Fachhochschule Südwestfalen
University of Applied Sciences

Software Testing: Gängige Testwerkzeuge im Vergleich

Bachelorarbeit
im Rahmen des Moduls „Bachelorprüfung“

Wintersemester 2021/2022

Prüfer: Prof. Dr. Andreas de Vries

Prof. Dr. Christian Leubner

Abgabetermin: 2021-10-11

Inhalt

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

ADB	Android Debug Bridge
API	Application Programming Interface
DOM	Document Object Model
GUI	Grafical User Interface
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IDE	Integrated Development Environment
QF-Test	Quality First Test
QTP	Quick Test Professional
PoC	Point of Control
PoO	Point of Observation
SAP	Systeme Anwendungen Produkte
SDK	Software Development Kit
SUT	System Under Test
UFT	Unified Functional Testing
URL	Uniform Ressource Locator

1 Einleitung

Softwaretechnologien haben sich in den letzten Jahren so schnell und umfangreich weiterentwickelt, dass Entwickler kaum noch Zeit hatten, sich zusätzlich um die Softwarequalität zu kümmern. Daher ist Testen ein immer wichtigerer Teil der Softwareentwicklung geworden, um die Entwickler zu entlasten.

Da in der heutigen Zeit das Leben der Menschen von Computersoftware abhängt, ist es notwendig, wenn eine professionelle Umsetzung erfolgt. Dies ist ein Grund, warum Softwaretesten von größter Bedeutung ist. Testen muss vor dem Einsatz der Software durchgeführt werden, denn dies hilft dabei, Fehler rechtzeitig zu entdecken und sicherzustellen, dass die Funktionalitäten der Software wie erforderlich funktionieren. Softwaretest ist ein wichtiger Teil der Software-Qualitätssicherung, eine Aktivität zur Bewertung und Verbesserung der Software-Qualität (Vgl. F. Okezie et al 2019 S. 1).

In den letzten Jahren kam es häufig vor, dass Software fast fertiggestellt war, bevor sie getestet wurde, weil viele Unternehmen noch nicht über definierte Testabteilungen verfügten. Nach einigen negativen Erfahrungen, die daraus gesammelt wurden, gibt es nun eine Einigkeit darüber, wie bedeutsam die Schulung und das Bewusstsein für die Wichtigkeit von Softwaretests während der Entwicklung inzwischen ist. Das Testen von Software ist notwendig, um den Anspruch an eine hohe Lieferqualität zu erfüllen, egal ob die Software an Kunden oder interne Anwender ausgeliefert wird (Vgl. Witte (2019), S. 4.).

Im Rahmen dieser Arbeit wird der Frage nachgegangen, wo die Unterschiede, Gemeinsamkeiten, Probleme und Grenzen zwischen den einzelnen Testtools liegen. Zusammenfassend werden die Testwerkzeuge miteinander verglichen. Dabei muss das Erreichen der Ziele, Qualität, und Zeit möglich sein, wobei auch die ökonomischen Aspekte berücksichtigt werden müssen.

Außerdem werden verschiedene Testwerkzeuge mittels unterschiedlicher Kriterien miteinander verglichen. In Kapitel 2 werden die für diese Arbeit relevanten Begriffe, wie Softwarequalität, Tests, Fehler, Testwerkzeuge und Testaufwand genau definiert. In Kapitel 3 werden die grundlegenden Testverfahren und Testarten vorgestellt. Zu den Testverfahren gehören der statische und der dynamische Test. Im statischen Test werden die werkzeugunterstützte Analyse sowie die strukturierten Gruppenprüfungen - auch Reviews genannt - genauer betrachtet. Im dynamischen Test werden die Blackbox- und die Whitebox-Verfahren erläutert. Zu den Testarten gehören der Unit Test, Integrationstest, Systemtest, Abnahmetest, funktionaler Test, nicht funktionaler Test, Negativtest, Regressionstest sowie End-to-End-Test. In Kapitel 4 wird das Fehlermanagement genau erläu-

tert. Daraufhin wird in Kapitel 5 eine Kategorisierung der Testwerkzeuge dargestellt. Dabei werden zwischen Tools für statischen Test und Tools für dynamischen Test unterschieden. Im Kapitel 6 werden die gängige Tools zueinander verglichen. Dabei wird definiert, was unter diesen Testwerkzeugen verstanden wird und wie diese funktionieren. Da die Lizenz von drei Tools nicht zugänglich war, darunter „Ranorex“, „TestComplete“, „UFT (Unified Functional Testing)“, wird es in dieser Arbeit nur auf die Beschreibung dieser Tools eingegangen. Weiterhin wird im Kapitel 7 gezeigt, wo die Testtools nach Systemart eingesetzt werden können. Schließlich werden im Kapitel 8 unter Berücksichtigung der Programmierkenntnisse und der Wirtschaftlichkeitsüberlegungen den Benutzern Empfehlungen vorgelegt.

2 Begriffsdefinitionen

Es ist wichtig zu verstehen, wie die Kernbegriffe des Testens zu definieren sind. Zu diesem Zweck werden in diesem Kapitel gängige Testbegriffe erklärt, die die Grundlagen für die folgenden Kapitel darstellen.

2.1 Softwarequalität

Es wird in mehreren Normen festgelegt, was Qualität von Software ist und welche Merkmale diese Qualität bestimmen. Die [DIN 55350-11 95] definiert die Qualität als „die Beschaffenheit einer Einheit bezüglich ihrer Eignung, festgelegte und abgeleitete Erfordernisse zu erfüllen“ (Vgl. Liggesmeyer (2002), S. 5). Weiterhin wird nach ISO-Norm 9126 der Begriff Softwarequalität als „die Gesamtheit der Merkmale und Merkmalswerte eines Software-Produkts, die sich auf dessen Eignung beziehen, festgelegte Erfordernisse zu erfüllen“, definiert (Vgl. Witte (2019), S. 22). Laut dieser Norm wird Softwarequalität wie bereits erläutert in folgenden Qualitätsmerkmalen behandelt.

- **Funktionalität**

  Unter Funktionalität werden vorhandene Funktionen mit vordefinierten Merkmalen verstanden. Diese Funktionen sollen diese definierten Aspekte erfüllen: Angemessenheit, Korrektheit, Ordnungsmäßigkeit.

- **Benutzbarkeit**

  Durch dieses Merkmal kann die Benutzergruppe eine Software unter bestimmten Bedingungen verstehen, erlernen und anwenden.

- **Zuverlässigkeit**

  Darunter wird verstanden, wie die Software ihr Leistungsniveau unter bestimmten Bedingungen für einen definierten Zeitraum beibehält.

- **Übertragbarkeit**

  Durch die Übertragbarkeit kann die Software von einer Systemumgebung in eine andere übertragen werden.

- **Änderbarkeit/Wartbarkeit**

  Die Änderbarkeit beschreibt den Aufwand, der notwendig ist, um bestimmte Änderungen an der Software vorzunehmen. Diese Änderungen sind: die Modifizierbarkeit, die Stabilität die Testbarkeit und die Konformität eines Systems.

- **Effizienz**

  Hier handelt es sich um das Verhältnis zwischen dem Leistungsniveau der Software und dem Umfang der eingesetzten Betriebsmittel unter festgelegten Bedingungen (Vgl. Witte (2019), S. 23).

2.2 Tests

Ein Test ist der Nachweis dazu, dass eine Funktionalität eines Softwaresystems korrekt umgesetzt worden ist. Er entspricht die Ausführung eines Testfalls durch den Tester, welcher hierzu Testdaten verwendet. Für jeden Test sollten folgende Informationen festgehalten werden:

- Tester
- Datum und Uhrzeit des Tests
- System, auf dem der Test durchgeführt wurde
- wann der Test als erfolgreich abgeschlossen ist
- Ergebnis des Tests
- Fehlerbehebung

Diese Hinweise helfen beim Nachvollziehen der Ergebnisse. Dadurch kann der Entwickler den Fehler reproduzieren und die Fehlerursache feststellen, damit dieser dann korrigiert werden kann. Es kann auch sein, dass die Anforderungen besser zu definieren sind (Vgl. Witte (2019), S. 23).

Zur Behebung des Fehlers, muss er in der Software gefunden werden. Zunächst ist nur die Auswirkung bekannt, nicht aber die genaue Stelle in der Software, an der der Fehler auftritt. Das Auffinden und Beheben des Fehlers ist die Aufgabe des Softwareentwicklers und wird auch Debugging (Fehlerkorrektur) genannt. Das Debuggen wird oft mit dem Testen gleichgesetzt, was aber nicht korrekt ist, da die Bedeutung diese beiden Begriffe völlig unterschiedlich ist. Beim Debugging werden die Fehlerzustände behoben, während beim Testen Fehlerwirkungen aufgedeckt werden (Vgl. Spillner & Linz (2019), S. 8).

Die Planung und Durchführung von Softwaretests erfolgen durch festgelegte Ziele. Darüber hinaus kann das Ziel des Testens unterschiedlich sein:

- **Fehlerkosten reduzieren**

Die Korrektur von Fehlern, sobald die Software beim Kunden im Einsatz ist, kann enorme finanzielle Kosten verursachen und diese Fehler können durch Softwaretests reduziert werden, was wiederum Qualitätssicherungskosten verursacht. Diese Kosten können in zwei Gruppen aufgeteilt werden. Dies sind Fehlerkosten und Fehlerwirkungskosten, wobei Fehlerkosten den Aufwand zur Identifikation von Fehlern im Softwareprodukt und deren Behebung beschreiben. Fehlerwirkungskosten sind alle Kosten, die als negative Folge des Fehlers resultieren.

- **Softwarequalität beurteilen**

Softwaretest ermöglicht die Überprüfung der Softwarequalität durch Einhaltung der Qualitätsmerkmale. Mögliche Qualitätsmerkmale sind wie bereits im Kapitel 2.1.1 erörtert wurde Funktionalität, Zuverlässigkeit, Benutzbarkeit, Effizienz, Änderbarkeit und Übertragbarkeit.

- **Erfüllung von Standards und gesetzlichen Vorgaben**

Unternehmen sollen beim Softwaretest die erforderten Normen und Standards oder gesetzliche Regeln einhalten, weil dadurch der globale Marktzugang, die Unterstützung im internationalen Handel, die Wirtschaftlichkeit, die Unterstützung von Innovationen sichergestellt wird.

- **Vertrauen schaffen**

Vertrauen kann durch gute Testergebnisse aufgebaut werden. Auf der Seite des Testers und der Kunden muss deutlich sein, dass der Test kein Selbstzweck ist. In diesem Fall kann die Qualitätssicherung nicht auf den Kunden übertragen oder von ihm im Leistungsumfang begrenzt werden. Darüber hinaus kann Vertrauen zwischen dem Auftraggeber und den Softwaretestern aufgebaut werden, wenn beide transparent agieren und sich aktiv am Testprozess beteiligen. Der Kunde erhält durch die regelmäßige fachliche Auseinandersetzung mit den Sachverhalten ein vertieftes Wissen über die Arbeitsweise der am Projekt beteiligten Personen und der Softwaretester kann wiederum in dieser Situation als Experte auftreten und den Auftraggeber entsprechend beraten (Vgl. Torsten Cleff (2010), S. 60).

