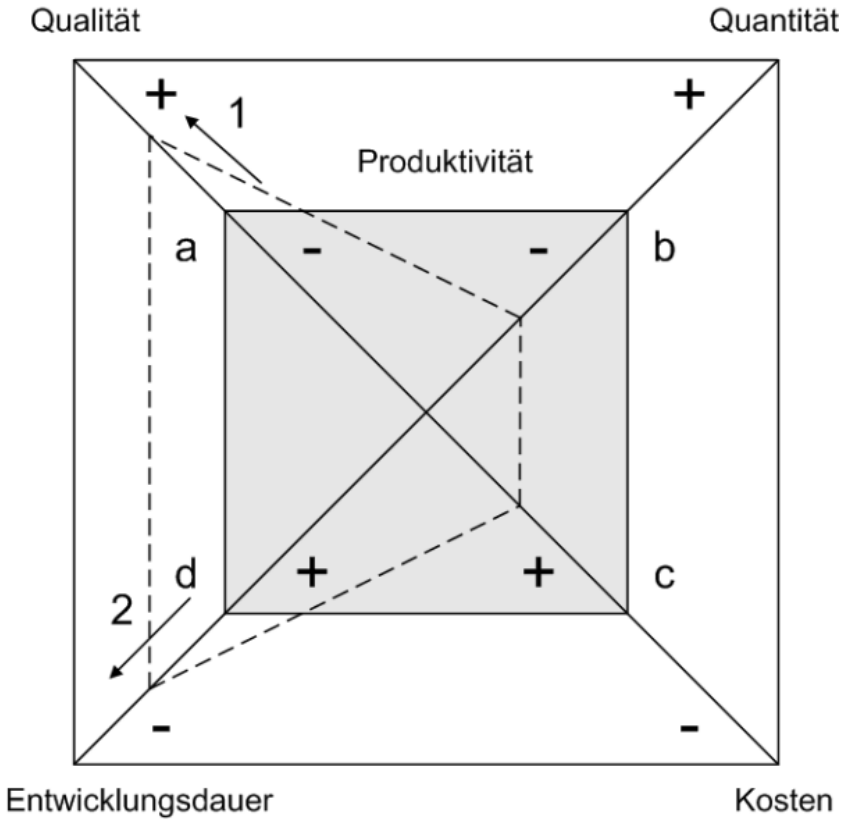


Abbildung 3-4: Teufelsquadrat, Quelle: [Pet11]

4 Arten von Softwaretests

In diesem Kapitel werden die verschiedenen Arten von Softwaretests betrachtet.

Grundsätzlich lassen sich Softwaretests in die zwei Kategorien Blackbox- und Whitebox-Tests unterteilen. Die folgenden Unterkapitel zum Blackbox- bzw. Whitebox-Test folgen den Vorschlägen aus dem Buch Software-Qualität – Hoffman [Hof08].

4.1 Blackbox-Test

Ein Test, der das Softwaresystem bzw. die Komponenten von außen überprüft, wird als Blackbox-Test bezeichnet. Bei solchen Tests sind also keine Kenntnis der inneren Strukturen des zu testenden Systems bzw. der Komponente von Nöten. Allgemeinhin werden mittels dieser Methode Eingabedaten in die Software eingegeben und die daraus resultierenden Ausgaben auf Richtigkeit überprüft. Dies erfordert, dass die zu testenden Bereiche schon nahezu komplettiert sind, da die Abläufe zur Verarbeitung der Eingabewerte bis zur Ausgabe vollständig vorhanden sein müssen. Die für den Blackbox-Test zu erstellenden Testfälle werden aus den Spezifikationen abgeleitet. Es wird also besonders überprüft, ob die Anforderungen an die Funktionalitäten des Programms erfüllt sind. Im Folgenden sollen nun verschiedene Verfahren zum Blackbox-Test gegeben werden.

4.1.1 Äquivalenzklassenbildung

Die Äquivalenzklassenbildung ist eine Testmethodik, um die Menge an Testfällen durch Klassifizierung zu verringern. Das bedeutet, dass gleichartige Testwerte in Klassen zusammengefasst werden und aus den Klassen immer nur ein beliebiger Wert zum Testen aller Werte der Klasse ausreicht. Da die Klassifizierung der Werte aus der Spezifikation abgeleitet wird und kein Wissen über die inneren Strukturen vorhanden sein muss, ist dies eine Variante des Blackbox-Tests. Je nach der Anzahl an Eingangsparameter entstehen neue Dimensionen der Klassen. Man unterscheidet zwischen eindimensionalen und mehrdimensionalen Äquivalenzklassen. Wenn auch ungültige Eingabewerte möglich sind, kann eine weitere Unterscheidung in partielle und vollständige Äquivalenzklassentests getroffen werden.

Eindimensionale Äquivalenzklassen

Bei einem eindimensionalen Äquivalenzklassentest ist nur ein Eingangsparameter vorhanden bzw. nur einer zu beachten. Im ersten Schritt werden über die Spezifikation bzw. Erfahrung die Klassen definiert. Als nächstes werden die möglichen Eingangsdaten den Klassen zugeordnet. Im letzten Schritt wird aus den jeweiligen Klassen ein Wert entnommen mit welchem dann ein Testfall durchgeführt wird.

Im Folgenden soll ein kleines Beispiel dies verdeutlichen:

In einem Programm gibt es eine Funktion „Absolute“, die einen Eingabewert verlangt. Die Funktion berechnet den Betragswert.

```
public int Absolute(int value)
{
  if (value < 0)
    return –value;
  else
    return value;
}
```

Listing 4-1: Funktion „Absolute“ zum berechnen des Betrages

Im Normallfall ist die konkrete Implementierung der Funktion, die getestet werden soll, nicht bekannt bzw. nicht relevant. Wie zu erkennen ist, wäre der Aufwand jeden möglichen Integer-Wert zu testen viel zu groß.

Im ersten Schritt der Äquivalenzklassenbildung sind die Klassen zu bilden. Schon intuitiv wird man eine Unterteilung in positive und negative Werte vornehmen. Die Null könnte dabei als positiver oder als negativer angesehen werden. Es sind nach der Bildung also zwei verschiedene Klassen vorhanden (Abbildung 4-1). Die Zuordnung der Werte in die Klassen sollte in diesem Beispiel kein Problem darstellen. Im letzten Schritt muss noch aus den jeweiligen Klassen ein Wert zum Testen der Funktion gewählt werden. Damit ist eine starke Reduktion der Testfälle erreicht worden.

Abbildung 4-1: Äquivalenzklasse der Funktion „Absolute“

Mehrdimensionale Äquivalenzklassen

Sind nun mehrere Eingabeparameter vorhanden, so entstehen auch mehrdimensionale Äquivalenzklassen. Sind zum Beispiel zwei Eingabeparameter vorhanden, so lassen sich die möglichen Klassen dann gut in Tabellen erfassen. Die folgende Funktion berechnet den Quotient zweier Eingangsparameter. Diese sind zum einen der Dividend und der Divisor. Die Funktion sieht wie folgt aus:

```
public double Divide(int dividend, int divisor)
{
  return dividend /divisor;
}
```

Listing 4-2: Funktion „Divide“ mit Divident und Divisor als Parameter

Die zu bildenden Klassen lassen sich nun sehr schön als Tabelle darstellen.

	Dividend < 0	Dividend = 0	Dividend > 0
Divisor > 0	negativ	0	positiv
Divisor = 0	Fehler	Fehler	Fehler
Divisor < 0	positiv	0	negativ

Abbildung 4-2: Äquivalenzklassen der Funktion „Divide“ mit zwei Parametern

Es lassen sich also insgesamt neun Klassen identifizieren. Es werden Klassen für die Werte kleiner Null, Größer Null und gleich Null, jeweils für Divisor und Divident, erstellt. Die Null wird in einer gesonderten Klasse erfasst, da sie bei einer Division eine Sonderstellung einnimmt. Weiterhin ist zu erkennen, dass einige der Klassen dasselbe Ergebnis hervorrufen und damit zusammengefügt werden können. Zum Beispiel führen alle Klassen bei dem der Divisor „0“ ist zu einem Fehler, unabhängig vom Dividenden. Nachdem nun die Klassen definiert sind, gilt es wieder die jeweiligen Werte zuzuordnen und aus den Klassen jeweils einen Testfall zu erstellen, mit dem Überprüft wird, ob das richtige Ergebnis geliefert wird.

Partielle Äquivalenzklassentests

Falls ungültige Werte für die zu testende Funktion bzw. den Bereich bekannt sind, so werden diese bei der partiellen Äquivalenzklassenbildung außen vorgelassen. Solche ungültigen Werte treten dann auf, wenn nur bestimmte Bereiche der Eingabe definiert sind, zum Beispiel wenn Zahlen feste Werte zugeordnet sind (Enums). Es werden dabei also nur die gültigen Werte geprüft. Dies wird im Normalfall dann getan, wenn die ungültigen Werte so nie auftauchen können oder die Spezifikation keine Aussagen über die möglichen Ausgänge trifft. Im oberen Beispiel ist der Divisor an der Stelle „0“ zwar unzulässig im Sinne, dass die Funktion einen Fehler liefert, jedoch geht aus der Spezifikation hervor, dass die Division fehlerhaft sein muss. Es sind also Aussagen über den Ausgang möglich und es ist denkbar, dass dieser Wert eingegeben wird.

Vollständige Äquivalenzklassentests

Bei vollständigen Äquivalenzklassentests werden auch die ungültigen Werte mit in den Test aufgenommen. Dabei werden für diese ungültigen Werte, wie auch bei den gültigen, Klassen gebildet und diesen Klassen wieder die passenden Werte zugeordnet, um dann daraus Testfälle zu erstellen.

4.1.2 Grenzwertanalyse

Die Grenzwertanalyse ist eine Erweiterung der Äquivalenzklassenbildung. Da sich die meisten Fehler an den Grenzen der Äquivalenzklassen befinden, werden bei der Grenzwertanalyse die Testfälle aus genau diesen Werten gebildet. Das heißt, die Bildung der Klassen erfolgt analog zur Äquivalenz-klassenbildung und auch das Zuordnen der Werte erfolgt nach derselben Vorgehensweise. Bei dem Herausfiltern der Werte für den Testfall werden aber nicht willkürliche Eingabewerte aus der

Klasse gewählt, sondern diejenigen die sich am Rand der Klasse befinden. Genauer gesagt werden an den Grenzen der Klassen ein Wert am inneren Rand und ein Wert am äußeren gewählt.

Dies soll an der Funktion „Divide“ die zwei Parameter, Dividend und Divisor, braucht verdeutlicht werden. Die Bildung der Klassen erfolgt wie schon erwähnt analog zur Äquivalenzklassenbildung. Die Grenzwerte werden dann an den Übergängen zwischen den Klassen, aber auch an den äußeren Grenzen gewählt. Bei den äußeren Grenzen entsteht das Problem der offenen Grenzen, da prinzipiell keine Einschränkungen in der Hinsicht gemacht wurden. Es müssen daher künstliche Schranken gewählt werden. In dem Fall ließe sich zum Beispiel der minimale bzw. maximale Integer-Wert als äußere Grenzen verwenden.

Problem der Grenzwertanalyse ist, dass die Werte immer eine Ordnung haben müssen. Bestehen die Eingangsparameter beispielsweise aus Strings so ist nicht immer klar wo die Grenzen liegen.

4.1.3 Zustandsbezogene Tests

Zustandsbezogene Tests werden genutzt, wenn verschiedene Zustände im System möglich sind, die getestet werden sollen. Zustandsbezogene Tests sind also nutzbar, wenn der bisherige Systemablauf neben den Eingabewerten einen Einfluss auf das Ergebnis bzw. den Verlauf des Tests hat. Bei zustandsbezogenen Tests werden Zustandsautomaten erstellt, die den Zustand als auch die Übergange zu den nächsten Zuständen erfassen. Aus den Übergängen eines Zustands in einen anderen werden dann Testfälle abgeleitet. Dies wird für alle vorhandenen Zustandsübergänge durchgeführt. Für die Eingabewerte die je nach Zustand möglich sind, können wieder die Methoden der Äquivalenzklassenbildung oder der Grenzwertanalyse herangezogen werden.

4.1.4 Use-Case Test

Diese Testmethode basiert auf der Nutzung von Anwendungsfällen (Use Case). Dabei ist ein Anwendungsfall eine Aktion bzw. eine Folge von Aktionen die durch den Nutzer des Systems ausgelöst werden. Die Beschreibung der Aktionen erfolgt aus Sicht des Nutzers, also eine nicht-technische Ansicht. Anwendungsfälle und sogenannte Anwendungsfalldiagramme, die solche Fälle im Zusammenhang mit dem Akteur (Nutzer) aufzeigen, sind durch die UML-Spezifikation genau beschrieben. Es sind nun für alle Anwendungsfälle Tests zu erstellen. Dabei werden die Eingabewerte, welche für die jeweiligen Anwendungsfälle möglich sind, hergenommen um diese dann mit den am Anfang vorgestellten Mitteln, Äquivalenzklassenbildung und Grenzwertanalyse, zu testen. Diese Testmethode ist eine der wichtigsten Blackbox-Testtechniken, da sie ein einfaches Erstellen der

Testfälle aus den Anwendungsfällen heraus ermöglicht und Systemtests geeignet ist.

4.1.5 Testen mittels Entscheidungstabellen

Das Testen mittels Entscheidungstabellen basiert auf dem Ursache-Wirkungs-Prinzip. Es wird also eine Tabelle aufgebaut, in welcher Bedingungen festgehalten werden, die Aktionen im System auslösen. Weiterhin werden die Aktionen, die aus den jeweiligen Bedingungen resultieren, eingetragen. Das folgende Beispiel soll dies verdeutlichen:

Testfälle																
1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	
Bedingungen																
J	J	J	J	J	J	J	J	N	N	N	N	N	N	N	N	Konto aktiv
J	J	J	J	N	N	N	N	J	J	J	J	N	N	N	N	Geld vorhanden
J	J	N	N	J	J	N	N	J	J	N	N	J	J	N	N	Gültige Tan
J	N	J	N	J	N	J	N	J	N	J	N	J	N	J	N	Karte gültig
Aktionen																
J	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	Überweisung tätigen
J	N	J	N	J	N	N	N	N	N	N	N	N	N	N	N	Einloggen
N	J	N	J	N	J	N	J	N	J	N	J	N	J	N	J	Konto sperren

Abbildung 4-3: Entscheidungstabelle für Online-Banking

Dies ist eine Entscheidungstabelle zum Online-Banking. Im oberen Teil befinden sich alle Bedingungen im unteren alle Aktionen. Ein „J“ bedeutet, dass die Bedingung erfüllt bzw. die Aktion ausgelöst wird, ein „N“ besagt das Gegenteil. Es kann auch vorkommen, dass statt booleschen Ausdrücken, Wertebereiche zum Erstellen der Tabellen genutzt werden. Schon bei diesem kleinen Beispiel wird die Tabelle relativ groß. Wendet man diese Methodik auf reale Probleme an, so wächst die Tabelle schnell in eine unüberschaubare Größe. Deshalb gilt es, Bedingungen, die gleiche Aktionen hervorrufen, zusammenzufassen. Wie beispielsweise zu erkennen ist, ruft eine ungültige Karte unabhängig von den anderen Bedingungen immer dieselben Aktionen hervor. Daher kann man all diese zu einer Spalte und damit einem Testfall zusammenschmelzen. Eine gekürzte Variante des oberen Beispiels könnte wie folgt aussehen:

Testfälle					
1	2	3	4	5	
Bedingungen					
N	J	J	J	J	Karte gültig
-	N	J	J	J	Konto aktiv
-	-	N	J	J	Geld vorhanden
-	-	-	N	J	Gültige Tab
Aktionen					
N	N	N	N	J	Überweisung tätigen
N	N	J	J	J	Einloggen
J	N	N	N	N	Konto sperren

Abbildung 4-4: Verringerte Entscheidungstabelle für Online-Banking

Mit dieser neuen Tabelle wurden also einige Testfälle gespart, jedoch können auch mit der verkleinerten Version je nach Komplexität sehr große Tabellen entstehen.

Nachdem nun die Tabelle soweit generiert wurden, können die Testfälle aus den Bedingungen erstellt werden und es kann geprüft werden, ob die Ausführung des Programms dieselben Aktionen hervorruft, wie in der Tabelle angegeben.

4.2 Whitebox-Test

Whitebox-Tests sind das Gegenstück zu den Blackbox-Tests. Sie haben Zugriff auf den Quellcode und sind gerade dafür da, die inneren Strukturen zu testen bzw. zu analysieren. Es soll dabei überprüft werden inwiefern Tests bzw. Testfälle den Quellcode ausführen. Denn wenn ein Programmteil nicht ausgeführt wird, so kann auch kein Fehler in diesem gefunden werden. Man kann dabei entweder schon vorhandene Testfälle aus den Blackbox-Tests übernehmen oder neue Testfälle erstellen, die eine möglichst gute Abdeckung erreichen. [Kur07, S. 104] Bei Whitebox-Test spielen Kontrollflussgraphen eine zentrale Rolle. Ein Kontrollflussgraph zeigt den Kontrollfluss innerhalb bzw. über mehrere Funktionen hinweg. Mit Hilfe dessen können die Tests durchgeführt werden. Nachfolgend sollen die drei wichtigsten Whitebox-Testtechniken vorgestellt werden.