2.3 Fehler

Der Fehler ist definiert, als die spezifizierten Anforderungen nicht zu erfüllen (Franz (2015), S. 31). Im Zusammenhang mit Fehlern gibt es vier Begriffe, die unterschieden

werden müssen und den Begriff des Fehlers erweitern. Diese Begriffe werden im Folgenden erläutert:

- **Fehlerwirkung**

  Fehlerwirkung definiert die Auswirkung oder das Auftreten eines Fehlers, d. h. das beobachtbare Versagen des Systems während der Programmausführung. Dieser Fehler wird am häufigsten vom Tester oder Benutzer beobachtet oder bemerkt, während die Software läuft. Andere Begriffe dafür sind Fehlfunktion, externer Fehler oder Ausfall. Der englische Fachbegriff lautet „failure“.

- **Fehlerzustand**

  Der Fehlerzustand beschreibt beispielsweise eine falsch programmierte oder vergessene Anweisung im Programm und stellt die Ursache für die obige Fehlerwirkung dar. Eine Fehlerwirkung geht immer einem Fehlerzustand voraus, die ihn auslöst. Andere Bezeichnungen für den Fehlerzustand wären defekt oder innerer Fehler. Der Fachbegriff hierfür ist „fault“, obwohl der Begriff „bug“ in der Softwareentwicklung gebräuchlicher und bekannter ist.

- **Fehlhandlung**

  Eine Fehlhandlung liegt vor, wenn beispielsweise eine fehlerhafte Programmierung durch den Entwickler besteht. Diese ist wiederum der Grund für das Auftreten des Fehlerzustands. Zudem führt die durch eine Person verursachte Fehlhandlung zu einem Fehlerzustand, der wiederum zu einer Fehlerwirkung führt, soweit der Fehler durch den Test aufgedeckt wird. Der englische Fachbegriff dafür lautet „Error“

- **Fehlermaskierung**

  Durch Fehlermaskierung wird ein Fehlerzustand durch andere Fehlerzustände in anderen Teilen des Programms unsichtbar. Nachdem ein Fehlerzustand korrigiert wurde, können somit Seiteneffekte auftreten, welche in diesem Fall zu noch mehr Fehlerwirkungen führen können (Vgl. Spillner & Linz (2019), S. 8).

2.4 Testwerkzeuge

Für den gezielten Einsatz von Testwerkzeugen ist es notwendig zu unterscheiden, welche Werkzeuge zur Verfügung stehen und für welche Aufgaben diese eingesetzt werden können, da für jede Testaktivität ein eigenes Werkzeug erforderlich ist. Außerdem ist es notwendig für die Testplanung, Informationen über die Funktionalität und Qualität der in Frage kommenden Tools und den Zeitbedarf für die Einarbeitung zu berücksichtigen. Der

Test erfordert unterschiedliche Einarbeitungen, z. B. in den Bereichen Skriptprogrammierung und Fachmethodik. Wenn der Testmanager gut betreut ist, kann die Effizienz in diesem Bereich deutlich erhöht werden (Vgl. Witte (2020), S. 34).

2.5 Testaufwand

Es werden in der Regel Daten aus der Vergangenheit benötigt, um den Testaufwand abschätzen zu können. Hierbei handelt es sich um die übernommene Fehlerhäufigkeit (Anzahl der in früheren Projekten gefundenen Fehler) und Fehlerdichte im Sinne der Anzahl der Fehler pro 1000 oder 100 Größeneinheiten als Maß auf das neue System. Die Codezeilen, Anweisungen, Funktionspunkte oder sogar Testfälle sind als Maßeinheit bezeichnet und müssen durch den Grad der Testabdeckung bedingt werden. Es reicht also nicht aus, nur die Anzahl der absoluten Fehler zu kennen, sondern es müssen auch die Größe der betreffenden Software und der Grad ihrer Testabdeckung betrachtet werden.

Zudem müssen so viele effektive Testfälle durch den Tester entwickelt werden, damit potenzielle Fehlerursachen identifiziert werden. Die Produktivität ist beim Systemtest ein Maß für die Anzahl der Testfälle, die ein Tester pro Zeiteinheit durchführen kann. Diese Anzahl ist von dem Projekt, der Testautomatisierung und der Erfahrung sowie Kreativität des Testers abhängig. Jedoch können zukünftige Leistungen nur auf der Grundlage der bisherigen Produktivität geschätzt werden. Hierfür werden entsprechende Erfahrungswerte für die Testplanung benötigt, aus denen sich das Verhältnis der Anzahl der durchgeführten Testtage zur Anzahl der ausgeführten Testfälle ableiten lässt. Um die Testautomatisierung zu evaluieren, sollte der Automatisierungsgrad der Testfallerstellung, der Testdurchführung und des Testreportings im Vorfeld genau festgelegt werden. Obwohl die Vorbereitung und Planung der Tests sowie die Bewertung der Testergebnisse einen oft unterschätzten Aufwand benötigen, wird häufig nur die Durchführung der Tests berücksichtigt (Vgl. Witte (2020), S. 34).

3 Software Testing Methoden

3.1 Testverfahren

Nachdem im letzten Kapitel die Grundbegriffe des Softwaretestens erläutert wurden, sollen nun in diesem Kapitel die Testverfahren genau betrachtet werden. Bei den Testverfahren muss grundsätzlich zwischen statischen und dynamischen Testverfahren unterschieden werden. Beim statischen Testen wird das Testobjekt nicht ausgeführt. Beim dynamischen Testen hingegen geht es um die Ausführung des Testobjektes (Programmcode).

3.1.1 Statische Tests

Ein statischer Test bzw. eine statische Analyse ist eine Überprüfung des Testobjekts an sich, ohne dass es ausgeführt wird. Dabei werden strukturierte Gruppenprüfungen und werkzeugunterstützte Analyse unterschieden.

- **Strukturierte Gruppenprüfungen**

  Hier werden durch intensives Lesen und Nachvollziehen der untersuchten Dokumente die Analyse und Denkfähigkeit der Menschen angefordert, um komplexe Sachverhalte zu prüfen und zu bewerten. Das heißt, das Programm oder Dokument wird durch eine oder mehrere Personen analysiert, was auch als Reviews bezeichnet wird. Reviews sind geeinte Verfahren, um die Qualität der untersuchten Dokumente zu sichern und sollten so schnell wie möglich nach Fertigstellung der Dokumente durchgeführt werden, damit Fehler und Unstimmigkeiten frühzeitig erkannt werden. Die Behebung von Fehlern und Abweichungen führt zu einer Verbesserung der Qualität der Dokumente und hat einen positiven Effekt auf den gesamten Entwicklungsprozess, da die Entwicklung mit weniger oder sogar fehlerfreien Dokumenten fortgesetzt wird. Zusätzlich kann die Fehlerbeseitigung kostengünstig sein, wenn diese Fehler durch Reviews frühzeitig entdeckt wurden.

  Reviews können sich mittels untersuchten Testobjekts in zwei Gruppen aufteilen:

  - Reviews, die sich auf Produkte oder Teilprodukte beziehen, die während des Entwicklungsprozesses erstellt werden.

  - Reviews, die das Projekt oder den Entwicklungsprozess selbst analysieren.

Bei der ersten Gruppe der Reviews werden folgende Review-Arten betrachtet (Vgl. Spillner & Linz (2019), S. 92):

Walkthrough

Bei dieser Art von Review stellt der Autor den Reviewern seine Dokumente während einer Sitzung vor, wobei die Beteiligten gegebenenfalls Fragen stellen können. Die Vorbereitung ist im Umfang begrenzt. Der Walkthrough lässt sich besonders gut in kleinen Entwicklungsgruppe durchführen, um Alternativen zu überdenken und Wissen im Team zu verteilen.

Inspektion

Die Inspektion ist die formalste Review-Art, welche von einem ausgebildeten Moderator durchgeführt wird. Bei der Vorbereitung der Inspektion ist eine Checkliste mit Eingangs- und Ausgangskriterien für die Prüfschritte erforderlich. Zweck dieser Prüfung ist es, die Qualität des Testobjekts zu kontrollieren und den Entwicklungs- und Testprozess zu verbessern.

Technisches Review

Beim technischen Review kann der Autor im Gegensatz zu Walkthrough unbekannt bleiben und die Ergebnisse der Prüfer werden durch den Sitzungsleiter vorab schriftlich vorgelegt, um sie nach ihrer angeblichen Wichtigkeit zu bewerten. Dabei ist es möglich zu diskutieren, Alternativen zu bewerten und schließlich Entscheidungen zu treffen.

Informelles Review

Diese Art von Review ist als informelles Review bezeichnet, da weder der formalisierte Prozess noch die Form der Ergebnisdokumentation festgelegt sind. Aufgrund des geringeren Aufwands wird diese Review-Art häufig in der Praxis anerkannt und verwendet.

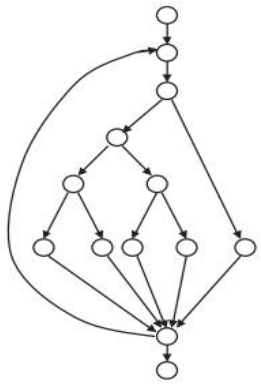
Bei der zweiten Gruppe des Reviews handelt es sich um Managementreviews oder Projektreviews, die den Entwicklungsprozess analysieren. Hier sollen die geplanten Anforderungen und die Erfüllung der notwendigen Aufgaben begutachtet werden. Die Beurteilung des Projektfortschritts ist nach technischen, wirtschaftlichen, zeitlichen und Management-Aspekten zu betrachten (Vgl. Spillner & Linz (2019), S. 92 ff.).

- **Werkzeugunterstütze Analyse**

Hier wird die Identifizierung von vorhandenen Fehlern im Dokument durch Werkzeuge durchgeführt. Damit ein Dokument von einem Werkzeug geprüft werden kann, muss es eine vorgegebene formale Struktur beinhalten. Es gibt zwei Analysewerkzeuge, die die Analyse von Programmcodes ermöglichen - einerseits der Compiler, der die Konformität mit der Syntax der betreffenden Programmiersprache überprüft und andererseits die Analysatoren, welche die Einhaltung von Konventionen oder Codier-Standards überprüfen können.

Weiterhin ist auch möglich, durch Datenflussanalyse Fehler aufzudecken. Mittels dieser Analyse kann die Verwendung von Daten auf Pfaden durch den gesamten Programmcode geprüft werden. Die daraus resultierenden Fehler werden Datenflussanomalie genannt. Diese Datenflussanomalien weisen nicht auf explizite Fehler hin, sondern nur auf potenzielle Fehlerquellen durch ungewöhnliche Datenflüsse oder variable Nutzung.

Zudem gibt es auch die Kontrollflussanalyse, die mit Hilfe eines Kontrollflussgraphen durchgeführt wird. Durch die Kontrollflussgraphen können die Abläufe in einem Programm leicht erfasst und mögliche Anomalien festgestellt werden. Bei dem Graph sind ein Knoten als eine Verzweigung im Codestück und eine Kante als eine Verbindung zur nächsten Verzweigung bezeichnet, wobei eine Verzweigung beispielsweise als eine Entscheidung (IF-Anweisungen), eine Schleife oder als ein Rücksprungspunkt im Code angesehen werden können (Vgl. Spillner & Linz (2019), S.102). In der Abbildung 1 ist ein Beispiel von Kontrollflussgraph dargestellt.



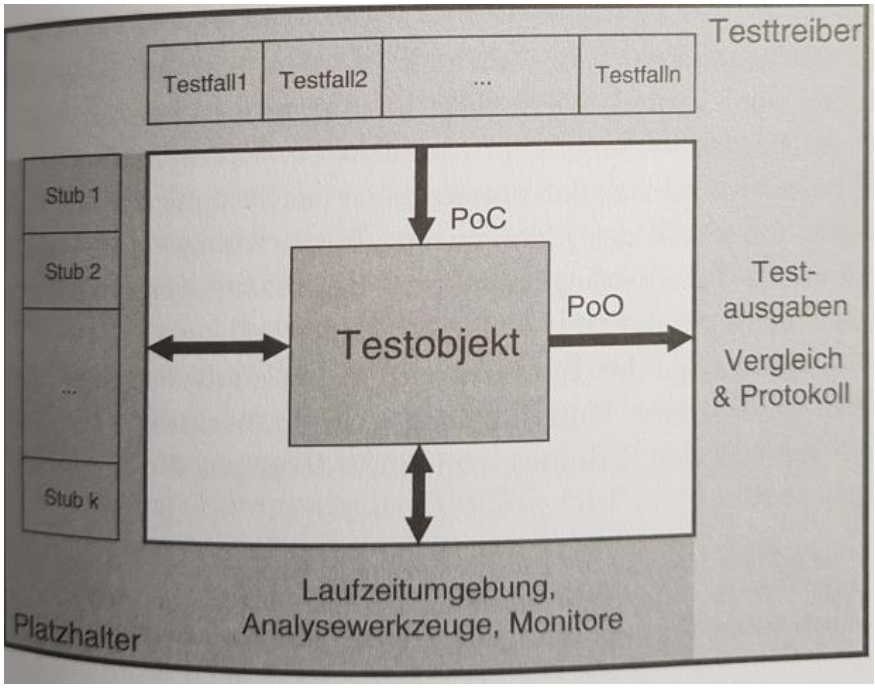
QUELLE: Spillner & Linz (2019)

ABBILDUNG 1: BEISPIEL VON KONTROLLFLUSSGRAPH

3.1.2 Dynamische Tests

Im Gegensatz zu statischen Tests handelt es sich bei dynamischen Tests um das Testen von Software durch die Ausführung von Quellcodes. Hier wird als Voraussetzung ein lauffähiges Programm benötigt, das für den Text ausgeführt wird. Da dies in den unteren Testebene wie Komponenten- und Integrationstest noch nicht möglich ist, muss das Testobjekt in einem Testrahmen eingebettet werden, um ein ausführbares Programm zu bekommen.

Das Testobjekt ist in diesem Testrahmen eingebettet, der sowohl die Teile des Programms simuliert, die das Testobjekt über definierte Schnittstellen aufrufen, als auch die Teile, die vom Testobjekt selbst aufgerufen werden, wobei die Teile, die das Testobjekt aufrufen, auch als Testtreiber bezeichnet werden. Sie versehen das Testobjekt mit Daten und simulieren die nachfolgenden Programmteile, die das Objekt später aufrufen werden. Diese Schnittstelle zwischen den Testtreibern und dem Testobjekt wird auch „Point of Control“ (PoC) genannt und die Programmteile, die das Testobjekt aufruft, werden Platzhalter oder auch Stellvertreter genannt. Die Ausgabe der Testergebnisse und der Vergleich von Soll- und Ist-Verhalten des Testobjekts erfolgt über den „Point of Observation“ (PoO) (Vgl. Spillner & Linz (2019), S. 109, 113). In Abbildung 2 ist der Testrahmen zu sehen.



QUELLE: Spillner & Linz (2019)

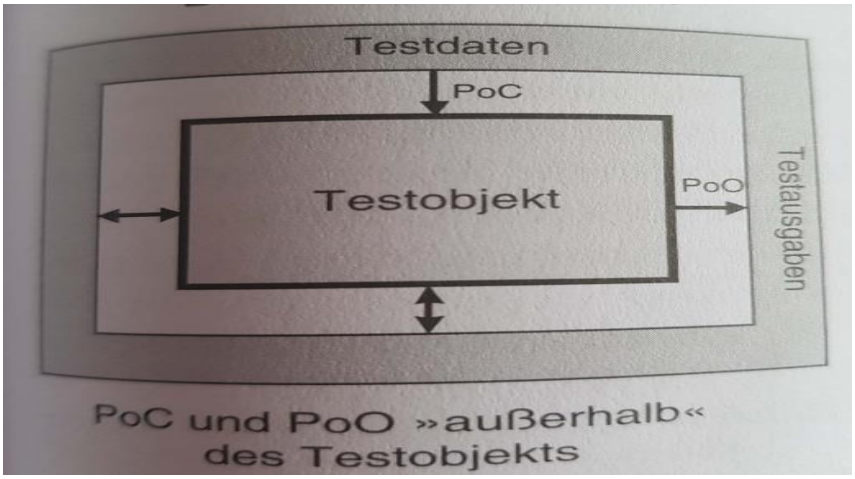
ABBILDUNG 2: TESTRAHMEN FÜR DYNAMISCHE TESTS

Zur Erstellung der Testfälle für das Testobjekt im Testrahmen existiert eine Vielzahl von unterschiedlichen Methoden, die eingesetzt werden können. Die Methoden können in zwei Gruppen gegliedert werden: Blackbox-Test und Whitebox-Test.

- **Blackbox-Tests**

  Das Testobjekt wird beim Blackbox-Test als schwarzer Kasten angesehen. Die interne Struktur des Testobjekts wird nicht erkannt und nicht zum Testen verwendet, sondern das Verhalten des Testobjekts wird von außen beobachtet, das heißt der PoO (Point of Observation) befindet sich außerhalb des Prüfobjekts. Außerdem ist eine Steuerung des Ablaufs des Testobjekts außer durch die Wahl der Eingabetestdaten nicht möglich, was auch bedeutet, dass sich der PoC (Point of Control) außerhalb des Testobjekts befindet. Deswegen werden die Testfälle aus der Spezifikation abgeleitet. Daher wird das Blackbox-Verfahren als funktionales oder spezifikationsbasiertes Testverfahren bezeichnet.

  Die Black-Box-Methode lässt sich gut zur Überprüfung der Funktionalität einsetzen, wobei eine falsch spezifizierte Funktionalität nicht erkannt werden kann. Die Tests werden auf der Grundlage falscher Spezifikationen durchgeführt und nur der Menschenverstand kann diese Fehler korrigieren. Darüber hinaus können auch nicht geforderte Funktionalitäten im Testobjekt vorhanden sein, welche über die Spezifikation hinausgehen. Diese lassen sich mit dem Blackbox-Test nicht erkennen und testen, sondern mit dem Whitebox-Test (Vgl. Spillner & Linz (2019), S. 114).



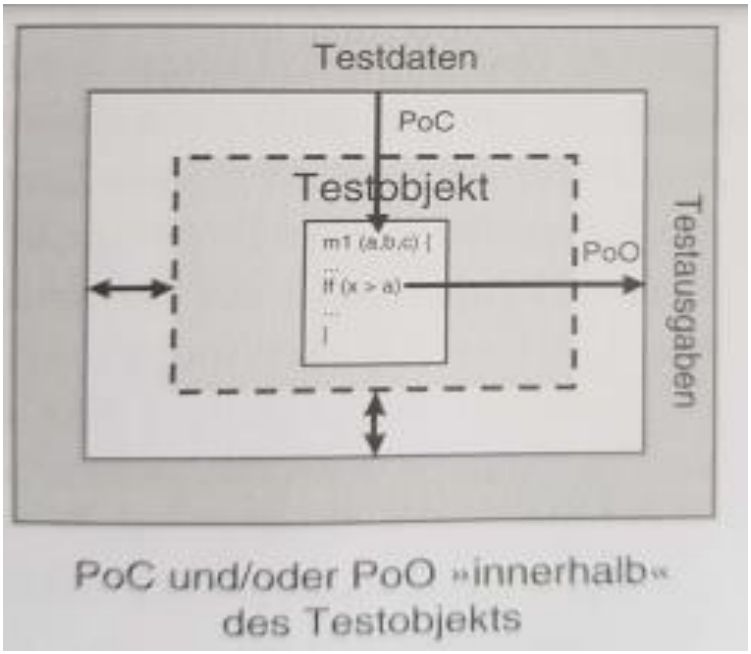
QUELLE: Spillner & Linz (2019)

ABBILDUNG 3: TESTRAHMEN BEI BLACKBOX-VERFAHREN

- **Whitebox-Tests**

Beim Whitebox-Test wird das Testobjekt als durchsichtiger Kasten betrachtet und er konzentriert sich im Unterschied zum Blackbox-Test an die Struktur des Programmcodes. Hier wird bei der Ausführung der Testfälle der innere Aufbau im Testobjekt behandelt, also der „Point of Observation“ und der „Point of Control“ liegen innerhalb des Testobjekts.

Grundsätzlich werden bei der Whitebox-Methode alle Teile des Quellecodes eines Testobjekts mindestens einmal ausgeführt. Deshalb wird das Whitebox-Verfahren auch strukturelles Testverfahren genannt. Ausgehend von der Programmlogik werden prozessorientierte Testfälle ermittelt und ausgeführt. Dabei wird die Spezifikation berücksichtigt, damit das Verhalten des Testobjekts als korrekt oder fehlerhaft bewertet wird (Vgl. Spillner & Linz (2019), S. 149). Die folgende Abbildung stellt das Whitebox-Verfahren dar.