4.2.1 Anweisungsüberdeckungstest

Der Anweisungsüberdeckungstest sollte alle Anweisungen eines Programms einmal durchlaufen. Dabei wird eine vollständige Abdeckung angestrebt, was bedeutet, dass wirklich jede Anweisung mindestens einmal durchgeführt wird. Im Kontrollflussgraphen müssen daher alle Knoten durchlaufen werden. Das folgende Beispiel soll den Sachverhalt verdeutlichen. Dazu soll wieder das Beispiel „Divide“ aus der Äquivalenzklassenbildung hergenommen werden. Die Funktion wird so erweitert, dass die Betragswerte dividiert werden.

```
public double DivideAbsolute(int dividend, int divisor)
{
```

```
    if(divisor < 0)
      divisor =- divisor;

    if(dividend < 0)
      dividend =- dividend;
    return dividend /divisor;
  }
```

Listing 4-3: Funktion „DivideAbsolute“

Der daraus resultierende Kontrollflussgraph sieht folgendermaßen aus:

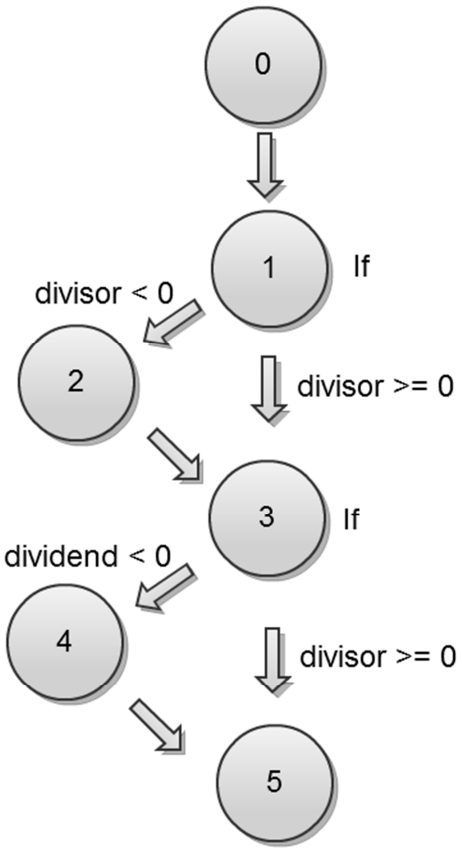


Abbildung 4-5: Kontrollflussgraph für Funktion „DivideAbsolute“

Wie zu erkennen ist, gibt es einen definierten Start- und Endpunkt. In einer If-Anweisung findet eine Aufteilung im Graphen, die die If-Anweisung repräsentiert, statt. Wird ein Divisor kleiner Null und ein Dividend kleiner Null gewählt, so werden alle Knoten im Kontrollflussgraphen und damit alle Anweisungen durchlaufen. Ein Test der anderen Zweige findet aber nicht statt. Des Weiteren entfällt damit auch das Testen positiver Werte komplett. Es reicht also ein einziger Testfall mit negativen Werten aus. Wie daraus zu erkennen ist, ergibt sich das Problem, dass eine komplette Anweisungsüberdeckung mit einer kleinen Menge an Testfällen zu erreichen ist. Allerdings werden damit im Normallfall wichtige Fälle nicht berücksichtigt, wie in dem Beispiel die positiven Werte. Im Allgemeinen ist der Anweisungsüberdeckungstest die Minimalanforderung an diese Art von Tests.

4.2.2 Zweigüberdeckungstest

Der Zweigüberdeckungstest durchläuft alle Kanten eines Kontrollflussgraphen. Es werden damit auch alle Knoten durchlaufen. Dadurch ist der Anwendungsüberdeckungstest quasi mit in diesem integriert. Um für die „DivideAbsolute"-Funktion eine hundertprozentige Abdeckung zu gewährleisten, reicht ein Testfall nicht mehr aus. Es müssen für jeden Parameter einmal ein negativer und einmal ein positiver Wert übergeben werden. Das heißt es sind wenigstens zwei Testfälle notwendig.

4.2.3 Pfadüberdeckungstest

Beim Pfadüberdeckungstest werden prinzipiell alle möglichen Pfade, die vorhanden sind durchlaufen. Dies führt auch schon bei kleineren Programmen zu einem enormen Aufwand. Gerade bei Schleifendurchläufen können Schwierigkeiten entstehen, da es unendlich viele Pfade geben kann. Um zumindest das Problem der unendlichen vielen Pfade zu lösen, gibt es Abwandlungen dieser Testmethode, welche nur eine begrenzte Anzahl an Schleifendurchgängen vorsehen. Dennoch ist diese Testart aufgrund des insgesamt zu hohen Aufwands kaum relevant.

4.3 Einordnung von Blackbox- bzw. Whitebox-Tests in Testphasen

Neben der Kategorisierung von Tests in Blackbox- bzw. Whitebox-Tests lassen sich auch verschiedene Teststufen festlegen in welchen diese Testarten eingeordnet werden können. Die Teststufen sind dabei der Komponententest, der Integrationstest, der Systemtest und der Abnahmetest.

4.3.1 Komponententest

Der Komponententest ist die Teststufe in der einzelne Komponenten und Funktionen getestet werden. Er kommt in der Entwicklung am frühesten vor, da zu Anfang nur die Funktionen einzeln getestet werden können. In dieser Phase kommen die Whitebox-Testtechniken zum Einsatz, da diese die Komponenten testen können, auch wenn die einzelnen fachlichen Funktionalitäten noch nicht komplett implementiert sind und bei diesen Tests eine enge Zusammenarbeit mit den Entwicklern erforderlich ist. [Kla07] Auch die Blackbox-Testtechniken, Äquivalenzklassenbildung und Grenzwertanalyse können zum Einsatz kommen. Weiterhin ist es möglich, an geeigneten Stellen zustandsbasierte Tests durchzuführen.

4.3.2 Integrationstest

Integrationstests werden durchgeführt, wenn mehrere Komponenten zusammengefügt werden. Es wird also insbesondere das Zusammenspiel der einzelnen Komponenten getestet. Bei Integrationstests sind auch die Blackbox-Testtechniken Äquivalenzklassenbildung, Grenzwertanalyse und zustands-basierte Tests im Einsatz, da nach der Zusammenführung, einerseits die Komponenten für

sich und andererseits die Schnittstellen getestet werden müssen. Des Weiteren können auch Whitebox-Testmethoden zum Einsatz kommen, wobei durch die Nutzung dieser Testtechnik im Komponententest schon die meisten Fehler, die durch diese aufgedeckt werden können, erkannt werden sollten.

4.3.3 Systemtest

Bei einem Systemtest werden größere Abschnitte des Programms bzw. das gesamte System getestet. Die Tests werden dabei mit einem möglichst realistischen Ablauf durchgeführt. Dies erfordert, dass das Programm bzw. die Abschnitte, die getestet werden sollen, voll funktionsfähig sind. Im Systemtest sind am ehesten die Blackbox-Tests, Use-Case Test und entscheidungstabellen-basierte Tests anzutreffen, denn diese Methoden betrachten das Programm auf Systemebene.

4.3.4 Abnahmetest

Der Abnahmetest ist der letzte Test vor der Übergabe des Produktes an den Kunden. Diese Tests finden also mit der Auslieferung des Softwareproduktes statt. Dabei geht es noch einmal darum, zu überprüfen, ob die Softwarequalität im gewünschten Maße erreicht wurde. Es ist prinzipiell auch ein Systemtest. Der Abnahmetest wird, wie auch der Systemtest, mit Blackbox-Testtechniken durchgeführt, da hauptsächlich die Vollständigkeit und Richtigkeit der verlangten Funktionalitäten überprüft werden soll.

4.3.5 Regressionstest

Bei der Entwicklung und Wartung eines Softwareproduktes kommt es oft vor, dass vorhandene Funktionen abgeändert und damit wieder getestet werden müssen. Zum Beispiel werden bei der Wartung eines Softwaresystems, Erweiterung erstellt oder im Nachhinein gefundene Fehler behoben. Die Komponenten, die von den Änderungen betroffen sind und deren abhängige Komponenten, müssen dann wieder getestet werden. Dies wird als Regressionstest bezeichnet. Je nachdem, was für Änderungen und in welchem Umfang die Änderungen vorgenommen wurden, können alle Testmethoden zum Einsatz kommen. Wird zum Beispiel im Zuge einer Erweiterung eine neue Komponente entwickelt, sollte sowohl ein Komponententest als auch nach der Integration in das Gesamtsystem ein Integrationstest stattfinden. Auch ein anschließender Systemtest ist denkbar.

5 Einführung in das automatische Testen von Software

Das folgende Kapitel soll eine Einführung in das automatische Testen liefern. Dabei wird zuerst die Definition betrachtet, um dann Unterschiede bzw. Vor- und Nachteile zum manuellen Testen zu geben.

5.1 Definition

Testautomatisierung ist der Vorgang bei dem versucht wird Testfälle möglichst ohne die Interaktion eines Nutzers durchzuführen. Beim automatischen Testen läuft also ein Programm welches die Testfälle gegen das zu testende System ausführt. Dabei sollte das Testprogramm die folgenden Voraussetzungen erfüllen.

- Der Test sollte ohne Interaktion durch den Nutzer ablaufen.
- Falls im zu testenden System spezifische Einstellungen für den Test zu treffen sind, sollte dies, wenn möglich, durch das Testtool erledigt werden.
- Ein Aufzeichnen aller wichtigen Aktionen sollte durch das Testtool erfolgen.
- Eine Überprüfung der erwarteten gegenüber den eigentlichen Werten sollte durch das Tool ermöglicht werden.
- Nachdem ein Testfall durchlaufen ist, sollte aufgezeigt werden, ob der Testfall generell ohne oder mit Fehlern beendet wurde.

Wie aus den Voraussetzungen zu erkennen ist, ist es für eine Testautomatisierung nicht zwingend notwendig die Testfälle selbst automatisch zu generieren. Gerade wenn man an Blackbox-Tests denkt, so ist ein automatisches Generieren von Testfällen kaum möglich, da die Testfälle aus der Spezifikation abgeleitet werden und das Testtool diese sicherlich nicht interpretieren kann. Bei Whitebox-Tests hingegen ist eine automatische Testfallgenerierung eher denkbar, indem die Codestrukturen automatisch analysiert werden, daraus die Kontrollflussgraphen erstellt werden, um dann die Eingabewerte zu generieren.

5.2 Abgrenzung zum manuellen Testen

Beim manuellen Testen werden die Testfälle durch den Nutzer selbst abgearbeitet. Das heißt, die jeweiligen Eingabedaten werden durch den Nutzer eingegeben, die ausgelösten Aktionen werden durch den Nutzer überwacht und im Nachhinein ausgewertet. Diese Dinge sollten beim automatischen Testen natürlich ohne Eingriff des Nutzers geschehen. Es gibt allerdings auch Zwischenstufen. Zum Beispiel kann es vorkommen, dass ein Test prinzipiell automatisch abläuft, jedoch bestimmte Aktionen durch den Nutzer bestätigt oder durchgeführt werden müssen. Dies birgt den großen Nachteil, dass man während des Tests anwesend sein muss, um an den notwendigen Stellen die Aktionen durchzuführen.

Der große Vorteil automatischer Tests ist die potentielle Kosten- und Zeitersparnis. Erfüllt das Testtool idealerweise alle genannten Voraussetzungen und bringt die nötige Stabilität mit, so kann man die Testfälle prinzipiell unbeaufsichtigt laufen lassen. Ein weiterer Vorteil des automatischen Testens ist, dass ein erstellter Testfall immer wieder durchgeführt werden kann, solange keine Änderungen im zu testenden System, an den betroffenen Stellen, vorgenommen werden.

Ein Problem ist, dass eine hundertprozentige Fehlerfreiheit des Testtools nie gewährleistet werden kann. Es ist also immer möglich, dass ein Testfall durch einen unerwarteten Fehler abbricht und so im schlimmsten Fall der ganze Test nicht weiter durchgeführt wird. Man sollte sich also nicht gänzlich auf das fehlerfreie Funktionieren verlassen und zumindest eine zeitweise Überwachung in Betracht ziehen. Weiterer Nachteil an der Testautomatisierung ist, dass natürlich ein geeignetes Testtool zum automatischen Testen vorhanden sein muss. Daher muss entweder ein fertiges Tool gekauft werden, diese sind allerdings meist sehr teuer, ein Freeware Tool genutzt werden, welche allerdings meist nicht alle benötigten Funktionalitäten mitbringen, oder ein eigenes Tool entwickelt werden. Man hat deshalb, bevor man überhaupt automatisch Testen kann, einen relativ großen Aufwand, um ein Testtool zu haben, welches für die Anwendung optimal funktioniert.

5.3 Anwendungsgebiete

Generell ist eine Testautomatisierung dann angebracht, wenn gleiche Tests oft wiederholt werden müssen. Weiterhin sind automatisierte Test dann nützlich, wenn Änderungen am zu testenden System keine oder nur geringe Auswirkungen auf das Testtool oder die Testfälle haben bzw. das Testtool auf die Änderungen geeignet reagieren kann.

Im Normallfall werden Whitebox-Tests automatisiert ausgeführt, da der Aufwand des manuellen Testens viel zu groß wäre, denn man müsste jede Funktion des Programms manuell mit allen benötigten Eingabewerten ausführen. Da dieser Vorgang prinzipiell unabhängig von der Spezifikation des Programms abläuft, lassen sich generelle Tools erstellen, welche für jede Art von Programm unter einer festgelegten Programmiersprache funktionieren. Beispielsweise bieten bestimmten Versionen der integrierten Entwicklungsumgebung „Visual Studio“ solche Funktionalitäten an.

Bei Blackbox-Tests ist es eher wahrscheinlich, dass Testfälle oder im schlimmsten Falle ganze Testtools nicht mehr zu verwenden sind bzw. abgeändert werden müssen. Gerade wenn im Zuge von Use-Case-Tests oder ähnlichem ganze Systemabschnitte getestet werden, können Änderungen in diesen Probleme verursachen. Soll beispielsweise eine Neuerung eingefügt werden oder

Änderungen an vorhandenen Funktionalitäten vorgenommen werden, hat dies oft zur Folge, dass die Testfälle nicht mehr ordnungsgemäß funktioniert. Gerade deshalb ist es in der Testautomatisierung von Blackbox-Tests ein großes Ziel Testfälle einfach und leicht wartbar zu halten, um den Aufwand bei notwendigen Änderungen gering zu halten.

Automatisierte Blackbox-Tests bieten sich besonders bei Regressionstest an. Beim Einbau neuer Funktionalitäten bzw. bei Änderungen müssen auch die abhängigen Funktionen erneut überprüft werden. Dabei ist zu testen ob diese noch genauso funktionieren wie vorher. Es werden gleiche Tests wiederholt ausgeführt. Weiterhin bietet sich eine Automatisierung solcher Blackbox-Tests bei Komponententests an. Da bei dieser Methode nur die einzelnen Funktionen getestet werden und die Eingabedaten und Resultate sich nur selten ändern bzw. die Abänderung der Testfälle einen nicht zu großen Aufwand nach sich ziehen.

6 Automatisches Testen von Webanwendungen

In diesem Kapitel soll eine ausführliche Betrachtung zum automatischen Testen von Webanwendungen erfolgen. Dabei stehen Blackbox-Tests im Fokus. Zu Anfang sollen Besonderheiten zum Testen von Webapplikationen betrachtet werden. Dann werden die Web-Technologien, welche für das automatische Testen von Bedeutung sind, vorgestellt. Im Weiteren werden dann Vorgehensweisen zum automatischen Testen aufgezeigt. Dabei sollen Beispiele diese Herangehensweisen veranschaulichen. Danach erfolgt eine Betrachtung verschiedener Testtools, im Besonderen das Tool „Selenium“. Zuletzt werden Konzepte zum Aufbau eines eigenen Testtools unter Nutzung von Selenium aufgezeigt. Dabei werden verschiedene Probleme und dazu passende Lösungsansätze genannt.

6.1 Besonderheiten

Im Wesentlichen gibt es beim Testen von Webapplikationen zwei Besonderheiten zu beachten. Dazu gehört einerseits, dass die Oberfläche aus HTML und CSS aufgebaut ist und dass zum anderen das Benutzen der Applikation immer eine Kommunikation zwischen Client und Server erfordert bzw. Teile der Applikation auf dem Server und andere Teile auf dem Client ablaufen.

6.1.1 Besonderheit: HTML und CSS

Will man zum Beispiel im Zuge von Oberflächen- oder Blackbox-Tests automatisch auf die Oberfläche des Systems zugreifen und mit dieser Interagieren, so ist immer zu beachten, dass diese aus HTML und CSS besteht. Es müssen also andere Methoden zum automatischen Zugriff auf die Elemente in einer Webapplikation genutzt werden, als dies zum Beispiel bei Desktopapplikationen der Fall ist.

6.1.2 Besonderheit: Client-Server Architektur

Die Client-Server Architektur ist ein Modell welches bei Webseiten, Webapplikationen, Webservices und ähnlichem genutzt wird. Es dient der Kommunikation zwischen Client und Server. Der Server ist dabei in ein Netzwerk eingebunden und kann verschiedene Dienste bereitstellen. Ein solcher Dienst kann zum Beispiel eine Webapplikation sein. Möchte ein Client nun auf die Webapplikation zugreifen, so sendet dieser eine Anforderung (Request) an den Server, welcher diesen verarbeitet und dann eine Antwort (Response) zurück sendet. Bei einer Webapplikation wird die Antwort eine HTML-Seite mit entsprechenden Informationen sein.

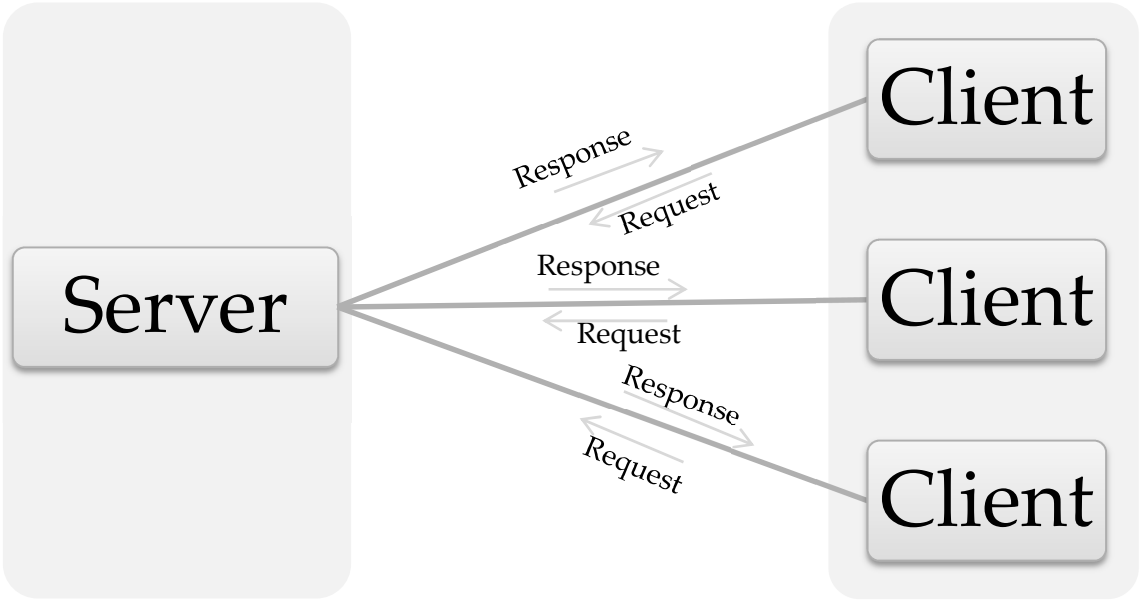


Abbildung 6-1: Client-Server Architektur

Wie zu erkennen ist, ermöglicht eine solche Architektur, dass vielen Clients ein solcher Dienst zur Verfügung gestellt werden kann.