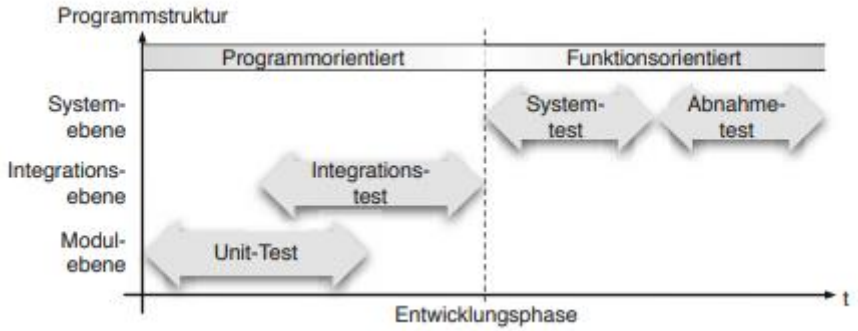
QUELLE: Spillner & Linz (2019)

ABBILDUNG 4: TESTRAHMEN BEI WHITEBOX-VERFAHREN

3.2 Beschreibung der einzelnen Teststufen

Das Testen von Software durchläuft in unterschiedlichen Entwicklungsphasen, die in den folgenden Abschnitten behandelt und analysiert werden. Es können die in der folgenden Abbildung dargestellten Teststufen unterschieden werden (Vgl. Hoffmann (2013) S.

159). Es werden deren Eigenschaften beschrieben, um die Unterschiede zwischen ihnen deutlich zu machen.



QUELLE: Hoffmann (2013)

ABBILDUNG 5: PRÜFEBENEN DES SOFTWARETESTS

3.2.1 Unit-Tests

Ein Unit-Test bzw. Komponenten-Test überprüft, ob die Methoden automatisch die erwarteten Werte zurückgeben und dann die Korrektheit einer oder mehrerer Anforderungen prüft. Diese Testart besitzt ein hohes Automatisierungspotenzial, denn die Komponentenschnittstelle bleibt unverändert, obwohl sich die Implementierung möglicherweise ändert. Daher kann den Test einmal programmiert und dann zu verschiedenen Zeiten durchgeführt werden. Allerdings gibt es spezifische Unit-Test-Frameworks zum Beispiel JUnit, NUnit, PHP Unit für unterschiedliche Programmiersprachen, die die nützlichen Bibliotheken zur Erstellung von Tests zur Verfügung stellen (Vgl. Dr. Axel Kalenborn, Thomas Will, Rouven Thimm, Jana Raab, Ronny Fregin (2006), S. 1 f.).

Bei der Durchführung von Unit-Tests wird für jede Klasse eine weitere Testklasse programmiert, mit der beliebig viele Elemente der Klasse aufgerufen und überprüft werden können. Darüber hinaus wird jeder Fehlerfall beispielweise jede ein- und ausgehende Exception einer Methode oder eines Objekts als einzelner Testfall ausgeführt. Das heißt, der Programmcode soll schrittweise auf Basis von Ein- und Ausgabeanforderungen entwickelt und getestet werden. Bevor die Tests durchgeführt werden, werden die Funktionalitäten programmiert und die dazugehörenden Testfälle beschrieben, um den internen Programmaufbau zu prüfen. Nachdem die Tests mit Erfolg durchgeführt sind, werden die Testfälle und der Code weiter in kleinen Teilen programmiert, bis der Gesamtcode programmiert ist (Vgl. Klaus Franz (2015), S. 107). Es gibt neben dem Unit-Test den Integrationstest, mit dem einzelne Programmmodule zu größeren Software-Komponenten zusammengesetzt werden.

3.2.2 Integrationstests

Der Integrationstest stellt fest, ob das Zusammenspiel zwischen den integrierten Komponenten fehlerfrei funktioniert, wobei dieser Fehler durch zwei verschiedene Programmierer verursacht werden kann, da sie die Schnittstellenbeschreibung unterschiedlich verstanden haben (Vgl. Stephan Grünfelder (2017), S. 123).

Es ist wichtig bei dem Integrationstest die zu integrierenden Komponenten zuerst zu prüfen, um Fehler zu verhindern. Zusätzlich werden Testfälle bei jeder Abhängigkeit zwischen zwei Komponenten für den Integrationstest definiert, diese zeigen die erfolgreiche Kommunikation der Komponenten über ihre Schnittstellen, d. h. Daten werden ausgetauscht und sind entsprechend den Anforderungen funktionsfähig. Es gibt unterschiedliche Strategien, wie Komponenten zu einem Gesamtsystem integriert werden können (Vgl. Hoffmann (2013), S. 164).

Die Bottom-Up-Integration

Die Bottom-Up-Integration beginnt mit den Komponenten, die nicht abhängig von anderen Modulen sind. Hier werden die integrierten Komponenten zunächst in der untersten Schicht eingesetzt und erst wenn alle Basiskomponenten integriert sind, werden die Komponenten der nächsten Schicht verarbeitet. Es werden Testtreiber benötigt, die die Tests für die einzelne Komponente ausführen. Dabei ist ein Testtreiber ein temporäreres Programmgerüst, der Aspekte der späteren Ausführung simuliert. Diese Treiber werden im Laufe der Entwicklung durch die richtigen aufrufenden Komponenten ersetzt, für die wiederum Testtreiber benötigt werden (Vgl. Hoffmann (2013) S. 164).

Top-Down-Integration

Im Top-Down-Integration hingegen beginnt die Integration mit den höchsten Komponenten und geht zu den Komponenten niedrigerer Ordnung über. Der Vorteil von Top-down-Integrationstests ist, dass schon zu Beginn ein Produkt mit einem groben Ablauf zu erkennen ist und die noch nicht fertiggestellten Komponenten durch Platzhalter (Programmcode, der anstelle eines anderen Programmcodes steht) ersetzt werden, wobei die vorhin getestete Komponente als Testreiber bezeichnet wird. Damit die neue Komponente getestet werden kann, sind nochmals Platzhalter erforderlich. was mit der Zeit sehr zeitaufwändig werden kann (Vgl. Hoffmann (2013) S. 164).

Backbone-Integration

Hier wird ein Testrahmen im Vorhinein erstellt, der alle Testreiber und Platzhalter enthält und in den die Komponenten in einer zeitlichen Reihenfolge integriert werden können (Vgl. Klaus Franz (2015), S. 107).

Ad-hoc-Integration

Bei dieser Integration werden kein Testtreiber und Platzhalter zur Verfügung gestellt, sondern die Komponenten in der beliebigen Reihenfolge integriert. Die Testtreiber und Platzhalter werden nur bei Bedarf programmiert. Der Vorteil dieser Methode liegt im Gegensatz zu der oben genannten Methode in der Zeitersparnis, da der Aufwand für die Erstellung von einem Testrahmen ausgespart wird. Das heißt, die Komponenten können so früh wie möglich integriert werden (Vgl. Klaus Franz (2015), S. 107).

Sobald alle Komponente des Softwaresystems erfolgreich integriert sind, wird der Systemtest durchgeführt.

3.2.3 Systemtests

Beim Systemtest wird das Komplettsystem getestet, um sicherzustellen, dass die spezifischen Anforderungen erfüllt wurden, wobei der interne Aufbau der Software nicht berücksichtig wird, im Gegensatz zu Unit- oder Integrationstests. Daher wird beim Systemtest das Blackbox-Verfahren verwendet. In dieser Stufe werden funktionale als auch nicht funktionale Anforderungen getestet. Die Kunden- oder Endnutzersicht sollte beim Testen berücksichtigt werden, was besser von internen Testern als von Entwicklern erledigt werden kann (Vgl. Torsten Cleff (2010) S. 29). Beim funktionalen Systemtest werden Test-Design-Tools benötigt, um die Tests anhand der Spezifikation zu identifizieren und zu planen. Ebenfalls werden Capture/Capture für die Testautomatisierung von wiederholten Aufgaben erforderlich.

Das Testen nicht-funktionaler Anforderungen ist oft weniger genau beschrieben und schwieriger zu quantifizieren Soweit die Anforderungen interpretierbar sind, sind jedoch auch die Ergebnisse der Tests nicht eindeutig. Deshalb werden die Ergebnisse der nicht-funktionalen Tests eher subjektiv bewertet. Beim nicht-funktionalen Systemtest werden auch Capture/Replay-Tools benötigt (Vgl. Witte (2019), S. 79 f.). Nachdem der Systemtest erfolgreich abgeschlossen ist, muss der Auftraggeber die Freigabe der Anwendung erteilen, d. h. das Gesamtsystem abnehmen.

3.2.4 Abnahmetests (Akzeptanztest)

Beim Abnahmetest wird sichergestellt, dass die mit dem Kunden vereinbarten Anforderungen erfüllt sind. In diesem Fall gilt das Entwicklungsprojekt als erfolgreich, wenn die Software die spezifizierten Vorgaben erfüllt und fehlerfrei ist. Als Basis für den Abnahmetest dient die Prüfung des vertraglichen Anforderungsdokuments, um wesentliche Mängel aufzuzeigen (Torsten Cleff (2010) S. 35). In diesem Test kommt nur der Blackbox-Test zur Anwendung, weil der Auftraggeber keinen Zugriff auf den Quellcode des Projekts besitzt. Es gibt mehreren Formen des Abnahmetests und im Folgenden werden diese unterschieden (Vgl. Spillner & Linz (2019), S. 65 ff.):

Test auf vertragliche Akzeptanz

Dabei gelten die im Vertrag festgelegten Abnahmekriterien als Referenz für die Erfüllung der Anforderungen. Die Erfüllung eventuell relevanter gesetzlicher Vorschriften, Normen, Sicherheitsbestimmung gehört auch dazu. Für den Abnahmetest reicht es aus, die entsprechenden Testfälle für die Abnahme gemäß dem Vertrag durchzuführen und dem Auftraggeber nachzuweisen, dass die Abnahmekriterien des Vertrages erfüllt sind. Hier ist es wichtig, dass der Kunde die Akzeptanztestfälle selbst erstellt oder einem Review unterzieht, denn der Softwarehersteller kann die vertraglich vereinbarten Akzeptanzkriterien missverstanden haben (Vgl. Spillner & Linz (2019), S. 66 f.)

Test der Benutzerakzeptanz

Der Test auf Benutzerakzeptanz ist ein weiterer Aspekt der Abnahme und ist dann immer zu empfehlen, wenn Kunde und Anwender des Systems verschiedenen Personengruppen angehören. Außerdem haben die unterschiedlichen Personengruppen in der Regel ganz verschiedene Erwartungen an das neue System. Falls eine Anwendergruppe das System ablehnt, kann die Abnahme nicht erfolgen, obwohl das System vollkommen in Ordnung ist. Wenn beim Abnahmetest Akzeptanzprobleme auftreten, ist es oft zu spät, Maßnahmen zu ergreifen. Um solche Probleme in Zukunft zu vermeiden, ist es wichtig, den Prototyp bereits in einem frühen Stadium von Vertretern der künftigen Nutzer bewerten zu lassen (Vgl. Spillner & Linz (2019), S. 66 f.).

Akzeptanz durch Systembetreiber

Bei diesem Akzeptanztest soll sichergestellt werden, dass sich das neue System aus Sicht der Systemadministratoren in die vorhandene IT-Landschaft einfügt. Es werden zum Beispiel die Backup-Routinen, der Wiederanlauf nach Systemabschaltung, die Benutzerverwaltung oder Aspekte der Datensicherheit untersucht (Vgl. Spillner & Linz (2019), S. 66 f.).