Wenn man nun Tests durchführen möchte ist also bei Blackbox-Tests zu beachten, dass die Eingabewerte erst über den Client an den Server geschickt, auf dem Server ausgewertet und die Resultate dann an den Client zurückgeschickt werden. Die Überprüfung von Werten kann also erst nachdem all diese Aktionen durchgeführt wurden erfolgen bzw. können weitere Eingaben erst erfolgen, nachdem die Response vom Server angekommen ist.

6.2 Wichtige Web-Technologien für das automatische Testen von Webapplikationen

Im Folgenden sollen die Web-Technologien, die für das automatische Testen von Webapplikationen relevant sind, kurz erläutert werden.

HTML

HTML ist eine Sprache zum Strukturieren von Texten. Zusätzlich bietet es die Möglichkeit Bilder und andere multimediale Inhalte zu referenzieren, wie auch verschiedene Oberflächenkomponenten einzubinden. Es wird hauptsächlich zum Anzeigen von Webseiten verwendet. HTML basiert auf der SGML (Standard Generalized Markup Lanugage), welche durch die ISO-Norm 8879 definiert ist. [SEL11] Die Formatierung der Texte bzw. der multimedialen Inhalte erfolgt mittels CSS. HTML dient in den meisten Fällen als Schnittstelle zum automatischen Testen. Es werden dabei automatisch Daten in die Komponenten eingetragen bzw. Aktionen ausgeführt.

JavaScript

JavaScript ist hauptsächlich dazu gedacht die statischen HTML-Inhalte dynamischer zu gestalten. Es ermöglich also die Manipulation dieser Inhalte. Dabei können sowohl vorhandene Inhalte verändert werden als auch neue Inhalte hinzugefügt werden. Es ist auch möglich auf verschiedene Nutzeraktionen mittels JavaScript zu reagieren. Zum Beispiel auf die Eingabe in ein Textfeld. Die Interaktion mit HTML-Elementen ermöglicht somit, eine Automatisierung.

AJAX

AJAX (Asynchrounus JavaScript and XML) ermöglicht das Senden von Requests und Empfangen von Responses ohne dass die jeweilige HTML-Seite neu geladen werden muss. Um dies zu bewerkstelligen wird mittels JavaScript ein Request erzeugt. Dieser wird dann zum Server gesendet und von diesem verarbeitet. Danach wird die Response an den Client zurückgesendet. Diese ist aber meist keine HTML-Seite, sondern nur neue Daten die angezeigt werden sollen. Zur Übertragung hat sich deshalb JSON (JavaScript Object Notation), ein kompaktes Format zum Übermitteln von Daten, durchgesetzt. Es wird bevorzugt genutzt, da die Dateigröße geringer gehalten wird und die Daten JavaScript-Konform gespeichert werden. Das heißt, dass die Daten beim Client von JavaScript interpretiert werden können. Nutzt eine Webapplikation AJAX, muss dies bei der Automatisierung beachtet werden, weil damit auch Daten übertragen werden und durch die Übertragung Wartezeiten entstehen.

6.3 Vorgehensweisen zum automatischen Testen von Webanwendungen

Es lassen sich drei verschiedenen Methoden zum automatischen Testen von Webanwendungen in Form von Blackbox-Tests benennen:

- Generieren von Requests an den Server
- Einfügen von Daten in Formulare mittels JavaScript
- Nutzung von Mechanismen des Internet-Browsers

All diese werden in den folgenden Unterkapiteln näher erläutert.

6.3.1 Generieren von Requests

Request sind Anfragen, die von einem Client an einen Server gesendet werden. Zum Beispiel werden Daten, die in eine Webseite eingetragen wurden, in Form eines Requests an den Server gesendet, welcher diese auswerten kann. Das Konstrukt welches in der HTML-Seite zum Senden von Requests mit Daten genutzt wird, besteht standardmäßig aus dem HTML-Tag „Form“, einem Absende-Button und weiteren optionalen Eingabefeldern. Das Besondere an diesem Konstrukt ist

nun, dass mit dem Request auch die durch die Eingabefelder eingegebenen Werte mitgesendet werden. Im Folgenden ein Beispiel einer HTML-Seite, welche ein Formular enthält.

```
<html>
 <head>
 </head>
 <body>
  <form action="send.html" method="post" >
       <input type="text" name="myText" />
      <input type="checkbox" name="myCheckbox" />
      <input type="submit" name="mySubmitButton"
value="Absenden" />
  </form>
 </body>
</html>
```

Listing 6-1: Eine einfache HTML-Seite die eine Form enthält

Wie zu sehen ist, sind in dem Form-Tag die zwei Attribute „action“ und „method“. Mit dem Attribut „action“ wird die Seite angegeben auf die navigiert werden soll. In dem Attribut „method“ wird die HTTP-Request-Methode angegeben. Hierbei werden in den meisten Fällen die Methoden „post“ und „get“ genutzt. Es gibt jedoch noch weitere Methoden wie „delete“ oder „put“. Innerhalb des Form-Tags sind nun verschiedene Eingabe-Tags vorhanden, mit denen der Nutzer auf der Webseite interagieren kann. Das erste ist dabei ein einfaches Eingabefeld, in welchem der Nutzer Text eingeben kann. Dies wird durch das Attribut „type“ angegeben. Dieses ist für Eingabefelder „text“. Als nächstes befindet sich in dem Form-Tag eine Auswahlbox, auf die der Nutzer klicken kann. An letzter Stelle ist der Absenden-Button der durch den Typ „submit“ kenntlich gemacht wird. Drückt der Nutzer nun diesen Button, wird er Request ausgelöst. Wie weiterhin zu sehen ist, besitzt jedes der Eingabe-Felder einen Namen. Mittels diesem kann bei der späteren Auswertung auf dem Server nachvollzogen werden, welcher Wert zu welchem Eingabe-Element gehören. Das heißt, die Namen werden mit den Werten an den Server gesendet.

Im Folgenden die Request-Methoden „get“ und „post“ vorgestellt:

Get-Request

Bei einem sogenannten Get-Request werden die Eingabewerte in Form von Parametern an die aufzurufende URL angehängt. Beim Server angekommen ist es dann möglich, die Werte mittels einer serverseitigen Programmiersprache aus der URL zu filtern, wobei dies meist schon automatisch getan wird, und auszuwerten. Würde das obige Beispiel die Methode „get“ nutzen, könnte die daraus resultierende URL folgendermaßen aussehen.

```
http://www.mysite.de/send.html?myText=myTextfieldValue&myChe
ckbox=on&mySubmitButton=Absenden
```

Am Anfang steht die Adresse zur Webseite, gefolgt von dem Pfad zur HTML-Seite „send.html“. Nach dem Fragezeichen folgen die Parameter, die eingegeben wurden. Als Erstes findet man in diesem Beispiel „myText“, was der Namen des ersten Eingabefeldes war. Nach dem „=“ wird der zum Namen gehörige Wert angegeben. In dem Fall wurde der Text „myTextfieldValue“ eingegeben. Mittels des Ampersand werden die Parameter voneinander getrennt. Als nächstes folgt die Auswahlbox, die angeklickt wurde, also auf „on“ steht. Als letzter Parameter schließt sich der Absende-Button an. Wie zu sehen ist, wird hier als Wert der Text aus dem Attribut „value“ angegeben.

Es ist auch möglich solche URLs mit Parametern statisch selbst anzugeben. Man könnte also auch einen einfachen Link erstellen, welcher auf die Seite verweist und fest die Parameter eingetragen hat.

Der große Nachteil dieser Methode ist, dass der Nutzer beim Absenden des Requests im Browser immer erkennen kann, welche Parameter mit übergeben wurden, da diese in der ausgerufenen Adresse wiederzufinden sind. Gerade bei der Abfrage von Passwörtern kann so etwas natürlich große Probleme mit sich bringen, da es ohne weiteres möglich ist, die URL abzufangen und so die Daten zu lesen. Nicht zuletzt ist deswegen die bevorzugte Request-Methode, um Daten mittels Requests zu senden, die Post-Methode.

GET-Requests erzeugen:

Ein Get-Request lässt sich sehr leicht erzeugen, denn man muss einfach nur an die gewünschte URL die jeweiligen Parameter hängen. Im folgenden Beispiel wird ein GET-Request mittels der Programmiersprache C# erzeugt, welches dem aus Beispiel Listing 6-1 entspricht, wenn man dabei als Methode „get“ anstatt „post“ wählen würde. Vom Prinzip her ist dies natürlich auch auf andere Programmiersprachen übertragbar.

```
public class GetRequest
{
  public static void main(String[] args)
  {
     String parameter =
"myText=myTextfieldValue&myCheckbox=on&mySubmitButton=Absend
en";

     WebRequest webRequest =
WebRequest.Create("http://www.mysite.de/send.html?" +
parameter);
```

```
        StreamReader reader = new
    StreamReader(webRequest.GetResponse().GetResponseStream());
        Console.Write(reader.ReadToEnd());
      }
    }
```

Listing 6-2: Klasse um mittels Java einen Get-Request zu erzeugen

Im ersten Schritt werden die Parameter in einen String gespeichert. Sollten die Parameter variabel sein bzw. Sonderzeichen enthalten, so sollten diese mittels der Methode „UrlEncode“ der Klasse „HttpUtility“ entsprechend enkodiert werden. Als nächstes wird ein Objekt der Klasse „WebRequest“ angelegt. Dieses bekommt als Parameter den kompletten String zur auszurufenden Seite inklusive der Eingabewerte. Mittels „webRequest.GetResponse()“ wird die neue Seite aufgerufen, dann die Antwort als Stream in den StreamReader geschrieben und letztendlich auf der Konsole ausgegeben. Aus Übersichtlichkeitsgründen wurden die nötigen import-Anweisungen weggelassen.

Post-Request

Bei einem Post-Request werden die Informationen nicht an die URL angehängt, sondern es wird im Request ein extra Bereich geschaffen in welchem die Daten geschrieben werden. Dieser Bereich nennt sich „message body“. Mit dieser Methode ist es einerseits möglich, einfache Formulardaten zu senden, aber prinzipiell auch jede andere Form von Daten, zum Beispiel Bilder. Damit der Server die Daten verarbeiten kann, muss zusätzlich noch der Inhaltstyp (Content-Type) und die Länge der Daten (Content-Length) angegeben werden. Der Content-Type ist standardmäßig „application/x-www-form-urlencoded“. [Jam11] Dies ist ein spezielles Format zum Übertragen der Daten. Dabei werden die Daten in Schlüssel-Wert-Paaren angeordnet. Die Schlüssel werden über ein „=“ von den Werten getrennt und die Schlüssel-Wert-Paare werden mittels des Ampersand getrennt. Weiterhin werden Leerzeichen und Umlaute durch konforme Zeichen ersetzt. Nach der Enkodierung ist das Format dem eines Get-Requests sehr ähnlich. Würde die Daten des Formulars aus Listing 6-1 als Post-Request gesendet, so könnte der HTTP-Body[1] wie folgt aussehen.

```
POST /send.html HTTP/1.1
HOST: www.mysite.de
Content-Type: application/x-www-form-urlencoded
Content-Length: 61

myText=myTextfieldValue&myCheckbox=on&mySubmitButton=
Absenden
```

[1] Der HTTP-Body ist das Konstrukt, welches aus der Request-Zeile, den Headern und dem Message-Body besteht.

Listing 6-3: HTTP-Body bei einem Post-Request

Die erste Zeile ist die Request-Zeile, in der die Request-Methode, die aufzurufende URL und die HTTP Version, in dem Fall 1.1, angegeben werden. Als nächstes folgen die Header. Diese sind in dem Fall der Host, der Content-Type und die Content-Length. Danach schließt sich noch der Message Body an.

Post-Requests erzeugen:

In den meisten Programmiersprachen wird auch eine Unterstützung zum einfachen Erstellen von Post-Requests geboten. Auch hier soll ein Beispiel in C# folgen, welches der Form aus dem Beispiel Listing 6-1 entspricht.

```
public class PostRequest
{
  public static void main(String[] args)
  {
    String parameter =
“myText=myTextfieldValue&myCheckbox=on&mySubmitButton=Absend
en”);

    WebRequest webRequest =
WebRequest.Create("http://www.mysite.de/send.html");
  webRequest.Method = "POST";
  webRequest.ContentType = "application/x-www-form-
urlencoded";
  webRequest.ContentLength = parameter.Length;

  StreamWriter requestStreamWriter = new
StreamWriter(webRequest.GetRequestStream());
  requestStreamWriter.Write(parameter);
  requestStreamWriter.Close();

  StreamReader reader = new
StreamReader(webRequest.GetResponse().GetResponseStream());
  Console.Write(reader.ReadToEnd());
  }
}
```

Listing 6-4: Erzeugen eines POST-Requests mittels Java

Am Anfang werden auch hier die Parameter in einen String geschrieben. Dann wird wieder ein Objekt der Klasse „WebRequest“ angelegt. Dieses bekommt als Parameter aber nur die Seite zu welcher navigiert werden soll und nicht noch zusätzlich die Eingabewerte. Als nächstes wird mittels des WebRequest-Objektes der HTTP-Body erzeugt. Dabei wird erst die Request-Methode gesetzt und dann die Header. Als nächstes werden mittels des StreamWriters die Parameter in den Message-Body geschrieben. Damit ist der HTTP-Body komplett und der Request

bereit zum Absenden. Mittels „webRequest.GetResponse()" wird nun der Request abgesendet und es wird die HTML-Seite als Response zurückgegeben. Über die Konsole werden nun die einzelnen HTML-Quellcode-Zeilen ausgegeben.

6.3.2 Einfügen von Daten mittels JavaScript

Da mittels JavaScript eine Manipulation der HTML-Elemente möglich ist, können damit Daten in diese eingegeben werden, Aktionen ausgelöst werden usw. Das heißt, man kann mittels JavaScript beispielsweise Text in ein Eingabefeld eingeben oder einen Link klicken, ähnlich wie dies ein normaler Nutzer tun würde. Die folgende JavaScript-Funktion trägt in das Formular aus dem Beispiel Listing 6-1 die passenden Werte ein bzw. führt die benötigten Aktionen.

```
function enterDataAndSubmit()
{
  var myText = document.getElementsByName('myText')[0];

  var myCheckbox =
document.getElementsByName('myCheckbox')[0];
  var mySubmitButton =
document.getElementsByName('mySubmitButton')[0];

  myText.value = "myTextfieldValue";
  myCheckbox.click();
  mySubmitButton.click();
}
```

Listing 6-5: Einfügen von Daten in HTML mittels JavaScript

Als erstes werden mittels „document.getElementsByName" die jeweiligen Elemente aus dem HTML-Quellcode ausgelesen. Dann wird erst für das Eingabefeld der entsprechende Text gesetzt, es wird die Auswahlbox angeklickt und zuletzt der Absende-Button geklickt. Zum Ausführen dieser Funktion muss der Aufruf in das „onload"-Event der Seite, also „<body onload="enterDataAndSubmit();">". Bei diesem Vorgehen ist es völlig egal ob es sich beim Senden des Requests um die Post- oder Get-Methode handelt, in beiden Fällen ist der Ablauf identisch. Für diese Vorgehensweise gibt es zum Beispiel das JavaScript-Framework Selenium Core (Kapitel 6.4.4.1).

6.3.3 Nutzung interne Mechanismen des Internet-Browsers

Einige Browser besitzen auch die Möglichkeit, über interne Mechanismen eine Automatisierung durchzuführen. Dabei werden je nach Browser verschiedene Vorgehensweisen angeboten. Beispielsweise bietet der Internet Explorer verschiedene sogenannte Automation Controls, mit denen man eine Automatisierung erreichen kann. Weitere solche internen Mechanismen die eine

Automatisierung ermöglichen, werden zum Beispiel von Firefox, Google Chrome oder Opera bereitgestellt.66

6.3.4 Vergleich der Vorgehensweisen

Generieren von Requests		Einfügen von Daten mittels JavaScript		Nutzung interner Browser-Mechanismen	
Vorteile	**Nachteile**	**Vorteile**	**Nachteile**	**Vorteile**	**Nachteile**
Sind einfach zu erstellen	Cookies müssen selbst verwaltet werden	Kommt Nutzerverhalten sehr Nahe	Problematisch bei Nutzung über mehrere Browser	Gute Kompatibilität zum Browser	Mechanismus immer spezifisch zum Browser
Keine Notwendigkeit eigentliche Seite aufzurufen, um Requests zu senden	AJAX kann problematisch sein	Unabhängig von der Request-Methode	Eventuelle Inkompatibilitäten zu speziellen Nicht-Standard-HTML-Komponenten	Kommt Nutzerverhalten sehr Nahe	
Läuft unabhängig vom Browser	Hidden-Input Felder müssen beachtet werden		Ladezeiten müssen beachtet werden		
	Client-Seitige Überprüfungen, Änderungen bzw. Validierung werden nicht durchgeführt		JavaScript-Events werden nicht mit ausgeführt bzw. müssen separat ausgeführt werden		

Abbildung 6-2: Vergleich der Vorgehensweisen zum automatischen Testen von Webanwendungen

Da beim Generieren von Requests diese unabhängig vom Browser mittels systeminterner Schnittstellen ausgeführt werden, erfolgt keine Speicherung bzw. Verwaltung der Cookies durch den Browser. Wenn über die Response eines Webservers Cookies enthalten sind, so müssen diese ausgelesen, gespeichert und dann wieder mit den Requests mitgesendet werden. Loggt man sich beispielsweise in eine Webseite ein, so wird üblicherweise ein Cookie gesetzt, welcher genutzt wird um zu überprüfen, ob man eingeloggt ist oder nicht. Bei einer entsprechenden Komplexität der Cookies kann der Verwaltungsaufwand entsprechend schwierig werden. Auch AJAX kann bei dieser Methode Probleme bereiten. Prinzipiell werden mittels AJAX auch Requests ausgelöst, jedoch kann es entsprechend kompliziert sein, die Aufrufe aus dem HTML-Quellcode bzw. den übermittelten Daten herauszufiltern. Das bedeutet, man muss die auszurufende URL, Parameter und passende Werte erst herausfiltern, um den Request erzeugen zu können. Beim Generieren von Requests ist weiterhin zu beachten, dass in den Formularen auch Hidden-Input Felder vorhanden sein können und deshalb deren Werte mitgesendet werden müssen. Daher müssen die Namen und Werte dieser Felder immer ausgelesen und mitgesendet werden. Jegliche Änderungen, Validierungen oder

Überprüfungen die Client-Seitig durch zum Beispiel JavaScript an den Eingabedaten durchgeführt werden, werden bei dieser Methode nicht durchgeführt. Es wäre dann also notwendig die Funktionalität selbst nachzubauen oder die JavaScript-Funktionen selbst mit den entsprechenden Werten aufzurufen.

Größtes Problem beim Einfügen von Daten mittels JavaScript ist die Kompatibilität über mehrere Internet-Browser. Da unterschiedliche Implementierungen von JavaScript in den verschiedenen Browsern existieren, kann es passieren das gleiche JavaScript-Konstrukte auf verschiedenen Browsern, verschiedene Resultate hervorrufen. Ein weiteres Problem ist die mögliche Inkompatibilität bei Komponenten, die nicht standardmäßig in HTML vorkommen. Zum Beispiel können selbst erstellte JavaScript-Komponenten Probleme bereiten. Da bei dieser Methode mit den Komponenten interagiert wird, müssen dieser logischerweise auch auf der Webseite vorhanden sein. Das heißt etwaige Ladezeiten, egal ob vom Laden der Seite oder AJAX-Requests, müssen beachtet werden, damit die Aktionen erfolgreich ausgeführt werden können. Wenn man zum Beispiel in ein Textfeld über JavaScript den Text eingibt besteht weiterhin das Problem das alle damit verbundenen JavaScript-Events nicht mit ausgeführt werden. Die sind unter anderem das „OnFocus“-Event beim Fokussieren der Komponente oder das „OnBlur-Event beim herausklicken aus der Komponente. Wenn benötigt, müssen diese Events also manuell aufgerufen werden.

Da bei der Nutzung interner Mechanismen diese logischerweise vom Browser-Hersteller selbst eingefügt wurden, ist die Kompatibilität natürlich sehr hoch. Dies sollte eine gute Stabilität beim Durchführen der Tests hervorrufen. Nachteil ist, dass die Mechanismen immer nur spezifisch für die jeweiligen Browser funktionieren. Auch wenn neue Browser-Versionen veröffentlicht werden und Änderungen an diesen Mechanismen vorgenommen wurden, kann dies Probleme bereiten, indem vorhandene Tests eventuell nicht mehr ordnungsgemäß funktionieren.

Insgesamt lässt sich sagen, dass das generieren von Requests nur für einfache Webseiten zu nutzen ist. Bei komplexen Webapplikation, welche JavaScript und AJAX nutzen, ist diese Herangehensweise zu kompliziert. Das Einfügen von Daten mittels JavaScript ist, unter Nutzung entsprechender Frameworks, auch für komplexe Webapplikationen möglich. Die Nutzung interner Browsermechanismen eignet sich auch für das automatisierte Testen komplexer Webapplikationen.

6.4 Vorstellung einiger Testtools

In diesem Abschnitt sollen sowohl einige kostenpflichtige, als auch einige Freeware bzw. Open-Source Tools zum automatisierten Testen von Webanwendungen

vorgestellt werden. Dabei geht es insbesondere um das automatisierte Testen der Funktionalitäten der zu testenden Applikation.

6.4.1 Kostenpflichtige Testtools

HP Functional Testing: Dieses Testtool stammt von der Firma Hewlett Packard. Es unterstützt das funktionale Testen verschiedener Programmiersprachen bzw. Technologien, darunter auch Webapplikationen. Die Testskripte können dabei entweder durch Aufnehmen der Aktionen des Nutzers (Capture-Replay) oder durch das eigene Schreiben der Skripte erstellt werden. Genutzt wird dabei eine eigene Skriptsprache. Weiterhin werden verschiedene Web-Technologien bzw. Frameworks wie Silverlight, GWT, Dojo oder AJAX unterstützt. [Ank11]

Testing Anywhere: Mit diesem Tool ist ein Testen mit verschiedenen Capture-Replay-Varianten und einer Skriptsprache möglich. Auch dieses unterstützt über Webapplikationen hinaus verschiedene Programmiersprachen und Technologien. [Aut11]

WebUI Test Studio: Dies ist ein Testtool der Telerik Gruppe, welches speziell für Webtests erstellt wurde. Wieder wird ein Erstellen der Testfälle im Normallfall über das Aufnehmen der Aktionen durchgeführt. Da diese Firma selbst Komponenten für Webapplikationen anbietet, ist eine Unterstützung dieser direkt eingebaut. [Tel11]

6.4.2 Freeware bzw. Open-Source Testtools

Selenium: Selenium ist eine Toolsuite zum automatischen Testen von Webapplikationen. Diese Toolsuite besteht aus hauptsächlich drei Tools. Dabei ist es einerseits möglich Testskripte mittels Capture-Replay zu erstellen oder sich Tests mittels eines bereitgestellten Frameworks selbst zu erstellen. [Tho11_1] Das Framework ist dabei für viele verschiedene Programmiersprachen verfügbar.

Watin: Watin ist ein Testtool mit welchem Testskripte für Webapplikationen unter .NET erstellt und abgespielt werden können. [Wat11] Weiterhin wird auch hier eine Möglichkeiten zum Aufnehmen von Aktionen angeboten.

6.4.3 Vergleich der Tools

Wie zu sehen ist, haben die Tools im Bereich des automatischen Testens von Web-Applikationen alle ähnliche Features. In allen vorgestellten Tools sind sowohl Aufnahme-Möglichkeiten, als auch Möglichkeiten zum Programmieren von Test-Skripten vorhanden. Bei den kostenpflichtigen Programmen sind zwar zusätzlich noch die Unterstützung für andere Technologien gegeben und es ist womöglich ein besseres Testmanagement integriert, beim Testen der Web-Applikationen selbst werden aber sehr ähnliche Verfahren, wie die bei den kostenlosen Tools, angewandt. Daher ist es angebracht auf die kostenlosen Tools zurückzugreifen,

wenn nur die Kernfunktionalitäten für das automatische Testen von Webapplikationen benötigt werden bzw. nicht genügend Budget für eines der kostenpflichtigen Tools vorhanden ist. Bei den kostenlosen Programmen hat Selenium den Vorteil, dass es als Toolsuite neben den Kernfunktionalitäten auch zusätzliche Tools zum parallelen Ausführen von Tests und zum Remote-Testen bereitstellt.

Da Selenium als kostenloses Tool sehr umfangreiche Funktionalitäten bietet, soll es hier als Programm der Wahl genommen und näher betrachtet werden.

6.4.4 Selenium

Im folgenden Abschnitt soll Selenium mit all seinen Tools vorgestellt werden.

6.4.4.1 Selenium Core

Selenium Core ist eine JavaScript-Bibliothek, welche Webseiten in verschiedenen Browsern steuern kann. Selenium IDE und Selenium 1 nutzen diese Bibliotheken zur Automatisierung. Damit Selenium Core für die unterschiedlichen Browser kompatibel ist, sind verschiedene angepasste JavaScript-Bibliotheken für die jeweiligen Browser vorhanden.

6.4.4.2 Selenium IDE

Selenium IDE ist ein Firefox-Plugin, dessen Kernfunktionalität es ist, Testfälle aufzunehmen. Man kann also, wenn die Aufnahme läuft, Aktionen auf der gewünschten Webseite innerhalb von Firefox durchführen und die Selenium IDE wird sie dann aufnehmen. Dabei werden einerseits der durchgeführte Befehl, die Zielkomponenten und ein eventuell eingegebener Wert aufgezeichnet. Die Zielkomponente kann entweder über die ID, den Namen oder einen XPath-Ausdruck ermittelt. [Tho11] Neben dem Aufnehmen von Testfällen, ist auch das Schreiben von Testfällen, Abspielen von Testfällen, Debugging von Testfällen, zusammenstellen von Testfällen zu Testsuites und der Export in verschiedene Programmiersprachen möglich. Deshalb wird die Selenium IDE auch als Integrated Development Environment angepriesen. [Tho11] Die Skriptsprache, die verwendet wird um die Testfälle aufzunehmen, nennt sich Selenese welche im Prinzip aus HTML besteht. Der Export der Testfälle ist dabei in Java, C#, Perl, PHP und weitere Programmiersprachen möglich.

6.4.4.3 Selenium 1 - RC

Selenium 1 oder Selenium RC besteht einerseits aus einem Server und andererseits aus Programmierbibliotheken. Mit diesen ist es möglich Testfälle in einer Programmiersprache zu schreiben und auszuführen. Dieses Tool unterstützt dabei die gängigsten Internet-Browser unterstützt. Des Weiteren ist es möglich die Testfälle auf einem anderen Computer ferngesteuert auszuführen.

Selenium Server

Der Selenium Server ist dafür zuständig, ankommende Befehle im Internet-Browser auszuführen. Der Server kann dabei entweder auf einem entfernten Rechner sein oder lokal ausgeführt werden. Beim Start eines Testfalls öffnet der Selenium Server den angegebenen Internet-Browser und fügt die Selenium Core JavaScript-Bibliotheken ein, um mittels diesen die gewünschten Befehle auszuführen. Die Befehle werden an den Server mittels normaler Post- bzw. Get-Requests gesendet. [Tho11_2] Der Selenium Server fungiert dabei im Normallfall als Proxy-Server. Das heißt, auch die Anfragen an den Webserver bzw. Antworten von diesem werden über Selenium RC gesendet. Diese Funktion lässt sich jedoch ausschalten, sodass nur die Anfragen aus dem Testprogramm bzw. Antworten an das Testprogramm über Selenium RC laufen.

Selenium Client-Bibliotheken

Zum Programmieren solcher Tests gibt es eine Reihe von Selenium Bibliotheken für unterschiedliche Programmiersprachen, bspw. Java oder C#. Die API stellt dabei verschiedenste Befehle zur automatisierten Ausführung von Aktionen auf einer Webseite bereit, die bei der Ausführung in entsprechende Selenese Befehle umgewandelt werden und so vom Server interpretiert werden können. [Tho11_2] Nachdem eine Aktion ausgeführt wurde, sendet der Server ein Ergebnis zurück, welches dann als Rückgabewert der ausgeführten Funktion oder als Exception dem Entwickler zugänglich gemacht wird.

Die folgende Übersicht soll die übliche Funktionsweise von Selenium RC noch einmal verdeutlichen:

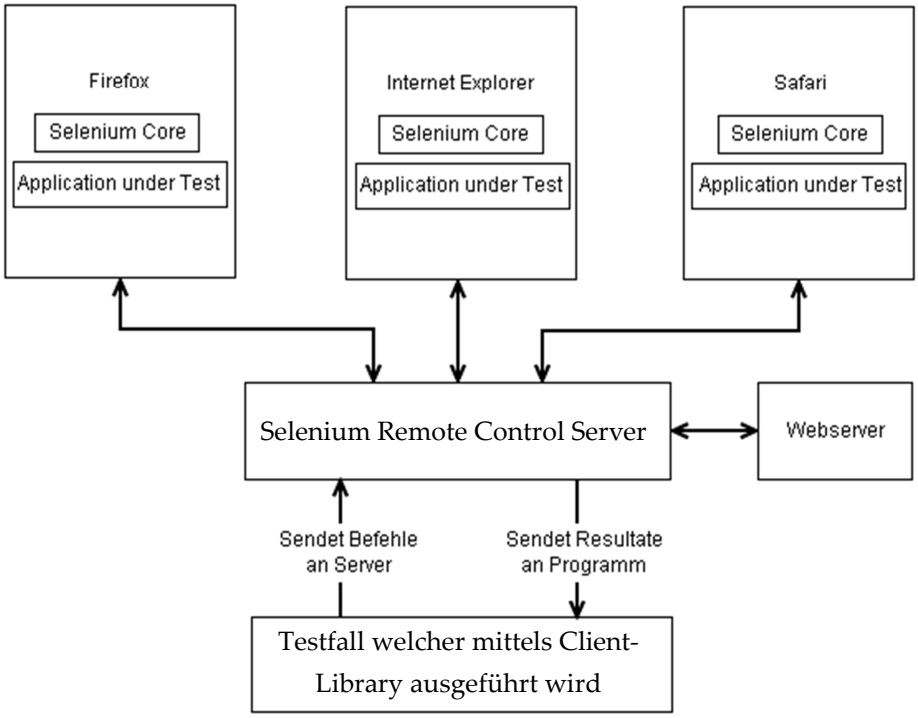


Abbildung 6-3: Vereinfachte Funktionsweise von Selenium RC, Vgl.: [Tho11_4]

Wie zu sehen ist, werden vom ausführenden Test-Programm aus die Befehle an den Selenium Server geschickt. Dieser gibt sie dann an den Internet-Browser bzw. an Selenium Core weiter, damit der Befehl in der Webapplikation ausgeführt werden kann. Nach der erfolgten Aktion, werden die Resultate vom Server zurück ans Programm gesendet. Der Selenium Server dient weiterhin als Proxy zwischen Webserver und Webseite. Wie zu sehen ist, können verschiedene Internet-Browser für die Tests genutzt werden. Es kann jedoch nur ein Testfall zur gleichen Zeit ausgeführt werden. Um mehrere Testfälle parallel laufen zu lassen, bedarf es dem Tool „Selenium Grid".

6.4.4.4 Selenium Grid

Dieses Tool ist zum parallelen ausführen mehrere Tests gedacht. Es verteilt automatisch Testfälle auf Rechner. Es müssen also mehrere Rechner vorhanden sein, auf denen die Tests laufen sollen. Für die Testfälle bzw. das Testprogramm selbst sind keine größeren Eingriffe notwendig, da sich Selenium Grid als einzelner Selenium Server präsentiert. [Tho11_3]

6.4.4.5 Selenium 2 – Webdriver

Selenium 2 bzw. Selenium Webdriver ist ein noch relativ junges Projekt, welches Selenium RC ablösen soll. Dabei ist Webdriver ein Framework, welches zum Ausführen der Befehle gedacht ist und ersetzt damit Selenium Core. Für lokale Tests ist auch der Selenium Server nicht mehr zwingend notwendig. Im Gegensatz zu Selenium Core nutzt Webdriver nicht JavaScript um die Befehle auszuführen.

Webdriver nutzt interne Mechanismen des Internet-Browsers, um eine Automatisierung zu erzielen. So wird im Firefox Webdriver als Extension eingebettet oder im Internet Explorer werden IE-Automation Controls verwendet. [Sim11]

6.5 Konzepte zum Aufbau eines Testtools

Im Folgenden sollen Konzepte zum Aufbau eines Testtools unter der Nutzung von Selenium beschrieben werden. Diese Konzepte sind zwar allgemeingültig, jedoch wird für die Umsetzung Wissen über die zu testende Webapplikation benötigt. Vor den Konzepten werden noch die Ziele, die mit einer solchen Eigenimplementierung erreicht werden sollen, näher erläutert.

6.5.1 Gründe und Ziele für eine Eigenimplementierung

Das Ziel der Eigenimplementierung besteht darin eine möglichste wartbare, flexible und einfach zu nutzende Lösung zum automatischen Testen von Webapplikationen zu schaffen.

Die konkreten Ziele der Eigenimplementierung sind:

- Einfache Anpassungsmöglichkeiten bei Änderungen in der Webapplikation
- Einfacher Aufbau der Testfälle
- Gute Wartbarkeit der Testfälle
- Ausschließen sensibler Daten aus den Testfällen
- Verwaltung verschiedener Umgebungen
- Möglichkeit zum Erstellen von Testsuites
- Geeignetes reagieren auf unerwartete Ereignisse
- Automatisch Wartezeiten bei AJAX-Requests einhalten

Selenium allein erfüllt dabei die genannten Anforderungen nicht in vollem Umfang.

Die Selenium IDE ist zwar an sich leicht zu verwendet, jedoch birgt die genutzte Capture-Replay Methode einige Nachteile. Das größte Problem ist wohl, dass die Werte die aufgenommen wurden fest eingetragen und nicht variabel gehalten sind. Das heißt, dass ein Testfall auch immer nur mit denselben Testfallwerten durchgeführt wird. Oft ist es aber erwünscht, denselben Testfall für verschiedenen Werten zu nutzen. Dies ist ein Problem, was viele solcher Aufnahme-Tools mit sich bringen. Ein weiteres Problem ist, dass nicht alle Aktionen so von Selenium erfasst werden, dass man die Testfälle ohne Nachbearbeitung nutzen kann. Zum Beispiel werden Wartezeiten bei AJAX-Requests nicht eingehalten, sondern es werden direkt die weiteren Aktionen ausgeführt. Die Nachbearbeitung der Testfälle bedarf im Normallfall wiederum einer Person, die sich gut mit dem Tool bzw. den Befehlen

auskennt. Somit ist also die Einfachheit nicht mehr gegeben. Damit erfüllt die Selenium IDE wie auch die meisten Capture-Replay Tools nicht die Anforderungen die gestellt werden.

Selenium RC bzw. Webdriver ist sicher eine flexiblere Lösung, da man die Testfälle direkt programmiert. Das große Problem ist hier natürlich, dass man dafür entsprechende Programmierkenntnisse braucht und damit die Einfachheit nicht gegeben ist.

Die Eigenimplementierung soll also die genannten Probleme eliminieren. Zum Zugriff auf die Webapplikation selbst soll aber weiterhin Selenium RC bzw. Webdriver genutzt werden.

6.5.2 Modell und Testfälle

6.5.2.1 Modell der Web-Applikation

Durch die Erstellung eines Modells der Webapplikation kann die Wartbarkeit des Testtools stark erhöht werden. In dem Modell werden alle vom Testtool potenziell verwendeten Elemente und zusätzliche zum Element gehörige Informationen gespeichert. Wenn nun ein Testfall erstellt oder ausgeführt wird, können sich dann alle Informationen aus dem Modell gegriffen werden. Der große Vorteil dieser Methode ist, dass wenn sich Elemente in der Webseite ändern, neue Elemente hinzukommen oder Elemente entfernt werden diese Änderungen nur im Modell aktualisiert werden müssen. Ein solches Modell kann zum Beispiel in Excel erstellt werden. Je nach Webapplikation baut man das Modell mit verschiedenen Informationen auf. Im Folgenden soll ein Ausschnitt eines solchen Modells gezeigt werden.

Bereich	Masken-name	Masken-nummer	Mapping	Typ	Navigation	ID	Ladebalken
KFZ	Basis-angaben	0	Beruf	Text		idKfzBeruf	idKfzBeruf_CB
KFZ	Basis-angaben	0	Weiter	Link	Standard (Antrag)	idKfzWeiter	

Abbildung 6-4: Auszug aus einem Modell einer Webapplikation

Dieses beispielhafte Modell beschreibt eine Webapplikation aus dem Versicherungsbereich. Dabei ist die Applikation in verschiedene Bereiche unterteilt, die sich mittels eines Menüs erreichen lassen. Die Bereiche sind dann in Masken unterteilt.