Feldtest (Alpha- und Beta- Tests)

Dieser Test ist eine besondere Form der Abnahmetests und wird hauptsächlich durchgeführt, wenn die Software in einer großen Anzahl unterschiedlicher Produktivumgebungen eingesetzt werden soll. Es ist unmöglich, alle Möglichkeiten und Kombinationen in einer Testumgebung zu reproduzieren. Zu diesem Zweck stellt der Hersteller einer ausgewählten Gruppe von Kunden stabile Vorabversionen zur Verfügung, die den Softwaremarkt gut repräsentieren. Diese Feldtests werden sehr häufig in der Computerspielindustrie verwendet, bei der jeder Endnutzer mit Sicherheit eine andere Hardware und Software hat. Dabei findet der Alpha-Test in der Testumgebung des Herstellers statt, während der Beta-Test in der Produktionsumgebung des Kunden durchgeführt wird (Vgl. Spillner & Linz (2019), S. 66 f.)

3.3 Grundlegende Testarten

Die in dieser Arbeit behandelten Testarten sind applikationsbezogene Testarten. Diese werden nachfolgend definiert und es werden Eigenschaften beschrieben, um die Unterschiede zwischen ihnen deutlich zu machen:

Funktionaler Test

Funktionaler Test eines Systems umfassen Tests, die die Funktionen bewerten, die das System ausführen soll. Die funktionalen Anforderungen können in Arbeitsprodukten wie Geschäftsanforderungen-Spezifikationen, User Stories, Use Cases oder funktionalen Spezifikationen beschrieben werden oder sie können undokumentiert sein. Außerdem sollte der funktionale Test auf allen Teststufen durchgeführt werden (z. B. Tests können für Komponenten auf einer Komponentenspezifikation basieren), wobei der Schwerpunkt auf jeder Ebene unterschiedlich ist. Der funktionale Test berücksichtigt das Verhalten der Software, so dass Blackbox-Verfahren verwendet werden können, um Testbedingungen und Testfälle für die Funktionalität der Komponente oder des Systems abzuleiten (Vgl. Klaus Olsen (chair), Meile Posthuma, Stephanie Ulrich (2018), S. 39).

Nicht funktionaler Test

Beim nicht funktionalen Testen eines Systems werden Eigenschaften von Systemen und Software wie die Benutzerfreundlichkeit, Leistung, Effizienz oder Sicherheit bewertet. Dabei wird geprüft, wie gut sich das System verhält. Im Gegensatz zu einem weit verbreiteten Fehler können und sollten nicht funktionale Tests auf allen Testebenen durchgeführt werden und zwar so früh wie möglich. Die spätere Entdeckung von nicht funktionalen Mängeln kann für den Erfolg eines Projekts äußerst gefährlich sein. Blackbox-

Verfahren können zur Ableitung von Testbedingungen und Testfällen für nicht funktionale Tests verwendet werden (Vgl. Klaus Olsen (chair), Meile Posthuma, Stephanie Ulrich (2018) S. 40).

Regressionstest

Während der Laufzeit eines neuen Softwareprodukts kann es in der Regel vorkommen, dass Änderungen oder Verbesserungen vorgenommen werden und dadurch in zuvor getesteten Programmen plötzlich Fehler verursachen. Um solche unerwünschten Fehler zu vermeiden, sollten mindestens einige der als Tests definierten Testfälle regelmäßig wiederholt werden. Es ist daher vorteilhaft, diese Regressionstests zu automatisieren, da sie unvermeidlich sehr repetitiv sind (Vgl. Lionel Pilorget (2012), S. 69). Regressionstests finden in allen Stufen des Entwicklungsprozesses statt und umfassen funktionale und nicht funktionale Tests. Um Testfälle für Regressionstest zu verwenden, müssen sie wiederholbar und ausreichend dokumentiert sein. Deshalb sind Testfälle für Regressionstests bevorzugte Kandidaten für die Testautomatisierung. Die Frage, die sich bei Regressionstests stellt ist, was aufgrund von Änderungen getestet werden muss. Es lassen sich folgende Möglichkeiten unterscheiden (Vgl. Spillner & Linz (2019), S.65 ff.):

1. Alle Tests wiederholen, die Fehlerwirkungen erzeugt haben, deren Ursache der (nun korrigierte) Defekt war (Fehlernachtest),

2. Test aller Programmteile, an denen Korrekturen oder Änderungen vorgenommen wurden (Test geänderte Funktionalität),

3. Test aller neu eingefügten Programmteile oder -sätze (Test der neuen Funktionalität),

4. das komplette System (vollständiger Regressionstest).

Der Fehlernachtest und der Test am Ort der Modifikation sind für die Sicherstellung der Funktionalität zu wenig, da in dem Softwaresystem scheinbar eine simple lokale Änderung unerwartete Auswirkungen und Seiteneffekte auf beliebige andere (auch weit entfernte) Systemteile haben kann. Ein vollständiger Regressionstest ist in der Regel immer zeitaufwändig und teuer. Daher werden Kriterien gesucht, um zu entscheiden, welche Testfälle notwendig sind und welche weggelassen werden können. Wie immer beim Testen ist dies eine Abwägung zwischen niedrigeren Kosten und höherem Risiko. Folgende Auswahlkriterien werden häufig angewendet (Vgl. Spillner & Linz (2019), S. 65 ff.):

- Wiederholen nur die Tests aus dem Testplan, denen eine hohe Priorität zugewiesen ist
- bei funktionalen Tests Verzicht auf gewisse Varianten (Sonderfälle),
- Einschränkung der Tests auf bestimmte Konfigurationen (z. B. nur Test der englischsprachigen Produktversion, nur Test auf einer bestimmten Betriebssystemversion),
- Einschränkung der Tests auf bestimmte Teilsysteme oder Teststufen.

Negativtest

Der Negativtest stellt fest, ob ein Programm bei Eingaben von falschen Daten den richtigen Verarbeitungsprozess durchführt. Dabei können z. B. inkorrekte oder unvollständige Werte eingegeben werden. Dabei wird der Test als erfolgreich bezeichnet, wenn das Programm mit einer Fehlermeldung reagiert. Beim Positivtest dagegen erfolgt der Test mit Eingaben von richtigen Daten (Vgl. Lionel Pilorget (2012), S. 69 f.).

End-to-End-Test

Zusätzlich zur Grundfunktionalität der Software muss auch die erweiterte Funktionalität der Software durch zusätzliche Tests geprüft werden. End-to-End-Tests beschreiben die Durchführung und Prüfung des ganzen Prozesses. Das heißt, Daten werden an einer Seite in das System eingegeben und auf der anderen Seite wird auf das richtige Ergebnis gewartet. Durch diesen Prozess wird die Funktionalität der Kommunikation zwischen den verschiedenen Anwendungen und die Übertragung von Daten verfolgt. Mit End-to-End-Tests kann sichergestellt werden, dass die definierten Prozesse befolgt wurden und die richtigen Informationen an die richtige Stelle gelangen (Vgl. Lionel Pilorget (2012), S. 69 f.)

4 Fehlermanagement

Fehler sollten beim Testen nicht als Misserfolg betrachtet werden, sondern als notwendige Investition auf dem Weg zu besseren Lösungen. Fehler, die beim Testen von Software gefunden werden, müssen erfasst, beschrieben, teilweise mit verschiedenen Projektmitgliedern diskutiert, korrigiert und erneut getestet werden. Bei diesem Prozess geht es darum, Fehler vollständig zu korrigieren sowie Kosten und Bearbeitungszeit für die Fehlerkorrektur so gering wie möglich zu halten (Vgl. Witte (2019), S. 94).

4.1 Fehlerarten

Funktionale Fehler

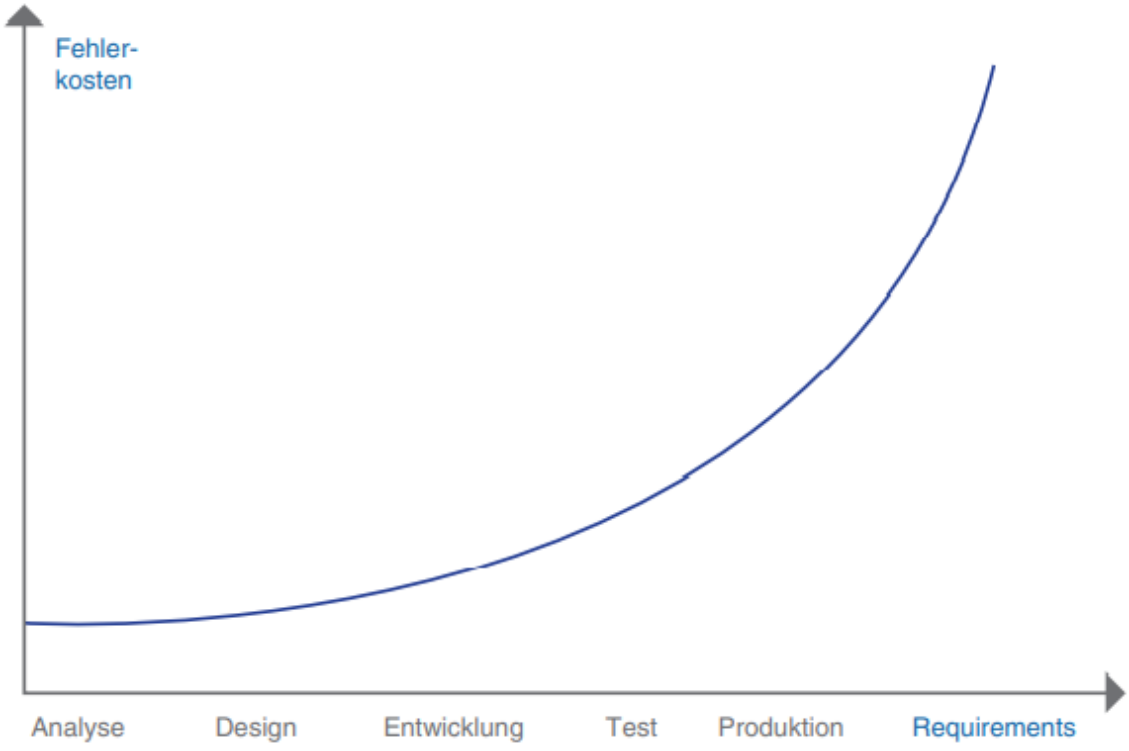
Funktionale Tests überprüfen den funktionalen Prozess, wie er vom Benutzer betrachtet wird. Er prüft beispielsweise, ob ein Anmelde-Button vorhanden ist und ob er wie erwartet funktioniert. Um die Tests überschaubar zu halten, wird in der Regel nur geprüft, ob die entsprechenden Templates vorhanden sind und funktionieren sie wie erwartet für die Benutzer mit unterschiedlichen Rollen und Rechten. Die während dieses Prozess auftretenden Fehler werden funktionale Fehler genannt (Vgl. Witte (2019), S. 94).