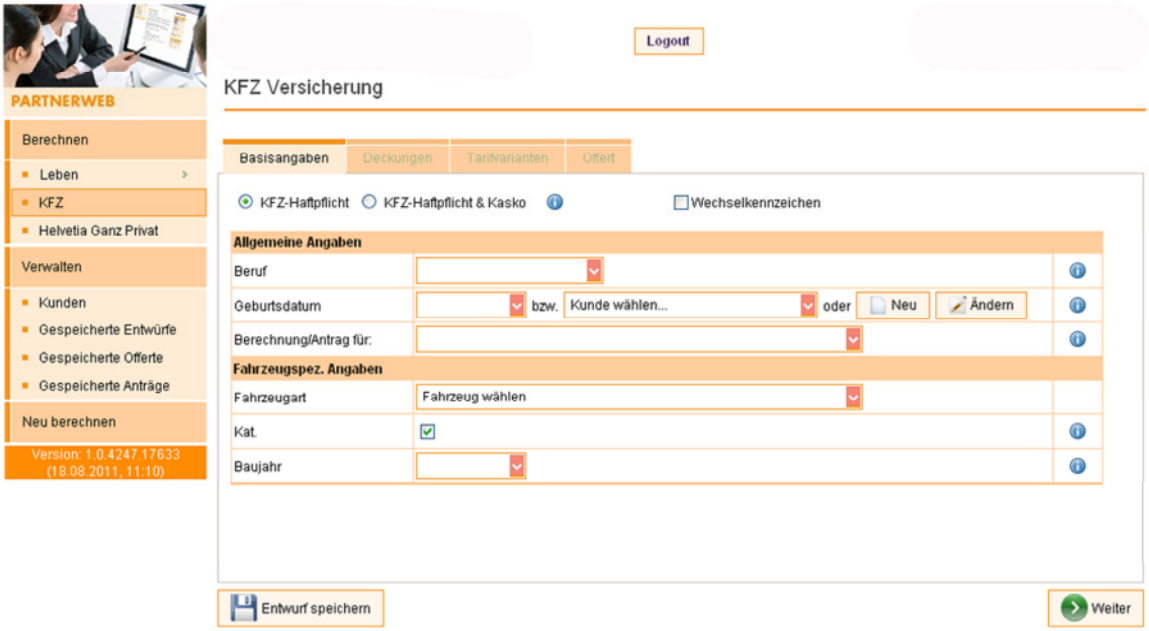


Abbildung 6-5: Screenshot aus der Webapplikation „HPSaktivWeb“

In dem abgebildeten Modell ist in der ersten Spalte der Bereich angegeben, in dem sich die jeweilige Komponente in der Webapplikation befindet. Danach folgt der Maskennamen, der jeweiligen Maske. Jede Maske besitzt in dem Beispiel einen Index, welcher in der dritten Spalte angegeben wird. Als nächstes folgt ein Name zum eindeutigen Identifizieren der Komponente. Damit dieser eindeutig ist, darf jeder Name in der jeweiligen Maske immer nur einmal vergeben werden. Dadurch kann über den Bereich, Maske und Name eindeutig die Komponente identifiziert werden. Danach wird der Typ der Komponente angegeben, um später die passenden Aktionen auszuführen zu können. Die Typen können zum Beispiel Textfelder, Links, Radiobutton, etc. sein. Die Spalte „Navigation“ ist in dem Fall für Links reserviert. Es wird angegeben, welche Navigationsart für den Link gilt und gegebenenfalls wohin navigiert wird. Dabei kann beispielsweise eine Standardnavigation erfolgen, welcher einen normalen Seitenaufruf hervorruft. Weiterhin könnte durch Klick auf einen Link auch ein AJAX-Request ausgeführt werden, welcher dann einen entsprechend anderen Navigationstyp bekommt. Nachfolgend wird die ID der Komponente angegeben, wie sie in der Webapplikation vorhanden ist. Damit kann Selenium später auf die Komponente zugreifen. Neben der ID, können auch XPath-Ausdrücke oder Namen verwendet werden, um die Komponente zu identifizieren. Wird dies getan muss allerdings eine Unterscheidung zwischen diesen Typen im Modell eingebaut werden. In der letzten Spalte ist noch die ID der Ladebalken-Komponente anzugeben, falls durch Interaktion mit dem Element eine AJAX-Request ausgelöst wird. Dieser wird benötigt, um später die Wartezeiten einhalten zu können. Die ID der jeweiligen Komponenten befinden sich im HTML-Quellcode der Webapplikation. Sie lässt sich zum Beispiel mit dem Tool „Firebug“, ein Plugin für Firefox, sehr leicht auslesen.

Im Beispiel sind nun zwei Komponenten angegeben, die beide im Bereich „KFZ“ befindlich sind. Auch sind beide in der Maske „Basisangaben“, womit auch die Maskennummer jeweils „0“ ist. Zum Identifizieren (Mapping) wurde einmal als Name „Beruf“ und „Weiter“ vergeben. Der Typ der Komponente „Beruf“ ist Text, also ein Eingabefeld und der der Komponente „Weiter“ ist Link. Die Navigation ist nur für den Link relevant und ist eine Standardnavigation zum Bereich „Antrag“. Nachfolgend sind die IDs aus dem HTML-Quellcode der jeweiligen Komponenten angegeben. Das Eingabefeld löst, nachdem Text eingetragen wurde, einen AJAX-Request aus. Die ID des dabei angezeigten Ladebalkens wird daher mit festgehalten.

6.5.2.2 Aufbau der Testfälle

Testfälle sollen möglichst einfach, auch von Nicht-Informatikern, zu erstellen sein. Das heißt es sollen keine Fachbegriffe oder ähnliches vorkommen. Weiterhin sollen nur die nötigsten Informationen abgefragt werden, um die Testfälle möglichst übersichtlich zu halten. Da der Nutzer oft nur Eingaben tätigen und dann nur die Ausgaben überprüft will, sollte es auch möglich sein Testfälle zu erstellen die keine Navigationsinhalte haben. Das heißt, dass im Zweifel auch eine automatische Navigation über das Tool erfolgen muss. Um ein einfaches Erstellen von Testfällen zu gewährleisten, bietet sich Excel an. Ein möglicher Testfall, passend zum oberen Modell, könnte wie folgt aussehen.

Bereich	Maske	Schlüssel	Wert
KFZ	Basisangaben	Beruf	Programmierer
		Assert(Beruf)	Programmierer
		Weiter	

Abbildung 6-6: Testfall zum Testen einer Webapplikation

In den ersten beiden Spalten werden der Bereich und die Maske eingegeben in denen sich die Komponenten mit denen Interagiert werden soll befinden. Dadurch kann beim Starten des Tests automatisch in den gewünschten Bereich navigiert werden, ohne dass der Nutzer dies selbst tun muss. Da die Navigation zum Bereich immer dieselbe ist und von keinerlei Parametern abhängig, außer der Bereich in den navigiert wird, gibt es nichts Wesentliches zu testen. Nachdem nun klar ist in welchem Bereich und welcher Maske die Komponenten zu finden sind, muss die Interaktion mit diesen durchgeführt werden. Dabei wird als Schlüssel der vergebene Name im Modell verwendet. So ist es einerseits möglich, später auf die Komponente im Modell zurückzugreifen, um somit alle notwendigen Informationen zu erhalten und auf der anderen Seite sind die Namen für den Nutzer verständlich und nachvollziehbar, insofern sinnvolle Namen im Modell für die Komponenten vergeben wurden. Da die Komponente „Beruf“ ein Eingabefeld ist, muss ein Wert in diesen eingegeben werden. In diesem Fall wurde als Beruf „Programmierer“ gewählt. In der nächsten Zeile erfolgt eine Überprüfung des

eingegebenen Wertes. Wie zu sehen ist muss dafür ein Befehl eingeführt werden, um zu unterscheiden ob es sich um eine Eingabe oder Überprüfung handelt. Als Befehl für eine Überprüfung wurde das Schlüsselwort „Assert“ gewählt. In Klammern steht dabei die zu überprüfende Komponente und im Feld „Wert“ der zu prüfende Wert. Es wird also geprüft, ob in dem Eingabefeld „Beruf“ tatsächlich „Programmierer“ ordnungsgemäß eingetragen wurde. Als letzte Aktion wird noch auf den Weiter-Link geklickt. Dafür ist kein Eingabewert notwendig. Will man eine Maske wechseln, wäre dies auch möglich, indem einfach nur im Feld „Maske“ die neue Maske eingegeben würde. Dies könnte dann folgendermaßen aussehen.

Bereich	Maske	Schlüssel	Wert
KFZ	Basisangaben	Beruf	Programmierer
		Assert(Beruf)	Programmierer
	Zusatzangaben	Alter	12

Abbildung 6-7: Testfall zum Testen einer Webapplikation über mehrere Masken[2]

Hier wird von der Maske „Basisangaben“ zu Maske „Zusatzangaben“ gewechselt, ohne dass der Nutzer explizit angeben muss, dass auf den Weiter-Link geklickt werden soll. Dies wird erreicht da bekannt ist, dass die vorgestellte Webapplikation immer einen Weiter-Button zum Wechseln der Maske angibt. Somit wird es möglich, dass das Tool automatisch auf den Button klickt. Je nach Webapplikation kann dieser Vorgang differieren. Natürlich sollten die Testfälle auch funktionieren, wenn die Angabe zum klicken des Links vorhanden wären.

Wie zu sehen ist, können mit dieser Methode auf recht einfache Art und Weise übersichtliche Testfälle erstellt werde. Der schwierigste Teil dabei dürfte sein, die richtigen Schlüssel aus dem Modell herauszufiltern.

6.5.2.3 Wartbarkeit der Testfälle

Um das Testtool gegen Änderungen in der Webapplikation flexibel und wartbar zu machen, wurde das oben beschriebene Modell genutzt. Solche Änderungen können aber auch die Testfälle betreffen. Wird zum Beispiel ein neues Pflichtfeld eingebaut, so muss auch dieses bei den betroffenen Testfällen eingefügt werden. Gerade an den Stellen, an welchen über viele Testfälle hinweg immer dieselbe Aktion durchgeführt wird, ist eine Auslagerung dieser sinnvoll. Um dies zu bewerkstelligen, muss ein Einfüge-Befehl eingeführt werden. Dieser referenziert dann auf einen anderen Testfall, welcher dann an der Stelle des Einfüge-Befehls eingesetzt wird. So können Testfälle erstellt werden, die dann in beliebig vielen anderen eingefügt werden können. Ändert sich dann eine Aktion so muss diese nur in einem Testfall geändert werden. Das nachfolgende Beispiel zeigt einen Testfall mit Einfüge-Befehl und den dazugehörigen einzufügenden Testfall.

[2] Teile dieses Testfalles sind nicht in dem Ausschnitt des Modell aufgeführt

Bereich	Maske	Schlüssel	Wert
		Include(KFZ-Beruf)	
	Zusatzangaben	Alter	12

Abbildung 6-8: Testfall mit Einfüge-Befehl

Wie zu sehen ist, befindet sich im Testfall aus Abbildung 6-7 nun statt der Eingabe des Berufes ein Include-Befehl. Dabei wird der Name des zu inkludierenden Testfalles angegeben. Der zu inkludierende Testfall wird dann genau an der Stelle eingefügt.

Bereich	Maske	Schlüssel	Wert
KFZ	Basisangaben	Beruf	Programmierer
		Assert(Beruf)	Programmierer

Abbildung 6-9: Testfall zum einfügen

So könnte daraus der zu inkludierende Testfall aussehen. Es wurde also die Eingabe des Berufes und die Überprüfung der Eingabe ausgelagert. Mittels des Einfüge-Befehls kann dieser Testfall nun an jeder beliebigen Stelle eingefügt werden. Dabei ist bei diesem zu beachten, dass ein Bereich und eine Maske angegeben wurde, was zu einer Navigation zu dieser führt. Will man dies vermeiden so muss der Bereich und Maske weggelassen werden. Dann erfolgt nur eine Einfügung der Schlüssel und Werte. Dadurch ist es möglich nur bestimmte Aktionen, Aktionen für eine ganze Maske oder gar für einen ganzen Bereich auszulagern. Auch ist es so möglich Testfälle nur mittels des Einfügens anderer Testfälle aufzubauen.

Wird ein Testfall mit Einfüge-Befehl ausgeführt, so werden zuvor alle zu inkludierenden Testfälle angefordert und diese in den Haupttestfall zusammengeführt.

6.5.2.4 Login-Daten aus Testfällen ausschließen

Oft ist es der Fall, dass in Webapplikationen Login-Systeme vorhanden sind. Daher ist es notwendig sich bei einem Test in die Applikation einzuloggen. Allerdings sollte vermieden werden, dass die Login-Daten in den Testfällen stehen, denn dann müsste man natürlich Benutzername und Passwort in Klartext abspeichern. Gibt man nun einen Testfall an einen Mitarbeiter weiter und vergisst die Login-Daten zu entfernen, kann dies natürlich sehr große Probleme verursachen. Daher ist es sinnvoll, den Login-Prozess in das Testtool zu integrieren und die Login-Daten im Tool abzufragen. Da sich am Login im Normallfall selten etwas ändert und es an der Stelle auch nicht viel zu testen gibt, stellt ein solches Vorgehen auch kein großes Problem dar. Allerdings gilt es zu beachten, dass es verschiedene Login-Arten geben kann. Zum Beispiel könnte es für unterschiedliche Benutzergruppen oder Umgebungen unterschiedliche Login-Arten geben. Um dies im Code zu realisieren, bietet es sich an für jede Login-Art eine Klasse, ein Interface welches die Login-

Klassen implementieren und eine Factory zum Weiterleiten an die jeweilige Login-Art zu erstellen. Dies könnte beispielsweise wie folgt aussehen.

ILogin – Interface für Login-Klassen

```
interface ILogin
{
  void ProcessLogin();
}
```

Logintyp 1,2,…

```
class LoginTyp1 : ILogin
{
  public void ProcessLogin()
  {
    //Vorgang zum einloggen
  }
}
```

Factory zum Login

```
class LoginFactory
{
  public static ILogin GetLogin(LoginType type)
  {
    if(type == LoginType.LoginType1)
    {
      return new LoginTyp1();
    }
  }
}
```

Listing 6-6: Aufbau eines Login-Systems für ein Testtool

Als erstes wird das Interface für den Login deklariert. Es umfasst dabei lediglich die Methode „ProcessLogin“, in welcher der Login-Vorgang abgearbeitet werden soll. Als nächstes folgen die einzelnen Login-Arten. Jede der Klassen sollte das Interface „ILogin“ und damit die ProcessLogin-Methode implementieren. In der ProcessLogin-Methode sollte nun der eigentliche Login abgearbeitet werden. Dabei sollten die einzelnen Komponenten aus dem Modell gelesen werden. Natürlich können noch weitere Methoden zum Setzten von Variablen etc. in der Klasse folgen. Als letztes folgt die Klasse „LoginFactory“. Sie besitzt die statische Methode „GetLogin“ welche ein Objekt vom Typ „ILogin“ zurückgibt. Je nach Login-Art, was in dem Fall ein Enum ist, wird die entsprechende Login-Art gewählt und ein Objekt der jeweiligen Klasse zurückgegeben. An der Stelle, an welcher der Login durchgeführt werden soll, muss nun nur die GetLogin-Methode der LoginFactory genutzt werden, um an das entsprechende Objekt zu kommen und dann die ProcessLogin-Methode aufgerufen werden.

6.5.3 Verwaltung

6.5.3.1 Verwaltung für Umgebungen

Oft ist es der Fall, dass mehrere Server vorhanden sind, auf denen eine Webapplikation läuft. So gibt es im Normallfall mindestens einen Entwicklungs- und Produktionsserver. Die unterschiedlichen Server werden mit verschiedenen URLs angesprochen, es werden eventuell unterschiedliche Login-Daten benötigt etc. Daher bietet es sich an, dass der Nutzer Umgebungen im Testtool anlegen und verwalten kann. So ist es möglich, dass dann entweder für jeden Testfall oder für den jeweiligen durchzuführenden Test, welcher aus mehreren Testfällen bestehen könnte, eine Umgebung festgelegt wird. Weiterhin könnte auch der zu verwendende Browser und benötigte Browserkonfigurationen angegeben werden. Eine Umgebung könnte zum Beispiel die folgenden Informationen beinhalten.

Name	URL	Browser	Server	Benutzername	Passwort
Entwicklung	www.Entwicklung.de	Firefox	Entwicklung	Nutzer1	Pw1

Abbildung 6-10: Umgebung für das Testen einer Webapplikation

Die Umgebung besteht also aus einem Namen als Referenz, der URL, dem gewünschten Browser, der Server auf dem die Webapplikation läuft, der Benutzername und das Passwort. Beim Einbau einer solchen Verwaltung in das Testtool, sollte das Passwort natürlich nicht in Klartext angezeigt und verschlüsselt abgespeichert werden.

In das Testtool wurde ein Mechanismus eingebaut, bei dem man die Umgebung auswählt und jeder Test gegen die Umgebung läuft bis eine andere Umgebung zum Testen ausgewählt wurde.

6.5.3.2 Erstellen von Testsuiten

Testfälle sollten zu Testsuiten zusammengefasst werden können, damit beim Starten eines Tests alle Testfälle nacheinander ablaufen können. Dadurch ist der Nutzer also nicht gezwungen, jeden Testfall einzeln nacheinander zu starten, sondern kann viele mit einmal starten. Dabei bietet es sich, dass erstellte Testsuiten abgespeichert werden können. Damit muss nur einmal die Testsuite definiert werden und kann dann immer vom Nutzer ausgewählt werden.

6.5.4 Konzepte zur Implementierung

6.5.4.1 Projektaufteilung im Testtool

Am besten ist es das Tool in mindestens zwei Projekte aufgeteilt werden. Einerseits ein Projekt in welchem die eigentlichen Tests ablaufen und ein zweites, in dem sich die Oberfläche und verschiedene Verwaltungsaufgaben befinden. Aus dem Projekt, welches die Oberfläche und Verwaltung übernimmt, werden dann die jeweiligen Methoden zum Starten des Testfalles aufgerufen. Dieser Aufruf findet in einem separaten Thread statt. So ist dafür gesorgt, dass während des Tests weiterhin mit

der Oberfläche interagiert werden kann. Die Trennung in zwei Projekte sorgt dafür, dass die Projekte unabhängig voneinander sind und somit beispielsweise Projekt ohne größere Schwierigkeiten durch ein anderes, kompatibles Projekt ersetzt werden kann.

6.5.4.2 Probleme mit Selenium RC bzw. Webdriver

Bei Selenium RC bzw. Selenium Webdriver gibt es bei der Nutzung verschiedene Probleme zu beachten.

Selenium RC ist die Vorgängerversion von Selenium Webdriver. Sie wird zwar trotzdem noch von den Herstellern unterstützt, jedoch funktioniert das Tool nicht für neuere Browser-Versionen. So wird zum Beispiel Firefox 4 noch unterstützt, jedoch nicht Firefox 5. Deshalb gilt es darauf zu achten, bei Selenium RC nur passende ältere Browser-Versionen zu nutzen.

In Selenium Webdriver funktioniert das Klicken eines Links im Internet Explorer nicht. Das heißt, es erfolgt keine Aktion im Browser, wenn der Befehl ausgeführt wird. Um dies zu umgehen muss der Link fokussiert und dann die Enter-Taste gedrückt werden. So können auch Links im Internet Explorer geklickt werden. Weiterhin gilt es zu beachten, dass im Gegensatz zu Selenium RC in Selenium Webdriver nur mit Komponenten interagiert werden kann die sichtbar auf der Seite vorhanden sind. So muss dafür gesorgt werden, dass bspw. bei Untermenüs die unsichtbar sind, erst auf das jeweilige Hauptmenüelement geklickt werden muss, um das Untermenü sichtbar zu machen, um dann damit zu interagieren. Ein weiteres Problem ist, dass mit Webdriver bei der Suche nach Komponenten immer explizit angegeben werden muss, ob es sich um die ID, Name oder XPath-Ausdruck der Komponente handelt. Das heißt man muss in dem Fall im Modell die Typen unbedingt mit angeben.

6.5.4.3 Wrappen der Komponenten in der Webapplikation

Um in dem Tool eine einfache Interaktion mit den Komponenten (HTML-Elemente) zu erreichen lohnt sich ein wrappen dieser. Dabei wird im Testtool für jede Komponente eine Klasse erstellt, welche die jeweiligen Aktionen bereitstellt. Damit kann die Klasse genutzt werden, um auf die jeweilige Komponente zuzugreifen. So könnte man zum Beispiel eine Klasse „ButtonComponent“ erstellen, die zum Interagieren mit Links bzw. Buttons zuständig ist. Die Klasse könnte beispielsweise so aussehen:

```
class ButtonComponent
{
  private ISelenium selenium;
  private String id;

  public ButtonComponent(ISelenium selenium, String id)
```

```
    {
      this.selenium = selenium;
      this.id = id;
    }

    public void Click()
    {
      selenium.Click(id);
    }

    public String GetText()
    {
      return selenium.GetText();
    }

    public bool IsVisible()
    {
      return selenium.IsVisible();
    }
  }
```

Listing 6-7: Wrappen einer Komponente für die Nutzung im Testtool

Wie zu sehen ist, wird der Klasse zuerst das Selenium-Objekt und die Id übergeben.[3] Mit diesen können dann die jeweiligen Aktionen ausgeführt werden. Dabei werden Oberklassen definieret von welchen dann alle Komponentenklassen ableiten, um eine bessere Strukturierung im Testtool zu erreichen. Weiterhin wird man an vielen Stellen sicherlich mehr tun müssen, also nur, wie in dem Beispiel, den passenden Selenium-Befehl auszuführen. Beim Ausführen der Click-Methode müssen noch Wartemechanismen eingebaut werden, welche warten bis die Aktion die durch den Buttonclick ausgelöst wurde vollendet ist.

Mit dieser Technik wird das Arbeiten mit den Komponenten im Tool wesentlich vereinfacht.

6.5.4.4 AJAX-Ladezeiten erkennen

Grundsätzlich bietet Selenium eine Vorgehensweise zum Abwarten von AJAX-Requests an. Dabei wird auf die Sichtbarkeit bzw. das Vorhandenseins einer Komponente gewartet, die durch den AJAX-Request erstellt bzw. angezeigt wird. Das Konstrukt dafür sieht wie folgt aus.

```
public void AjaxWait(int timeout, String id)
{
  for(int seconds = 0; !Selenium.IsElementPresent(id);
seconds++)
  {
```

[3] In dem Beispiel wird Selenium RC verwendet

```
      if(seconds >= timeout) return false;
      Thread.Sleep(1000);
    }
    return true;
  }
```

Listing 6-8: Warten auf die Vollendung eines AJAX-Requests

In dem Beispiel wird in einer For-Schleife überprüft, ob die jeweilige Komponente vorhanden ist. Solange dies nicht der Fall ist, wird eine Sekunde abgewartet. Wenn nun so viel Zeit verstrichen ist, dass der Timeout überschritten wurde, wird mittels eines Return-Statements das Warten auf die Komponente abgebrochen.

Das Problem an dieser Variante ist, dass man nicht immer weiß, welche Komponenten durch den AJAX-Request erstellt werden bzw. ob überhaupt neue Komponenten durch den Request erscheinen. Die Wartezeit bis der Request vollendet ist muss aber trotzdem abgewartet werden. Wird ein AJAX-Request durchgeführt, so wird in der Webapplikation ein Indikator dafür angezeigt. Das heißt es wird ein Ladebalken oder ähnliches angezeigt. Diese Ladebalken sind erst unsichtbar im HTML und werden dann, wenn nötig, mittels JavaScript bzw. CSS sichtbar gemacht. Dadurch ist es möglich, mit einem ähnlichen Konstrukt wie aus Beispiel Listing 6-8 die Sichtbarkeit des Ladebalkens zu prüfen und damit sicherzustellen, ob ein AJAX-Request fertig ist oder nicht. Dabei ist nur die ID des Ladebalkens nötig. Nutzt man das Modell wie es in Abbildung 6-4 beschrieben wurde, kann man die zur Komponente gehörige Ladebalken-ID daraus auslesen. Gibt es nur einen Ladebalken für alle Komponenten, so reicht es natürlich die ID einmal separat zu speichern und nicht für jede Komponente einzeln.

6.5.4.5 Erkennen und reagieren auf unerwartete Ereignisse

Beim Ausführen von Aktionen können unerwartete Ereignisse auftreten. Beim Klicken eines Links können zum Beispiel statt des erwarteten Seitenwechsels Alert-Nachrichten oder Popups auftauchen. Dies muss natürlich erkannt und entsprechend reagiert werden. Mit Selenium ist es möglich, aufgepoppte Alerts zu erkennen und zu bestätigen. In den meisten Fällen handelt es sich dabei nur um Informationen, die Bestätigt nur werden müssen. Der Alert sollte aber nur dann bestätigt werden, wenn dies im Testfall angegeben wird, denn es könnte sich dabei auch um einen bedeutenden Fehler handeln. Dann sollte ein Abbruch des Testfalls stattfinden.

Taucht unerwartet ein sehr komplexer Popup auf, der sehr viele Eingaben erfordert, so ist dies nicht einfach mit dem Tool automatisch zu regeln, da das Tool nicht wissen kann was es einzugeben gilt. Daher ist an der Stelle ein Abbruch des Testfalls sinnvoll.

Taucht ein einfacher Popup der ähnlich wie ein JavaScript-Alert und zur Bestätigung einer Warnung etc. da ist, kann hier automatisch auf den Ok-Button geklickt werden. Wird der Popup aufgrund eines Fehlers angezeigt und nicht wegen einer Warnung, sollte eher ein Abbruch des Testfalls stattfinden. Um das automatische Bestätigen eines Popups zu realisieren, sollte dieser bzw. dessen Komponenten im Modell festgehalten werden. Dann kann immer wenn auf einen Link geklickt wird überprüft werden, ob der Popup vorhanden und sichtbar auf der Seite ist. Dazu muss also nur die Sichtbarkeit des Popups oder einer seiner Komponenten mittels Selenium überprüft werden.

6.5.4.6 Überprüfen von Werten in Tabellen

Da größere Informationsmengen bei Webapplikationen meist in Tabellen gehalten werden, ist es sinnvoll, eine flexible Möglichkeit zum Überprüfen der Werte innerhalb von Tabellen zu bieten. In Selenium RC und Selenium Webdriver gibt es jeweils Möglichkeiten, um einen Wert in der gesamten Tabelle, einer bestimmten Spalte, einer bestimmten Zeile oder einer Zelle zu suchen.

In Selenium RC ist das Arbeiten mit Tabellen mit dem Befehle „selenium.GetTable(id, zeile, spalte);“ möglich. Es wird dabei der Wert aus der jeweiligen Zelle extrahiert. Mittels Schleifen kann man nun über die ganze Tabelle, Spalte etc. nach einem Wert suchen.

Mittels Selenium Webdriver kann die Tabelle gesucht werden, dann kann in der Tabelle nach „<tr>“-Tags (Zeile) und in diesem wiederum nach „<td>“-Tags (Spalte) gesucht werden, um dann die jeweiligen Zellenwerte auszulesen. Auch hier kann wiederum über Schleifen durch die gesamte Tabelle, Spalten oder Zeilen gesucht werden.

Im Testfall kann dann ein Konstrukt ähnlich dem folgendem verwendet werden, um in bestimmten Zellen, Spalten usw. zu suchen.

In gesamter Tabelle suchen:	Assert(myTable)
In Spalte 5 suchen:	Assert(myTable;S5)
In Zeile 3 suchen:	Assert(myTable;Z3)
In Zelle bei Spalte 5 und Zeile 3 suchen:	Assert(myTable;S5;Z3)

Abbildung 6-11: Möglichkeiten um Werte in Tabellen zu Überprüfen

6.5.4.7 Events

Beim Abarbeiten der Testfälle ist es sinnvoll an angebrachten Stellen Events zu werfen. Beim Beenden eines Testfalles, beim Beenden der Testsuite oder im Fehlerfall wird ein entsprechendes Event geworfen. Dadurch wird eine bessere Unabhängigkeit zwischen den verschiedenen Projekten, wie im Punkt 6.5.4.1 beschrieben, gewahrt. Über das Event können verschiedene Folgeaktionen ausgelöst werden. Zum Beispiel wird beim Beenden eines Testfalls, über das dazu geworfene

Event, in der Oberfläche angezeigt, dass der Testfall vollendet ist und der nächste begonnen wird. Zusätzlich zum Event selbst, können noch eigene Event-Argumente mitgegeben werden. Diese können zum Beispiel den Namen oder eine Nachricht etc. enthalten. Im Folgenden ein kleines Beispiel dazu.

Event:

```
public event EventHandler<TestCaseEventArgs>
TestCaseCompleted;
```

TestCaseEventArgs:

```
public class TestCaseEventArgs : EventArgs
{
  public String TestCaseName {get; set; }
  public String Message {get; set;}
}
```

Listing 6-9: Erstellen eines Events mit zugehörigen Argumenten

Im ersten Schritt wird das Event definiert, indem Fall für einen durchgelaufenen Testfall. Weiterhin werden Event-Argumente für den Testfallnamen und eine Nachricht definiert. Das Werfen des Events kann dann folgendermaßen aussehen.

```
TestCaseCompleted(this, new TestCaseEventArgs {TestCaseName
= „Name“, Message = “Message” });
```

Listing 6-10: Werfen eines Events

Auf dieses Event kann nun eine Event-Methode registriert werden, die beim auslösen des Events aufgerufen wird.

6.5.4.8 Logging

Ein Loggen aller Aktionen sollte durchgeführt werden, um falls nötig nachvollziehen zu können, was alles passiert ist. Diese Informationen können dabei in eine Text-Datei, ein Textfeld im Tool oder ähnliches geloggt werden. Für das Logging könnten nun vorhandene Bibliotheken wie log4net genutzt werden oder ein eigenes Loggingsystem implementiert werden. Das Logging der Aktionen ist sehr simpel, da keine verschiedenen Logginglevel benötigt werden. Deshalb ist eine eigene Implementierung sinnvoll. Ein solches System könnte wie folgt aussehen.

```
public class TestLogger
{
    private static TestLogger logger;
    public static TestLogger GetLogger()
    {
        if(logger == null)
        {
```

```
                return new TestLogger();
            }

            return logger;
        }

        public event EventHandler<LogEventArgs> LogMessage;

        public void Log(String msg)
        {
            if(LogMessage != null)
            {
                LogMessage(this, new LogEventArgs{Message =
msg});
            }
        }
    }
```

Listing 6-11: Klasse zum Erfassen bzw. Loggen von Aktionen die während eines Tests durchgeführt werden

Mittels der Methode „GetLogger“ kann an jeder beliebigen Stelle die Instanz des Loggers zurückgegeben werden. Mittels der Log-Methode kann nun wird ein Event mit der entsprechenden übergebenen Nachricht gefeuert.[4] Auf dieses Event muss nun wieder eine Event-Methode registriert werden die nun zentral die Nachrichten in das Textfeld, die Textdatei oder ähnliches loggen kann.

6.6 Zusammenfassung

Durch eine Evaluierung verschiedener, sowohl kostenfreier als auch kostenpflichtiger Testtools, wurde das Tool „Selenium“ als Grundlage zum automatischen Testen gefunden. Es bietet die Möglichkeit, Aktionen in Webapplikationen auszuführen. Um Testfälle mit Selenium schreiben zu können, werden Bibliotheken für die verschiedensten Programmiersprachen angeboten. Diese Testfälle können dann, entweder lokal oder auf einem entfernten System ausgeführt werden. Auch verschiedenste Browser werden unterstützt. Jedoch kann Selenium für sich nicht alle benötigten Anforderungen erfüllen. So ist zum Beispiel das Schreiben von Testfällen in den jeweiligen Programmiersprachen keinesfalls einfach oder gut wartbar. Weiterhin ist keine hohe Flexibilität gegenüber Änderungen in der Webapplikation gegeben. Um diese Probleme zu umgehen, bedarf es der Implementierung eines eigenen Testtools, welches als Grundlage Selenium nutzt. Das heißt die Ausführung der Aktionen bzw. die Kommunikation mit dem Internet-Browser erfolgt zwar durch Selenium, die restlichen Anforderungen werden aber durch die Eigenimplementierung realisiert. Dafür können verschiedene Konzepte genutzt werden. So ist es zum Beispiel sinnvoll die

[4] Die Klasse „LogEventArgs“ wurde aus übersichtlichkeitsgründen nicht mit aufgeführt

Testfälle in Excel auszulagern, um eine einfache Nutzung auch für Nicht-Informatiker zu gewährleisten, das Einführen einer Abbildung der Webapplikation, um möglichst schnell und flexibel auf Änderungen in der Webapplikation reagieren zu können oder die Möglichkeit Testfälle in andere integrieren zu können, um eine bessere Wartbarkeit dieser zu schaffen.

7 Automatisches Testen von Desktopanwendungen unter Windows

In diesem Kapitel soll das automatische Testen von Desktopanwendungen untersucht werden. Auch hier werden Blackbox-Tests betrachtet. Im ersten Teil werden Besonderheiten, die beachtet werden sollten, genannt. Danach wird der Aufbau von Desktopanwendungen betrachtet. Weiterhin werden die verschiedenen Vorgehensweisen zum automatischen Testen von Desktopanwendungen analysiert.

7.1 Besonderheiten

Für das automatische Blackbox-Testen von Desktopanwendungen gilt es zu beachten, dass keine direkte Schnittstelle zum Zugriff auf die Anwendung vorhanden ist. Das heißt, man kann nur über den Quellcode selbst, falls dieser vorliegt oder über das Betriebssystem ein automatisches Interagieren mit der Anwendung erreichen. Über das Betriebssystem bzw. Dienste, die das Betriebssystem bereitstellt, gibt es verschiedene Möglichkeiten auf die Anwendung zuzugreifen.

7.2 Aufbau von Desktopanwendungen

Eine Desktopanwendung besteht aus einer Oberfläche, die entweder mit vorgegebenen Standardelementen oder eigens erstellten Elementen angefertigt wurde. Dabei sind die Standardelemente ähnlich wie bei Webanwendungen unter anderem Button, Eingabefeld, Auswahlbox usw.[5] Interagiert der Nutzer nun mit diesen Komponenten, so werden im Programm zu der Aktion gehörige Events ausgelöst. Mittels dieser kann dann die eigentliche Logik im Programm ausgeführt werden. Die folgende Abbildung soll den Vorgang vereinfacht darstellen.

Abbildung 7-1: Vereinfachter Ablauf von einer Nutzeraktion bis zum Ausführen der zugehörigen Methode

[5] Im Anhang B befindet sich eine Übersicht über Standardkomponenten, die sowohl in Web- als auch in Desktopapplikationen genutzt werden.

Der Nutzer klickt also beispielsweise einen Button an. Dabei wird dieser Klick über das System an das Programm geschickt. In der Klasse „Button“ die vom .NET-Framework bereitgestellt wird, erfolgt nun ein Auslösen des dazugehörigen Events, welches die, auf das Event registrierte, Methoden aufruft.

7.3 Vorgehensweisen zum automatischen Testen unter Windows

Generell lassen sich zum automatischen Testen von Desktopanwendungen unter Windows vier verschiedene Vorgehensweisen benennen.

Integrieren einer Testschnittstelle	Bei dieser Methode wird, am besten schon bei der Entwicklung, eine Testschnittstelle vorgesehen, um ein automatisiertes Testen zu ermöglichen.
Reflextion	Bei dieser Herangehensweise werden die zu den Oberflächenkomponenten gehörenden Events direkt aufgerufen
Aufruf über eine Schnittstelle zur Barrierefreiheit	In .NET sind verschiedene Schnittstellen zur Barrierefreiheit, u.a. auch zum automatischen Aufrufen von Funktionen vorhanden. Diese könne auch zum Testen genutzt werden.
Simulation mittels Low-Level Mechanismen	Bei dieser Methode werden die Eingaben über das Betriebssystem an die zu testende Applikation geschickt
Simulation über die Gerätetreiber	Es wird auf die Treiber der angeschlossenen Geräte (Tastatur, Maus) zugegriffen und mit diesen die Eingaben simuliert.

Abbildung 7-2: Übersicht der Vorgehensweisen zum automatischen Testen von Desktopanwendungen

7.3.1 Integrieren einer Testschnittstelle

Bei dieser Vorgehensweise, wird eine Testschnittstelle in das zu testende Softwaresystem integriert, um über diese ein automatisiertes Testen zu erreichen und so die Testbarkeit des Programms zu erhöhen. In einer gut entworfenen und strukturierten Applikation ist eine strikte Trennung zwischen der Oberfläche und der eigentlichen Logik vorhanden. So ist dies zum Beispiel bei dem MVC-Muster der Fall. MVC steht dabei für Modell-View-Controller. Dieses Muster schafft eine Unabhängigkeit zwischen Modell und View, sodass ein austauschen des Views möglich wird. Neben dem View muss aber auch der Controller dementsprechend angepasst werden, da dieser den View verwaltet. Das heißt, dass der Controller einerseits Benutzerinteraktionen aus dem View entgegen nimmt und andererseits den View zu verändern.[6] Es ist nun denkbar einen View zu erstellen, welcher Eingaben auf der Oberfläche simuliert. Es ist dabei kein eigentlicher View, sondern ein Konstrukt welches vorgegebene Werte, wie sie auch ein Nutzer eingeben

[6] Es gibt Varianten des MVC-Musters in denen der Controller keine Änderungen am View vornimmt und somit unabhängig vom View ist.

könnte, an die Controller gibt. Dazu müssen allerdings die Controller noch dementsprechend umgebaut werden.