Performancefehler

Früher war wurde so vorgegangen, dass man sich beim Testen einer Anwendung zuerst auf die Funktionalität und dann auf die Performance konzentrieren sollte. Jedoch besteht heutzutage eine zunehmende Tendenz zur Durchführung von Performancetests in den Anfangsphasen des Projekts. Es kann auf eine schlechte Systemarchitektur verweisen, wenn die Antwort- oder Zugriffszeiten unterschiedlich sind und Fehler, die an dieser Stelle auftreten, in einer früheren Projektphase korrigiert werden. Denn wenn man am Ende zwar eine funktionierende, aber leider viel zu langsame Software hat, kann dies Anpassungen in der Dateistruktur oder bei den Datenbankzugriffen bedeuten, sodass der ganze Test neu aufgesetzt und die Entwicklung nachgearbeitet werden muss (Vgl. Witte (2019), S. 94).

4.2 Dynamik der Fehlerkosten

Bezüglich der Kosten für die Fehlererkennung in den einzelnen Projektphasen wird auch von der „1:10:100:1000-Regel“ oder „Zehnerregel“ gesprochen. Diese Regel besagt, dass, wenn ein Fehler länger in den letzten Phasen eines Produkts oder sogar bis zum Kunden unbemerkt bleibt, dann werden sich die Kosten für die Behebung dieses Fehlers erhöhen (Vgl. Witte (2019) S. 99 f.). Anhand der folgenden „Barry-Boehm-Kurve“ wird die Dynamik der Fehlerkosten deutlich dargestellt.



QUELLE: Witte (2019)

ABBILDUNG 6: ZEITLICHE ABHÄNGIGKEIT DER KOSTEN FÜR ÄNDERUNGEN NACH BOEHM

Faktor 1: Bei Entdeckung des Fehlers im Anforderungsmanagement

Da im Requiments Engineering auf vollständige, nicht widersprüchliche, klar verständliche und testbare Anforderungen geachtet wird, wird es in allen nachfolgenden Schritten gespart und es entstehen nur Kosten für eine Arbeitseinheit. Häufige Ursache für Mehrkosten und Verzögerungen ist, wenn die Kunden nicht ausreichend einbezogen werden und die Anforderungen unvollständig oder unrealistisch sind (Vgl. Witte (2019) S. 97 f.).

Faktor 10: Bei Entdeckung des Fehlers durch den Programmierer selbst

Wenn der Programmierer selbst einen Fehler bei der Entwicklung eines Tests bemerkt, kann er es schnell korrigieren. Der zusätzliche Aufwand beschränkt sich auf die Fehlerbehebung selbst. Dabei muss ein Teil des Programmcodes gelöscht werden, die Anforderungen müssen angepasst werden, und es gibt Rückgaben an die Systemarchitektur und Anforderungsanalyse. Dies verursacht Kosten in Höhe von 10 Arbeitseinheiten. Oft kommt das Softwaredesign nicht rechtzeitig hinterher. Die Softwareentwicklung hat bereits fertige Programmkomponenten geliefert. Die Software-Architektur definiert aber erst nachträglich allgemeine Anforderungen und Datenmodelle zu einem viel zu späten Zeitpunkt. In dieser Situation gibt es eine Unterbrechung in der gesamten Prozesskette. Werden zu einem späteren Zeitpunkt Änderungen an der Softwarearchitektur vorgenommen, müssen die Entwicklung und die Tests erneut angepasst werden. In diesen Fällen mangelt es oft an Feedback zwischen den Beteiligten. Es kommt häufig vor, dass es bereits bestehende Software gibt, die einen großen Teil der Kundenanforderungen abdecken kann. Deswegen fängt man nicht bei „Null“ an, wie es vorgeschrieben ist. Dies ist auch eine der Hauptursachen für Projektverzögerungen und Budgetüberschreitungen. Schließlich ist es besser dem Endkunden ein fertiges Softwareprodukt zu präsentieren als ein einfaches universelles Datenmodell (Vgl. Witte (2019) S. 99 f.).

Faktor 100: Entdeckung des Fehlers durch einen Tester

Wird der Fehler durch den Tester festgestellt, wenn der neue Entwicklungsstand bereits auf einem Testsystem läuft, erhöht sich der Aufwand. Der Tester sollte die Fehler dokumentieren, eine entsprechende Aufgabe in der Aufgabeverwaltung/im Bugtracking-System erstellen bzw. wiedereröffnen und den Entwickler darauf aufmerksam machen, diese Fehler zu beheben. Der Status des Test-Servers wird erneut aktualisiert und es wird ein weiterer Test auf dem Testserver durchgeführt, bevor der neue Stand live übertragen werden kann. Dadurch können Kosten für einen zusätzlichen Testzyklus anfallen. Dies kann auch zu einer Verschiebung des Einführungsdatums führen, was nicht nur den Produktmanager verärgert. Dabei entstehen dann die Kosten von 100 Arbeitseinheiten. Die Höhe dieser Kosten hängt von der Testebene ab. Wird der Fehler beim Integrationstest gefunden, ist die Behebung an dieser Stelle auf jeden Fall billiger als beim Abnahmetest (Vgl. Witte (2019) S. 99 f.).

Faktor 1000: Bei Entdeckung im Live-Betrieb

Es ist besonders ärgerlich, den Fehler erst während des Live-Betriebs zu entdecken, da nicht nur Kosten für die aufwändige Fehlerbehebung anfallen, sondern muss nun auch zusätzlich zur Behebung dieses Fehlers der Stand auf dem Live-System aktualisiert werden. Falls die Software bereits im Einsatz ist, kommen für die extra durchzuführenden Rollout-Maßnahmen zusätzliche Kosten hinzu. Dadurch schadet ein solcher Fehler dem Image der Software und des gesamten Unternehmens, denn die Wirksamkeit der gesamten Qualitätssicherung wird in Frage gestellt. Häufig werden in der letzten Sekunde erhebliche Anstrengungen unternommen, darunter Überstunden und Wochenend-Einsätze, um Kosten für Vertragsstrafen, Sachfolgekosten und Vermögensschäden zu vermeiden. Es entstehen Kosten von 1000 Arbeitseinheiten (Vgl. Witte (2019) S. 100).

5 Kategorien von Testwerkzeugen

Da in dieser Arbeit nicht nur über die Grundlagen der Softwaretests gesprochen wird, sondern es sollen auch später acht Testwerkzeuge verglichen werden, wird es in den nächsten Kapiteln eine Übersicht über die Testtools geben.

5.1 Tools für statische Tests

Statische Test können an Quellcodes oder Spezifikationen durchgeführt werden, um Fehler und Abweichungen durch Werkzeuge zum statischen Test schon früh zu erkennen. Die Testtools für statische Tests werden im folgenden Abschnitt genauer betrachtet (Vgl. Spillner & Linz (2019), S. 214 f.).

- **Werkzeuge zur Reviewunterstützung**

Der Fokus dieses Werkzeugs liegt auf dem Planen, Durchführen und dem Auswerten der Ergebnisse von Reviews. Dabei werden Informationen über geplante und durchgeführte Reviewmeetings, über Reviewteilnehmer, Reviewbefunde und -ergebnisse gesammelt und dann ausgewertet und verglichen. Dadurch können nicht nur Review-Aufwände besser eingeschätzt und geplant werden, sondern auch Schwachstellen im Entwicklungszyklus erkannt und gezielt behoben werden. Die Werkzeuge zur Review-Unterstützung sind besonders bei der Durchführung von großen, geografisch verteilten Projekten in mehrere Teams geeignet (Vgl. Spillner & Linz (2019), S. 214 f.).

- **Werkzeuge für statische Analysen**

Statische Analysatoren ermöglichen, wie im vorigen Kapitel erläutert wurde, die Analyse der Quellcodes und Metriken. Weiterhin lassen sich Fehler und Unstimmigkeiten z. B. bezüglich der Daten- und Kontrollflüsse früh durch diese im Quellcode entdecken. Dieses Tool sammelt alle Schwachstellen auf und somit können Umfang und Schärfe der Analyse eingestellt werden. Am Anfang sollten die Analysefilter schwach eingestellt sein und später erhöht werden (Vgl. Spillner & Linz (2019), S. 214 f.).

- **Model Checker**

Dieses Werkzeug liest die Struktur eines Modells ein und überprüft die verschiedenen statischen Eigenschaften dieser Modelle. Zusätzlich können sie fehlende Zustände, fehlende Zustandsübergänge und andere Unstimmigkeit im getesteten Modell erkennen. Erweiterungen dieses Tools sind spezifikationsbasierte Testgeneratoren, die Testdaten und zugehörige Sollwerte aus einer Spezifikation ableiten. Diese Tools sind für Entwickler interessant (Vgl. Spillner & Linz (2019), S. 214 f.).

.

5.2 Tools für dynamische Tests

Tools für dynamische Tests dienen zur automatisierten Durchführung von dynamischen Testfällen. Dadurch werden die Tester von mühsamen mechanischen Arbeiten erleichtert. Dabei werden dem Testobjekt Eingabedaten zur Verfügung gestellt, dann wird die Reaktion des Testobjekts aufgezeichnet und schließlich wird der Testverlauf dokumentiert (Vgl. Torsten Cleff (2010) S. 267 ff.). Die Werkzeuge für dynamische Tests werden im folgenden Abschnitt näher erklärt.

- **Capture Replay Tools**

  Diese Tools sind Testausführungswerkzeuge, welche die Ausführung von Testfällen durch automatische Testskripte ermöglichen. Hier wird das Testskript nicht vom Tester programmiert, sondern vom Testwerkzeug automatisch erzeugt. Später können diese Tests immer wieder mittels eines Skripts mit dem Tool automatisch ausgeführt werden. Darüber hinaus stellen die Hersteller von den Testtools viele Programmiersprachen zur Verfügung, die vom Testwerkzeug interpretiert werden können (Vgl. Torsten Cleff (2010) S. 267 ff.).

- **Testrahmen**

  Ein Testrahmen wird, wie bereits im vorherigen Kapitel erwähnt, während des Komponententests benötigt. Er wird für die Testausführung eingesetzt, damit Systemkomponenten ausgeführt und entsprechend analysiert werden. Der Testrahmen besteht aus Testtreiber, Platzhalter und Entwicklungsumgebung IDE. Die Verwendung von Testrahmen kann auch im Systemtest in Frage kommen, da ein integriertes System während des Systemtests geeignete Platzhalter, eine Testumgebung und Testtreiber benötigen kann (Vgl. Torsten Cleff (2010) S. 267 ff.).

- **Vergleichswerkzeuge (Komparatoren)**

  Komparatoren stellen sicher, dass Unterschiede zwischen erwartetem und tatsächlichem Ergebnis automatisch verglichen werden. Dabei wird mit Dateien und Datenbankinhalten gearbeitet und Unstimmigkeiten zwischen Dateien mit Soll- und Ist-Daten verglichen. Capture/Replay Tools (Testroboter) unterstützen solche Vergleichsfunktionalitäten (Vgl. Torsten Cleff (2010) S. 267 ff.).

5.3 Fazit zum Softwaretest

Zum Schluss werden noch die sieben Grundprinzipien des Softwaretestens vorgestellt, welche sich in den letzten 40 Jahren im Softwaretest entwickelt haben. Diese können als allgemeiner Leitfaden für das Testen angesehen werden.

1. **Grundsatz: Testen zeigt die Anwesenheit von Fehlern**

   Testen hat als Motivation das Finden von Fehlern in einem Programm oder Codestück. Unter Testen darf nicht eine Tätigkeit verstanden werden, welche nur die richtige Funktion einer Software überprüft. Durch ausreichendes Testen wird die Wahrscheinlichkeit verringert, dass noch unentdeckte Fehlerzustände im Testobjekt vorhanden sind.

2. **Grundsatz: Vollständiges Testen ist nicht möglich**

   Das vollständige Testen eines Testobjektes ist nicht möglich. Es gibt viel zu viele Möglichkeiten, Randbedingungen und Situationen in denen ein System geraten kann und diese kann ein Tester nicht alle beachten.

3. **Grundsatz: Mit dem Testen frühzeitig beginnen**

   Testaktivitäten sollen im Softwarelebenszyklus so früh wie möglich begonnen werden, da wie schon oben erwähnt wurde, die Kosten für das Finden und Beheben von Fehlern am Anfang geringer sind.

4. **Grundsatz: Häufung von Fehlern**

   Hier wird erläutert, dass an einer Stelle eines Testobjekts, an der ein Fehler gefunden wurde, mit hoher Wahrscheinlichkeit noch weitere Fehler auftreten und sich an der gleichen Stelle häufen kann.

5. **Grundsatz: Zunehmende Testresistenz (Pesticide Paradox)**

   Dieser Grundsatz bedeutet, dass Tests, die immer wieder wiederholt werden, mit weniger Wahrscheinlichkeit neue Fehler aufdecken. So wie Schädlinge gegen Pflanzenschutzmittel resistent werden, werden auch Fehler resistent gegenüber Tests. Deshalb sollten die Testfälle regelmäßig auf ihre Effektivität und Angemessenheit überprüft und gegebenenfalls durch neue oder geänderte Testfälle ersetzt werden.

6. Grundsatz: Testen ist abhängig vom Umfeld

Je nach Einsatzgebiet und Umfeld des zu prüfenden Systems ist das Testen immer an die gegebenen Voraussetzungen anzupassen, z. B. sicherheitskritische Systeme verlangen andere Prüfungen als kommerzielle Systeme.

7. Grundsatz: Trugschluss: Keine Fehler bedeutet ein brauchbares System

Dieses Prinzip besagt: Keine Fehler gefunden, bedeutet nicht automatisch, dass ein brauchbares Programm vorliegt. Wenn die Vorstellungen und Erwartungen nicht die Kundenwünsche erfüllt, ist dies genauso schlimm wie ein völlig fehlerhaftes System, welche die spezifizierten Anforderungen erfüllen würde (Vgl. Spillner & Linz (2019), S.37 f.).

.

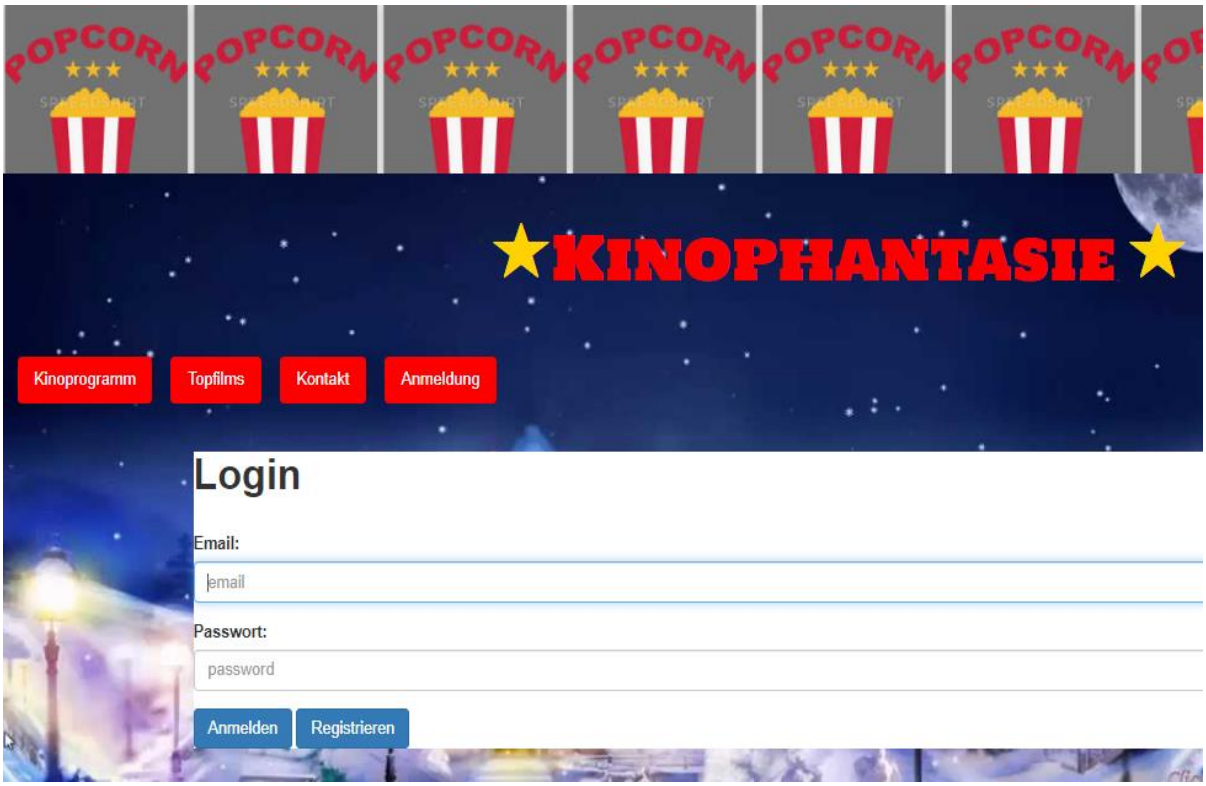
6 Vergleich Testwerkzeuge

In den nächsten Kapiteln werden einige ausgesuchte Testtools im Detail beschrieben. Dabei werden die Tools für Webanwendungen und die Tools für mobile Applikationen anhand unterschiedlicher Kriterien verglichen. Diese Tools sind auch im Internet verfügbar und dokumentiert. Es wird analysiert, welche Aufgabebereiche diese Tools abdecken können und wo die Unterschiede und Gemeinsamkeiten der Tools liegen. Abschließend werden auch die Probleme und Grenzen diesen Tools erläutert.

6.1 Auswahl

Für das Erzeugen automatisierter Tests können spezielle Tools verwendet werden. Darunter TestCafe, Selenium Webdriver, Selenium IDE, QF-Test für browserbasierte Tests auf Desktopumgebungen sowie Appium, Ranorex, TestComplete, Unified Functional Testing (UFT) für mobile Browser und Applikationen.

Es werden genau fünf Tools für das Testen von Webanwendungen verwendet, nämlich TestCafe, Selenium Webdriver, Selenium IDE, Appium, QF-Test. Diese Tools sind im Internet frei verfügbar und dokumentiert. Nicht alle Werkzeuge sind kostenlos erhältlich, deswegen wird bei Ranorex, TestComplete, und UFT nur auf die Beschreibung eingegangen. Mittels dieser Tools wird das Login von einer Kino Webseite getestet, die während des Studiums im Rahmen der Module Webtechnologie entwickelt wurde. Der folgende Screenshot stellt die zu testende Anwendung dar.



QUELLE: Eigene Darstellung

ABBILDUNG 7: KINO WEBSEITE

Da eine Webanwendung zum Testen zur Verfügung steht, sollte nun ein Testfall geschrieben werden. Dabei werden die Aktionen erläutert, die für die Durchführung des Tests benötigt werden. Dieser Testfall wird für die gesamte Arbeit verwendet.

- Öffne die Webseite http://localhost/erika/admin/anmeldung.php!
- Schreibe die Anmeldedaten für das Eingabe-Login!
- Schreibe das Passwort für die Passworteingabe!
- Klicke auf den Anmelde-Button!

6.1.1 TestCafe

TestCafe ist ein eigenständiges, kostenloses End-to-End Test-Framework, das die Durchführung von Tests in JavaScript, TypeScript oder CoffeeScript ermöglicht. Das Framework wurde von DevExpress entwickelt und ist als freie Software veröffentlicht. Die Durchführung von Tests im TestCafe erfolgt in folgenden Schritten:

1. Installation von TestCafe

   Bei der Installation muss sichergestellt werden, dass Node.js und npm auf dem Computer installiert sind und folgender Befehl ausgeführt wird: „npminstall -g testcafe“. Auf die Installation von Node.js und npm wird hier nicht näher eingegangen.

2. TestCafe Funktion erstellen

   Um einen Test zu erstellen, muss eine neue „.js“- oder „.ts“-Datei erstellt werden. Diese Datei muss eine besondere Struktur haben und die Tests müssen in Fixtures organisiert sein. In der folgenden Abbildung wird die TestCafe-Instanz mit der Funktion „createTestCafe“ erzeugt und eine HTTPS-Verbindung mit dem TestCafe-Server hergestellt (Vgl. Dmytro Shpakovskyi (2020), S. 10).