Vorteile und Nachteile

Vorteile dieser Variante ist eine sehr hohe Stabilität und gute Anpassbarkeit an das System.

Nachteile sind der sehr hohe Programmieraufwand durch die Implementierung eines solchen Test-Views und das ein Eingriff in das zu testende Softwaresystem vorgenommen werden muss.

7.3.2 Reflexion

Bei dieser Herangehensweise wird die Automatisierung dadurch erreicht, indem Methoden des zu testenden Programms direkt aufgerufen werden. Dabei werden die Events, die durch die Aktion ausgelöst werden, künstlich aufgerufen. Durch den Aufruf des Events werden, wie es auch bei einer richtigen Nutzeraktion wäre, alle Folgeaktionen ausgelöst. Es muss allerdings trotzdem darauf geachtet werden, dass nur dann entsprechende Events aufgerufen werden, wenn dies auch ein Nutzer mittels einer Interaktion mit dem Programm tun könnte. An manchen Stellen ist es auch notwendig, normale Methoden zusätzlich zu dem Event aufrufen. Zum Beispiel gibt es bei einem Eingabefeld kein Event, was den Text selbst einträgt, sondern nur welche, die auf die Eingabe reagieren. Allerdings gibt es eine Methode zum Eintragen des Textes. Die Methoden, welche die Events auslösen, befinden sich in den jeweiligen Komponenten-Klassen. Also zum Beispiel in der Klasse, die einen Button repräsentiert etc. Ein Problem, was sich dabei herausstellt besteht darin, dass von außen kein Zugriff auf die Methoden, welche diese Events auslösen, gewährleistet ist. Um dieses Problem nun zu umgehen, bedarf es der sogenannten Reflexion. Reflexion ist eine Technik, die es einer Programmiersprache erlaubt, die eigenen Programmstrukturen während der Laufzeit zu analysieren bzw. zu verändern. Das heißt, es ist unter anderem möglich, mittels Reflexion dynamisch Informationen aus Klassen auszulesen oder Methoden aufzurufen, ohne einen direkten Zugriff auf diese besitzen zu müssen. Ein einfaches und häufig genutztes Beispiel ist das Auslesen des Typs einer Klasse. In C# ist es möglich, mittels des Befehls „typeof“ den Typ auszulesen und so zum Beispiel auszugeben, ob es sich um eine Klasse handelt, diese Abstrakt ist usw. Damit diese Funktionalitäten bereitgestellt werden können, müssen beim Kompilieren des Programms einige zusätzliche Informationen gespeichert werden. Unter .NET wird beim Kompilieren der Quellcode in die Intermediate Language, eine „Zwischensprache“ und zusätzliche Metadaten umgewandelt. In der Intermediate Language bzw. den Metadaten werden die benötigten Informationen, die für Reflexion benötigt werden, gespeichert. Beim Ausführen des Programms werden nun diese Informationen

genutzt, um die Reflexions-Befehle abarbeiten zu können. [Mic11] Das folgende Beispiel soll nun zeigen wie man mittels Reflexion ein Button-Click Event auslöst.

```
private void ClickButton()
{
  Type type = typeof (Button);

  object[] parameter = new object[1];
  parameter[0] = EventArgs.Empty;

  MethodInfo methodInfo = type.GetMethod("OnClick",
BindingFlags.NonPublic | BindingFlags.Instance);
  methodInfo.Invoke(myButton, parameter);
}
```

Listing 7-1: Klicken eines Button unter Nutzung von Reflexion

Als erstes werden die Typ-Informationen der Klasse „Button“ mittels des Befehls „typeOf“ ausgelesen. In diesen Informationen sind auch die Methoden-Signaturen und sogar die Methoden-Inhalte zu finden. Als nächstes wird ein Object-Array angelegt, welches die Parameter für den späteren Methoden-Aufruf bereitstellen soll. Als einziger Parameter wird „EventArgs.Empty“, also leere Event-Argumente definiert. Mittels der Funktion „type.GetMethod(…)“ kann nun nach einer Methode bzw. deren Informationen gesucht werden. In dem Fall wird nach der OnClick-Methode gesucht, welche das Click-Event des Buttons auslöst. Als zusätzlichen Parameter können Attribute mitgegeben werden, welche die Suche spezifizieren. Das Attribut „NonPublic“ sorgt dafür, dass auch private Methoden mit in die Suche einbezogen werden und „Instance“, dass Instanzmember, also nicht-statische Methoden, mit einbezogen werden. Nachdem nun die Methoden-Informationen ausgelesen wurden, kann die Methode aufgerufen werden. Dies wird mit der Methode „Invoke“ der MethodInfo-Klasse getan. Als Parameter wird einerseits das Objekt in welchem die Methode aufgerufen wird benötigt, also in diesem Fall das Button-Objekt und andererseits die zuvor definierten Parameter. Das Problem an dem Quellcode ist, dass bei einem echten Nutzer beim Klick auf einen Button, mehr als nur das Klick-Event selbst ausgelöst wird. Eine etwas vollständigere Variante könnte folgendermaßen aussehen.

```
private void ClickButton()
{
  Type type = typeof (Button);

  object[] parameter = new object[1];
  parameter[0] = EventArgs.Empty;
```

```
  MethodInfo gotFocusMethodInfo =
type.GetMethod("OnGotFocus", BindingFlags.NonPublic |
BindingFlags.Instance);
  gotFocusMethodInfo.Invoke(myButton, parameter);

  MethodInfo clickMethodInfo = type.GetMethod("OnClick",
BindingFlags.NonPublic | BindingFlags.Instance);
  clickMethodInfo.Invoke(myButton, parameter);

  MethodInfo lostFocusMethodInfo =
type.GetMethod("OnLostFocus", BindingFlags.NonPublic |
BindingFlags.Instance);
  lostFocusMethodInfo.Invoke(myButton, parameter);
}
```

Listing 7-2: Erweiterte Methode zum Klicken eines Buttons mittels Reflexion

Wie zu sehen ist wurde vor dem Klicken des Buttons dieser noch mittels der Methode „OnGotFocus“ fokussiert und nach dem Klick-Event wurde, mittels „OnLostFocus“, der Fokus genommen. Jeder dieser aufgerufenen Methoden nimmt als Parameter immer nur Event-Argumente. Daher ist es möglich bei allen dieselben leeren Event-Argumente mitzugeben.

Das letzte Problem, welches es noch zu beheben gilt ist, dass man das Objekt des Buttons braucht um diesen klicken zu können. Im Normallfall gibt es von außen keine direkte Möglichkeit, um auf die Objekte zuzugreifen. Allerdings ist es mittels „Application.OpenForms“ möglich die geöffnet Fenster des Programms zu erreichen.[7] Nachdem man also mit der Funktion die Forms hat, kann man das benötigte heraussuchen und dann alle Oberflächenelemente auslesen, um wiederum das benötigte Oberflächenelement zu bekommen. Nachdem dann die Komponente vorhanden ist, folgt der gleiche Ablauf wie schon im vorigen Beispiel aufgezeigt.

```
public void ClickButton()
{
  FormCollection collection = Application.OpenForms;
  Form mainFrm = collection[0];

  Button myButton = (Button)mainFrm.Controls[1];
  …
```

Listing 7-3: Auslesen von geöffneten Fenstern und herausfiltern von Komponenten aus diesen

Durch „Appliaction.OpenForms“ werden die geöffneten Fenster in Form einer Collection zurückgegeben. Da in diesem Beispiel nur ein Fenster vorhanden ist, kann direkt beim Index null die Form ausgelesen werden. Sollten mehrere Fenster

[7] Unter WPF wäre der Befehl „Application.Current.Windows“

vorhanden sein so kann man zum Beispiel durch Überprüfung des Namens das gewünschte Fenster herausfinden. Mittels der Poperty „Controls“ der Form-Klasse können nun die Oberflächenkomponenten ausgelesen werden. In dem Fall ist der Button an Index-Stelle eins. Zurück bekommt man allerdings ein Objekt der Klasse „Control“ welches noch in einen Button gecastet werden muss. Der restliche Ablauf ist nun derselbe wie schon weiter oben beschrieben.

Vorteile und Nachteile

Der große Vorteil an dieser Methode liegt in einer ziemlich hohen Stabilität. Dies wird dadurch erreicht, dass die Events direkt im Programmcode aufgerufen werden und damit keine prozessübergreifenden bzw. hardwarenahen Aktionen nötig sind.

Nachteile sind einerseits der hohe Programmieraufwand der damit verbunden ist, für alle Komponenten die Abfolge von Events, welche die Nutzeraktionen repräsentieren, zu implementieren und auf der anderen Seite ist der Quellcode der Applikation zwingend notwendig.

7.3.3 Schnittstellen zur Barrierefreiheit

Im .NET-Framework werden zur Barrierefreiheit die Frameworks „MSAA“ (Microsoft Active Accessiblity) und „UIA“ (User Interface Automation) bereitgestellt. Sie bieten also die Möglichkeit, Programmabläufe zu automatisieren. Damit ist es auch möglich, die Frameworks zur Testautomatisierung zu nutzen. Dabei ist MSAA eine veraltete Technology, die von UIA abgelöst wurde. [Mic11_1] Da aber in vielen Applikationen immer noch MSAA im Einsatz ist, wird dieses Framework weiterhin unterstützt.

Sowohl MSAA und UIA nutzen Techniken zur Interprozesskommunikation. Das heißt, dass das Testtool unabhängig vom zu testenden Programm in einem anderen Prozess abläuft. Das zu testende Programm wird vom Testtool aus in einem separaten Prozess aufgerufen und dann werden, mittels verschiedener Techniken der Interprozesskommunikation, die Daten bzw. Aktionen an das zu testende Programm geschickt.

MSAA

MSAA basiert grundsätzlich auf dem Component Object Model (COM). COM ist eine der Technologie zur Interprozesskommunikation, die in diesem Fall zur Kommunikation mit dem zu testenden Programm genutzt wird. Unter MSAA wird dazu das Interface „IAccessible“ genutzt. Über dieses COM-Interface ist es also möglich Informationen zur Oberfläche einer Applikation zu bekommen bzw. mit dieser zu interagieren. [Mic11_2] MSAA kann in den Programmiersprachen C/C++, Microsoft Visual Basic 6 und verschiedenen Skript-Sprachen entwickelt werden. [Mic11_3]

UIA

UIA ist wie schon erwähnt die neuere Technologie, welche wenn möglich genutzt werden soll. Falls eine Applikation schon MSAA nutzt, gibt es auch Kompatibilitätsmöglichkeiten zum Arbeiten mit beiden bzw. Migrieren der neuen Version. Um die Kommunikation zwischen den Prozessen zu realisieren wird in dem Fall die DLL „UIAutomationCore.dll“ verwendet. Sie wird sowohl in das Tool für die Barrierefreiheit bzw. zum Testen und die Anwendung selbst geladen und sorgt damit für den Informationsaustausch. Das Klicken eines Buttons mittels des UIA-Frameworks sieht folgendermaßen aus.

```
public void ClickButton()
{
  ProcessStartInfo processStartInfo = new
ProcessStartInfo(@"D:\UIA-Example.exe");

  Process process = Process.Start(processStartInfo);

  process.WaitForInputIdle();

  var mainFrm =
AutomationElement.RootElement.FindFirst(TreeScope.Children,
new PropertyCondition(AutomationElement.ProcessIdProperty,
process.Id));

  var myButton = mainFrm.FindFirst(TreeScope.Children, new
PropertyCondition(AutomationElement.NameProperty,
"Button"));

  InvokePattern pattern =
myButton.GetCurrentPattern(InvokePattern.Pattern) as
InvokePattern;

  pattern.Invoke();
}
```

Listing 7-4: Klicken eines Buttons unter Nutzung des UIA-Frameworks

Im ersten Abschnitt wird das zu testende Programm gestartet. Dazu wird ein neuer Prozess angelegt. Nachdem der Prozess fertig geladen ist, also das Programm bereit zur Nutzung, wird mittels der Klasse „AutomationElement“ des UIA-Frameworks auf das Root-Element zugegriffen und damit nach dem Programm gesucht. Da das Programm ein Kind-Element des Stammelementes ist, reicht es als Reichweite nur nach Kind-Elementen zu suchen. Als Suchkriterium wird die Prozess-ID genutzt. Nachdem nun das Programm und damit das Hauptfenster, als Root-Elemente des Programms, gefunden wurde, kann mittels des Hauptfensters nach dem Button gesucht werden. Es wird dabei wieder in den Kind-Elementen gesucht, jedoch wird

als Suchkriterium der Name des Buttons angegeben. Um mit den einzelnen Komponenten zu interagieren gibt es verschiedene Pattern. Für jede der Standard-Komponenten gibt es Pattern die angeboten werden. Um einen Button zu klicken benötigt man das InvokePattern. Nachdem man mittels der GetCurrentPattern das InvokePattern zur Verfügung hat, muss nur noch die Methode Invoke, zum klicken des Buttons, aufgerufen werden.

Vorteile und Nachteile

Die Vorteile dieser Variante sind, dass keine Abhängigkeit zum Quellcode besteht, eine aktive Unterstützung bzw. Weiterentwicklung durch Microsoft erfolgt und insgesamt auch relativ stabil läuft.

Das größte Problem dieser Methode ist die Interprozesskommunikation, da diese zu unerwarteten Problemen führen kann. Zum Beispiel ist es nicht auszuschließen, das ein Deadlock auftreten kann. Das System bricht dabei zwar mit einer Exception ab, dies verhindert aber trotzdem ein Testen der Anwendung. Das nachfolgend beschriebene Szenario kann zum Beispiel einen Deadlock verursachen: Wenn eine Aktion an das zu testende Programm geschickt wird (Remote Procedure Call), wartet das Programm, welches die Aktion abschickte auf eine Antwort. Falls nun das zu testende Programm nachdem es die Nachricht erhielt, selbst eine Nachricht zurück schickt, bevor es die Antwort gesendet hat, so wartet dann auch dieses auf eine Antwort. Es warten dann also beide Programme auf eine Antwort, ohne jemals eine zu senden bzw. zu bekommen. Der Vorgang soll nachfolgend noch einmal illustriert werden.

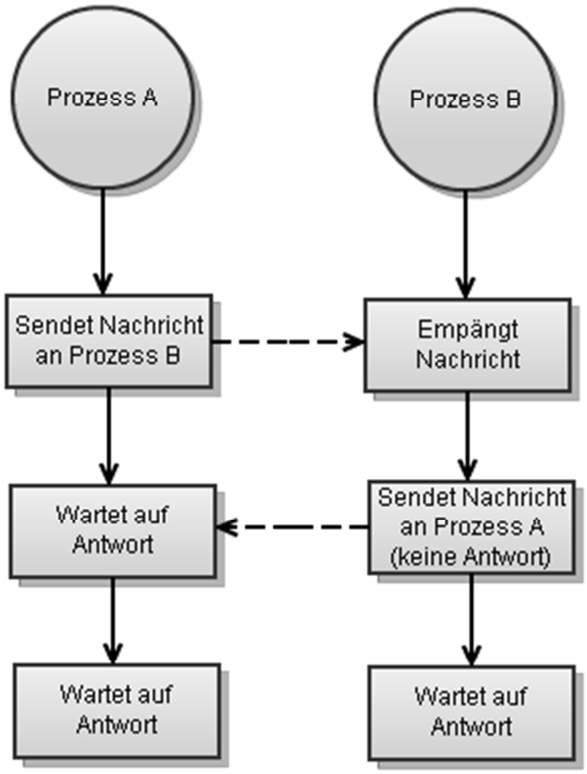


Abbildung 7-3: Entstehen eines Deadlocks bei der Kommunikation zwischen Prozessen

Ein weiterer Zusatzaufwand entsteht, falls in der zu testenden Applikation Custom Controls, also eigens erstellte Komponenten, verwendet werden. Dann muss eine Schnittstelle zum Zugriff auf die Komponente, durch das UIA bzw. MSAA, selbst geschaffen werden.

7.3.4 Low-Level Mechanismen

Bei dieser Methode wird entweder der systeminterne Eingabe/Ausgabe Mechanismus bzw. die Nachrichtenwarteschlangen der Prozesse/Threads zur Automatisierung genutzt. Das heißt, das Testtool und das zu testende Programm laufen in zwei verschiedenen Prozessen ab. Die Eingaben bzw. Aktionen werden aber nicht direkt an das zu testende Programm geschickt, sondern in den systeminternen Eingabe/Ausgabe-Strom bzw. in die Nachrichtenwarteschlange geschrieben. Es können also sowohl Tastatureingaben als auch Mausbewegungen bzw. Mausklicks simuliert werden. Um die Aktionen in den Eingabe/Ausgabe-Strom bzw. in die Nachrichtenwarteschlange schreiben zu können, bedarf es dem WIN32 Subsystem. Dieses System kümmert sich um die Interaktion mit WIN32-Anwendungen, also den normalen Windows-Anwendungen.[8] [Mic11_4]

[8] In 64-Bit Systemen gibt es statt dem WIN32-Subsystem, das WIN64-Subsystem

WIN32-Anwendungen	Weitere Anwendungsarten (DOS, WIN16, Spiele, etc.)
WIN32 Subsystem	
Systemdienste	
Manager (z.B. I/O-Manager, Resources-Manager, Processmanager)	
Hardware-Abstraction-Layer	
Hardware	

Abbildung 7-4: Vereinfachte Architektur von Windows NT

In dieser stark vereinfachten Form der Windows-Architektur wird deutlich, dass sich das Win32 Subsystem direkt unter den WIN32-Anwendungen befindet und damit als Schnittstelle zu anderen Anwendungen bzw. weiter unten liegenden Schichten dient. Das WIN32 Subsystem ist also in der Lage, Aktionen in den systeminternen Eingabestrom zu schreiben oder eben Nachrichten an andere Prozesse zu senden. Dazu muss die Anwendung jeweils passende Aufrufe an die WIN32 API schicken, welche die eigentlichen Funktionen ausführt. Solche API-Aufrufe können zum Beispiel mit C# durchgeführt werden. Nachfolgend ein Beispiel welche eine Nachricht zum Klicken eines Buttons in die Nachrichtenwarteschlange des Prozesses/Threads in dem sich der Button befindet senden.

```
public static void ClickButton()
{
  const string path = @"D:\Example.exe";
  Process p = Process.Start(path);
  p.WaitForInputIdle();

  Thread.Sleep(500);

  IntPtr mwh = FindWindow(null, "Form1");

  IntPtr butt = FindWindowEx(mwh, IntPtr.Zero, null,
"Button");

  const uint WM_LBUTTONDOWN = 0x0201;
  const uint WM_LBUTTONUP = 0x0202;
  PostMessage1(butt, WM_LBUTTONDOWN, 0, 0);
  PostMessage1(butt, WM_LBUTTONUP, 0, 0);
}
```

Listing 7-5: Klicken eines Buttons mittels der WIN32-API

Aus Übersichtlichkeitsgründen wurde der Import, der jeweilig benötigten Funktion zum Senden der Nachrichten an die WIN32-API, weggelassen. Die genutzten

Funktionen zur Interaktion mit der WIN32-Api befinden sich alle in der „User32.dll“, welche zum WIN32 Subsystem gehört und Interaktionen mit der Oberfläche einer Anwendung bereitstellt. Im Anhang befindet sich das komplette Codebeispiel.

Im ersten Abschnitt des gezeigten Codebeispiels wird das Programm gestartet und kurz gewartet, um sicher zu gehen, dass es bereit ist Eingaben entgegen zu nehmen. Als nächstes wird das Fenster des zu testenden Programms gesucht. Dazu wird der Titel des Fensters angegeben und es wird ein Pointer auf das Fenster zurückgegeben. Mittels dieses Pointers kann nun nach Komponenten in dem Fenster gesucht werden, in diesem Fall nach dem Button. Dabei wird wieder der Name angegeben. Danach werden die Nachrichten formuliert. Diese sind einmal das Drücken des linken Mausbuttons und das Loslassen dieses. Die Nachrichten werden dann mit dem betroffenen Pointer zur Komponente, also in dem angegebenen Beispiel dem Pointer zum Button, in die Nachrichtenwarteschlange, für den jeweiligen Thread zu dem die Komponente gehört, geschrieben.

Zum Senden der Aktionen an den Eingabestrom des Systems sind zum Beispiel die API‘s SendInput, SendKeys oder Mouse_Event vorhanden, die von Microsoft angeboten werden. Ähnlich wie bei der anderen Methode, müssen aber auch hier Aufrufe zur WIN32 API durchgeführt werden. Es findet also auch ein Import der nötigen Funktionen aus der „User32.dll“ statt und es werden wieder Messages definiert, die dann an die WIN32-API gesendet werden.

Vorteile und Nachteile

Dadurch, dass die Aktionen aus einem anderen Programm geschickt werden, besteht wiederum keine Abhängigkeit zum Quellcode. Weiterhin ist die Variante, bei der Nachrichten direkt in die Nachrichtenwarteschlange des Prozesses bzw. Threads geschrieben werden, relativ stabil.

Durch die notwendigen Importe und das Definieren der Nachrichten, ist diese Variante ziemlich kompliziert zu implementieren, da bspw. die Nachrichtencodes nicht viel Auskunft über die darunterliegende Aktion geben. Ein weiteres Problem ist, dass es beim Senden der Aktionen immer zu unerwarteten Problemen kommen kann. Wird zum Beispiel mehrfach ein gleichartiger Prozess ausgeführt, so werden die Aktionen nicht ordnungsgemäß ausgeführt. Bei der Variante, die den Eingabestrom nutzt, stellt sich das zusätzliche Problem, dass im Normalfall mit Mausbewegungen gearbeitet wird, um Komponenten auf der Oberfläche zu selektieren, was sehr fehleranfällig ist. Es passiert zum Beispiel häufig, dass die Maus die Komponente nicht richtig trifft und somit der Test nicht weiter ausgeführt werden kann. Zusätzlich gibt es auch keine Rückmeldungen über solche Probleme, da Prinzipiell der Klick erfolgreich durchgeführt wird, nur an der falschen Stelle.

Dadurch kann auch keine geeignete Reaktion auf das Problem erfolgen und der Testfall muss letztendlich abgebrochen werden.

7.3.5 Simulation über Gerätetreiber

Bei dieser Variante werden die Treiber der Tastatur bzw. Maus direkt angesprochen und somit Aktionen ausgelöst. Zum Zugriff auf die Gerätetreiber kann das Windows Driver Kit (WDK) genutzt werden. Es bietet Spezielle SDK's und Frameworks zum Programmieren und zum Zugriff auf Treiber.

Vorteile und Nachteile

Die Vorteile dieser Variante ist, dass es dem Nutzerverhalten sehr nahe kommt. Die Nachteile dieser Methode sind aber so gravierend, dass es kaum praktikabel ist, es sei denn man programmiert Treiber und will diese dann testen. Die Nachteile sind, dass der Zugriff auf die Treiber sehr schwer zu implementieren ist, ein tiefer Eingriff in das Betriebssystem erforderlich und damit eine hohe Fehleranfälligkeit gegeben ist.

7.4 Tools und Frameworks

Auch für das automatische Testen von Desktopanwendungen werden einige, meist kostenpflichtige Tools angeboten. Diese nutzen im Normallfall die Capture-Replay-Methode ähnlich wie die vorgestellten Testtools für Webapplikationen (Kapitel 6.4), nur für Desktopanwendungen. Damit sind auch dieselben Nachteile, also fest eingetragene Wert, Notwendigkeit der Nachbearbeitung der Testfälle, etc. vorhanden. Weiterhin wird beim Abspielen eines Testfalles auch die Maus bewegt, um Komponenten zu fokussieren, was eine erhöhte Fehleranfälligkeit mit sich bringt. Diese Tools nutzen im Normalfall Low-Level Mechanismen oder die Frameworks zur Barrierefreiheit zur Automatisierung.

Neben den meist kostenpflichtigen Tools, gibt es auch einige Frameworks zur freien Nutzung. Dazu gehören unter anderem TestApi und White Project. Beide Frameworks bieten die Möglichkeit mittels Low-Level Mechanismen oder den Schnittstellen zur Barrierefreiheit eine Automatisierung zu ermöglichen. Das heißt, im Prinzip wird nur eine einfacher zu nutzende API bereitgestellt, die aber letztendlich dieselben vorgestellten Methoden nutzt. Damit entstehen natürlich auch die gleichen Probleme wie bei den Methoden, außer dass eine vereinfachte Nutzung möglich ist. Nachfolgend ein Beispiel wie mittels des Frameworks „White Project" ein Button geklickt wird.

```
private static void ClickButton()
{
  Application application =
Application.Launch(@"D:\Example.exe ");
  application.WaitWhileBusy();
```

```
    Window mainFrm = application.GetWindows()[0];
    Button button = mainFrm.Get<Button>("Button");
    button.Click();
}
```

Listing 7-6: Klicken eines Buttons unter Nutzung des Frameworks „White Project“

Im ersten Schritt wird die Applikation mittels einer speziell bereitgestellten Klasse gestartet. Mit dieser können die Fenster die zur Applikation gehören ausgelesen werden. Im Beispiel wird das Hauptfenster, auf dem sich der Button befindet, herausgefiltert. Als nächstes wird der Button ausgelesen und geklickt. In dem Fall bewegt sich die Maus auf den Button und klickt diesen. Im inneren des Frameworks wird dazu die SendInput-API verwendet. Der Quellcode ist mittels des Frameworks übersichtlicher und verständlicher geworden.

7.5 Folgerungen

Grundsätzlich ist eine Testautomatisierung von Desktopanwendungen schwer zu realisieren, da wie schon erwähnt ein hoher Aufwand damit verbunden ist und oft keine gute Stabilität erreicht werden kann. Am besten dafür eignen sich noch die Vorgehensweisen bei der eine Testschnittstelle integriert wird bzw. Events aufgerufen werden, denn diese Herangehensweisen laufen im Gegensatz zu den anderen Stabil genug, um für eine Testautomatisierung genutzt werden zu können. Die anderen Varianten sind wie schon angedeutet nicht Stabil genug. Nur wenn kein Zugriff auf den Quellcode möglich ist und solche Tests zwingend notwendig sind, sollten diese Herangehensweisen in Betracht gezogen werden.

7.6 Zusammenfassung

In diesem Kapitel wurden verschiedene Vorgehensweisen zum automatischen Testen von Desktopanwendungen aufgezeigt und diese evaluiert. Dabei stellte sich heraus, dass es keine einfache Lösung gibt, um eine Testautomatisierung zu erreichen, da keine richtige Schnittstelle zum Einfügen der Aktionen vorhanden ist. Das heißt, es gibt nur die Möglichkeit, über den Quellcode die Aktionen einzufügen oder die Daten von außen über das Betriebssystem an das Programm zu schicken. Beide Varianten können dabei mehr oder weniger schwere Probleme verursachen die eine Testautomatisierung für Desktopanwendungen sehr schwierig gestalten. Auch eine Betrachtung verschiedener Tools bzw. Frameworks zeigt, dass diese ähnliche Probleme verursachen, da sie auch nur die gezeigten Mechanismen nutzen, bzw. sehr teuer in der Anschaffung sind. Das heißt, es ist zwar grundsätzlich möglich ein automatisches Testen für Desktopanwendungen durchzuführen, jedoch bringt dies einen sehr hohen Aufwand mit sich, um eine ausreichend stabile Lösung zu schaffen.

8 Zusammenfassung und Ausblick

Das Ziel dieser Arbeit bestand darin, Vorgehensweisen und Konzepte zum automatischen Testen von Webapplikationen und Desktopapplikationen zu geben. Dabei sollten einerseits vorhandene Testtools und andererseits eine Eigenimplementierung betrachtet werden.

Zu Anfang wurde das Auslieferungsmanagement betrachtet. Dieses enthält als wichtigen Schritt einen abschließenden Softwaretest des gesamten Systems. Dieser Schritt sollte mit einer Automatisierung vereinfacht werden, um somit eine Effizienzsteigerung zu erreichen. Weiterhin wurden Grundlagen zum Testen von Software im Allgemeinen gegeben.

Es wurden verschiedene Vorgehensweisen zum automatischen Testen, sowohl für Web- als auch für Desktopapplikationen betrachtet und deren Nutzbarkeit in realen Projekten bewertet. Weiterhin wurden Vor- und Nachteile von Testtools bzw. der Eigenimplementierung eines Testtools genannt. Dabei stellte sich unter den Webapplikationen Selenium als geeignetes Tool heraus, welches als Grundlage für eine eigene Implementierung genutzt wurde. Für diese Eigenimplementierung wurden Konzepte gegeben, welche alle Anforderungen erfüllten, die an das Testtool gestellt wurden.

Eine genauere Untersuchung der Browserkompatibilität bei Selenium wäre denkbar. Werden neue Browser-Version veröffentlicht, so ist ein Testen mit diesen neuen Versionen meist erst nach dem Update von Selenium möglich. Allerdings gibt es mit Selenium bei den verschiedenen Browserversionen generelle Kompatibilitätsprobleme, sodass einige, zum Teil für die Tests sehr wichtige, Funktionen Fehler verursachen.

Für das automatische Testen von Desktopapplikationen zeigte sich, dass es schwierig ist, eine Automatisierung zu erreichen, die alle Anforderungen erfüllt. Bei den meisten Vorgehensweisen besteht eine erhöhte Instabilität bei der Nutzung, sodass ein automatisches Testen mit diesen sehr fehleranfällig ist. Die einzigen Herangehensweisen, die eine ausreichend stabile Automatisierung ermöglichen, müssen entweder Zugriff auf den Quellcode haben oder es muss direkt eine Schnittstelle zum Testen mit in das Programm integriert werden. Diese Varianten bringen jedoch einen erhöhten Programmieraufwand mit sich.

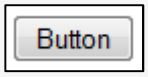
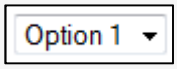
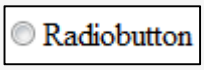
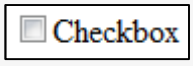
Zukünftig wäre die Eigenimplementierung eines Testtools für Desktopapplikationen unter Nutzung der vorgestellten Vorgehensweisen. Es wäre zu überprüfen, ob es möglich ist, ein Testtool zu implementieren, welches Anforderungen, wie eine einfache und flexible Nutzung gewährleistet. Auch eine genauere Evaluierung der vorhandenen Testtools wäre zu überlegen.

Insgesamt lässt sich sagen, dass eine Automatisierung einer Webapplikation mit überschaubarem Aufwand realisierbar ist, während bei der Automatisierung einer Desktopapplikation mit einem hohen Aufwand zu rechnen ist.

Anhang A: *Klicken eines Buttons mittels der WIN32 API (Komplett)*

```
[DllImport("user32.dll", EntryPoint = "FindWindow", CharSet
= CharSet.Auto)]
static extern IntPtr FindWindow(string lpClassName, string
lpWindowName);

[DllImport("user32.dll", EntryPoint = "FindWindowEx",
CharSet = CharSet.Auto)]
static extern IntPtr FindWindowEx(IntPtr hwndParent, IntPtr
hwndChildAfter, string lpszClass, string lpszWindow);

[DllImport("user32.dll", EntryPoint = "PostMessage",
CharSet = CharSet.Auto)]
static extern bool PostMessage1(IntPtr hWnd, uint msg, int
wParam, int lParam);

public static void ClickButton()
{
  const string path = @"D:\Example.exe";
  Process p = Process.Start(path);
  p.WaitForInputIdle();

  Thread.Sleep(500);

  IntPtr mwh = FindWindow(null, "Form1");

  IntPtr butt = FindWindowEx(mwh, IntPtr.Zero, null,
"Button");

  const uint WM_LBUTTONDOWN = 0x0201;
  const uint WM_LBUTTONUP = 0x0202;
  PostMessage1(butt, WM_LBUTTONDOWN, 0, 0);
  PostMessage1(butt, WM_LBUTTONUP, 0, 0);
}
```

Anhang B: *Übersicht über Standardkomponenten die sowohl in Web- als auch in Desktopapplikationen genutzt werden*

Button	
	Beim Klicken auf diese Komponente wird eine Aktion ausgelöst
Eingabefeld	
	In diese Komponente können, über die Tastatur, Informationen eingegeben werden
Auswahlliste	
	Mit dieser Komponente kann aus festgelegten Optionen ausgewählt werden
Radiobutton	
	Diese Komponente tritt Normalerweise in Gruppen auf. In so einer Gruppe wird einer der Radiobuttons ausgewählt.
Checkbox	
	Dies ist eine Komponente die an- und abgewählt werden kann.

Literaturverzeichnis

[Ank11] Ankur. www.learnqtp.com. [Internet]. [zitiert am 2011 Aug. 11]. Unter: www.learnqtp.com/qtp-now-supports-silverlight-ajax-google-webtoolkit-dojo-yahoo-user-interface.

[Apa11] Apache Software Foundation. http://maven.apache.org. [Internet]. [zitiert am 2011 Aug. 26]. Unter: http://maven.apache.org/what-is-maven.html.

[Aut11] Automation Anywhere, Inc. www.automationanywhere.com. [Internet]. [zitiert am 2011 Aug. 11]. Unter: www.automationanywhere.com/Testin/products/testing-features.htm.

[Hof08] Hoffman. *Software-Qualität*. Springer-Verlag, Karlsruhe, 2008.

[IEE90] IEEE - Institute of Electrical and Electronics Engineers. *Standard glossary of software engineering terminology*. New York, 1990.

[Int01] International Organization for Standardization. *ISO/IEC 9126*. 2001.

[Jam11] Marshall, James. www.jmarshall.com. [Internet]. [zitiert am 2011 Aug. 10]. Unter: http://www.jmarshall.com/easy/http/#postmethod.

[Kla07] Franz, Klaus. *Handbuch zum Testen von Webapplikationen: Testverfahren, Werkzeuge, Praxistipps*. Springer Berlin Heidelberg, Berlin, 2007.

[Kur07] Schneider, Kurt. *Abenteuer Softwarequalität*. dpunkt.verlag Gmbh, 2007.

[Mic11] Microsoft. MSDN. [Internet]. [zitiert am 2011 Aug. 17]. Unter: http://msdn.microsoft.com/de-de/magazine/cc163408.aspx.

[Mic11_1] Microsoft - MSAA and UIA compared. MSDN. [Internet]. [zitiert am 2011 Aug. 17]. Unter: http://msdn.microsoft.com/en-us/library/dd561918(v=VS.85).aspx.

[Mic11_2] Microsoft - Windows Accessibility for Developers. MSDN. [Internet]. [zitiert am 2011 Aug. 17]. Unter: http://msdn.microsoft.com/de-de/windows/gg712214.

[Mic11_3] Microsoft - UIA and MSAA. MSDN. [Internet]. [zitiert am 2011 Aug. 17]. Unter: http://msdn.microsft.com/en-us/library/ms788733.aspx#Programming_Languages_compare.

[Mic11_4] Microsoft - Windows Architecture. Technet. [Internet]. [zitiert am 2011 Aug. 18]. Unter: http://technet.microsoft.com/en-us/library/cc768129.aspx.

[Pet11] Consulting, Peterjohann. www.peterjohann-consulting.de. [Internet]. [zitiert am 2011 Aug. 23]. Unter: http://www.peterjohann-consulting.de/index.php?menu-id=sw.

[SEL11] SELFHTML e.V. de.selfhtml.org. [Internet]. [zitiert am 2011 Aug. 30]. Unter: http://de.selfhtml.org/intor/technologien/html.htm.

[Sim11] Stewart, Simon. Google-Opensource. [Internet]. [zitiert am 2011 Aug. 11]. Unter: http://google-opensource.blogspot.com/2009/05/introducing-webdriver.html.

[Tel11] Telerik. www.telerik.com. [Internet]. [zitiert am 2011 Aug. 11]. Unter: www.telerik.com/automated-testing-tools/benefits.aspx.

[Tho11] ThoughtWorks - Project Site. http://seleniumhq.org. [Internet]. [zitiert am 2011 Aug. 11]. Unter: http://seleniumhq.org/projects.

[Tho11_1] ThoughtWorks. http://seleniumhq.org. [Internet]. [zitiert am 2011 Aug. 12]. Unter: http://seleniumhq.org/projects/ide.

[Tho11_2] ThoughtWorks - Selenium RC. http://seleniumhq.org. [Internet]. [zitiert am 2011 Aug. 12]. Unter: http://seleniumhq.org/docs/05_selenium_rc.html#how-selenium-rc-works.

[Tho11_3] ThoughtWorks - Selenium Grid. Selenium-Grid. [Internet]. [zitiert am 2011 Aug. 12]. Unter: selenium-grid.seleniumhq.org.

[Tho11_4] ThoughtWorks - Selenium RC - Overview. http://seleniumhq.org. [Internet]. [zitiert am 2011 Aug. 17]. Unter: http://seleniumhq.org/projects/remote-control.

[Wat11] WatiN. http://watin.org/. [Internet]. [zitiert am 2011 Aug. 11]. Unter: http://watin.org/.

Selbstständigkeitserklärung

Ich erkläre hiermit, dass ich die Bachelorarbeit ohne fremde Hilfe angefertigt, und nur die im Literaturverzeichnis angeführten Quellen und Hilfsmittel benutzt habe.

Alle wörtlich oder sinngemäß übernommenen Textstellen habe ich als solche kenntlich gemacht.

Ort, Datum

Unterschrift