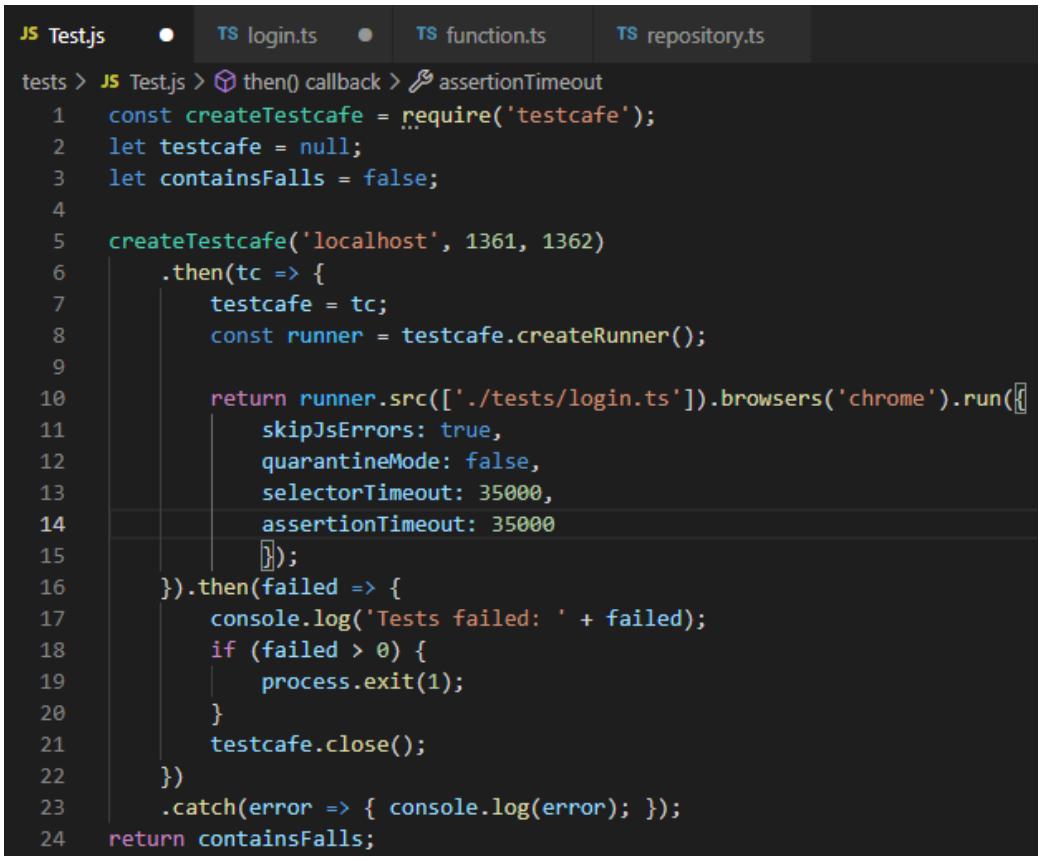


ABBILDUNG 8: TESTCAFE SCRIPT

3. Tests schreiben

Um die Organisation der Testcode-Struktur besser zu verstehen, lässt sich diese in mehrere Teile unterteilen: Fixtures, Tests, Start-Webseite.

- **Fixtures**

TestCafe-Tests werden in der Regel in Testsuiten, sogenannte Fixtures, gruppiert. Außerdem sollten jede JavaScript-, TypeScript- oder CoffeeScript-Datei mit TestCafe-Tests eine oder mehrere Fixtures beinhalten. Testfixtures können mit der Funktion „fixture“ deklariert werden, die nur ein Argument fixtureName enthaltet, das eine Zeichenkette für den Namen des Fixtures (Satz von Tests) ist. Ein Fixture ist im Grunde ein Wrapper, der den Beginn einer Reihe von Tests anzeigt. Diese Tests sind wie folgt aufgebaut:

- **Tests**

Tests werden normalerweise direkt nach der Fixture-Deklaration geschrieben. Um einen Test zu erstellen, wird die test-Funktion aufgerufen, die zwei Argumente akzeptiert:

testName: Eine Zeichenkette für den Namen des Tests.

Funktion: Eine asynchrone Funktion, die den Testcode enthält und ein Argument „t“, das ein Objekt für den Test-Controller ist. Der Test-Controller wird für den Zugriff auf alle Aktionen und „Assertions“ benutzt.

Dadurch, dass die Tests von TestCafe auf der Serverseite ausgeführt werden, können beliebig weitere Pakete oder Module verwendet werden. Außerdem kann innerhalb des Tests folgendes ausgeführt werden:

Testaktionen werden zur Interaktion mit der getesteten Webseite genutzt.

Selektoren und Client-Funktionen werden verwendet, um Informationen über den Zustand von Seitenelementen zu erhalten oder andere Daten von der Client-Seite zu erhalten.

Mit Hilfe von **Assertions** wird überprüft, ob die Seitenelemente die erwarteten Parameter haben.

- **Webseite starten**

Das Starten der Webseite kann mit der Funktion „fixture.page“ festgelegt werden, die als Startpunkt für alle Tests in einer Fixture gilt. Es akzeptiert nur ein Argument „url“, das eine Zeichenkette für die URL der Webseite ist, auf der alle Tests in einer Fixture beginnen:

Test erstellen

TestCafe unterstützt mit JavaScript, TypeScript oder CoffeeScript geschriebene Tests mit allen modernen Features, wie Pfeilfunktionen und „async/await“. Zuerst wird mit der Einbindung des TestCafe-Moduls begonnen, dann wird ein Fixture mit der Funktion „fixture“ deklariert, schließlich wird der erste Test mit der Testfunktion „Test“ durchgeführt (Dmytro Shpakovskyi (2020), S. 63).

Da die Kino-Webseite als Testprojekt ausgewählt wurde, wird diese URL (http://localhost/erika/admin/anmeldung.php) als Startseite für alle Tests in der „login“ Fixture mit der Seitenfunktion gesetzt.

```
tests > TS login.ts > ...
1  import { Selector } from "testcafe";
2  import * as $ from "../repository";
3  import { login } from "./function";
4
5  fixture("login")
6      .page($.url)
7      .beforeEach(async t => {
8          console.log("start");
9      });
10
11 test("login", async t => {
12 //connecting user to website
13     await login($.user, $.password)
14 });
```

QUELLE: Eigene Darstellung

ABBILDUNG 9: TESTCAFE SCRIPT

Aktionen ausführen

Mit dem Befehl „await t .typetext“ und „await t .click“ werden einige Aktionen auf der Seite durchgeführt. Alle Testaktionen sollten als „async“ Funktionen des Test-Controller-Objekts t verwendet werden. Ein Test-Controller-Objekt stellt die Methoden der Test-API zur Verfügung. Der Test-Controller wird an jede Funktion übergeben, die serverseitige Testcodes ausführen kann (wie „test“, „beforeEach“ oder „afterEach“). Der Test-Controller wird verwendet, um Testaktionen aufzurufen, Browserdialoge zu behandeln, die Wait-Funktion zu verwenden oder Assertions auszuführen (Vgl. DevExpress-TestCafe Dokumentation (2012-2021)).

```
tests > TS function.ts > login
1  import { Selector, t } from "testcafe";
2  import * as $ from "../repository";
3
4  export async function login(username, userpass) {
5
6      await t.typeText($.userInput, username)
7      await t.typeText($.passwortInput, userpass)
8      await t.click($.anmeldenButton);
9
10 }
```

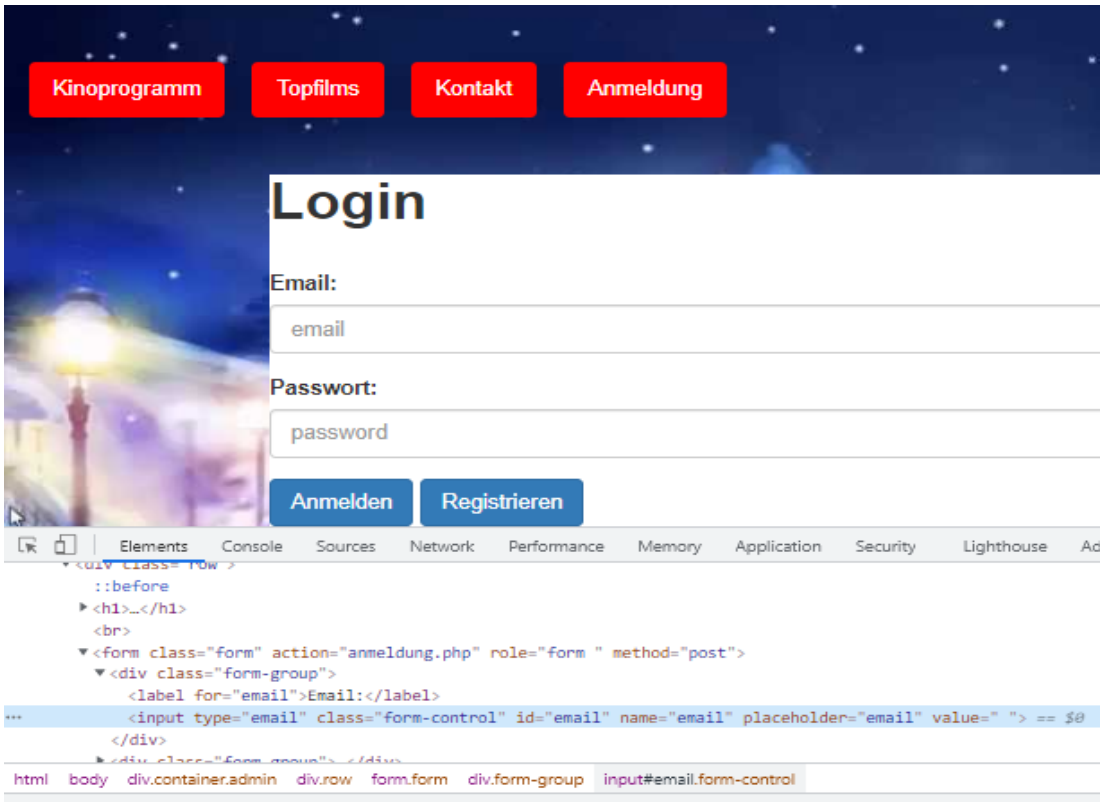
QUELLE: Eigene Darstellung

ABBILDUNG 10: TESTCAFE SCRIPT

Elemente untersuchen in Google Chrome

Google Chrome bietet eine eingebaute Funktion zur Überprüfung von Elementen. Diese erfolgt durch die Bewegung der Maus über ein gewünschtes Element auf der Seite. Dann wird mit der rechten Maustaste geklickt, um das Popup-Menü zu öffnen, und die Option "Element untersuchen" auszuwählen. Dies öffnet schließlich die Entwicklertools im Browser, wie der folgende Screenshot zeigt. Es können auch XPath- und CSS-Selektoren in den Google Chrome Entwicklung-Tools untersucht werden, indem auf Strg + F (auf Mac: Befehl + F) in der Registerkarte Elemente gedrückt wird. Dann erscheint ein Suchfeld. Danach wird einfach XPath oder CSS Selector eingegeben und passende Elemente werden in der Baumstruktur hervorgehoben.

Browser-Entwickler-Tools sind bei der Testskript-Entwicklung sehr nützlich. Diese Werkzeuge helfen, die Locator-Details für die Elemente zu finden, mit denen sie als Teil des Tests interagieren müssen. Diese Werkzeuge analysieren den Code einer Seite und zeigen die Informationen in einem hierarchischen Baum (Vgl. Gundecha & Avasarala (2018), S. 26)



QUELLE: Eigene Darstellung

ABBILDUNG 11: ELEMENTE UNTERSUCHEN

```ts
TS repository.ts > ...
1   import { Selector} from "testcafe";
2   // Username and Password
3   export const user = "erika@yahoo.fr"
4   export const password = "123"
5   export const url = "http://localhost/erika/admin/anmeldung.php"
6
7   export const userInput = Selector("input").withAttribute("id","email")
8   export const passwortInput = Selector("input").withAttribute("id","password")
9   export const anmeldenButton = Selector("button").withAttribute("class","btn btn-primary")
```

QUELLE: Eigene Darstellung

ABBILDUNG 12: TESTCAFE REPOSITORY

4. Test ausführen

Der Test kann einfach von einer Kommando-Shell aus gestartet werden, indem ein einziger Befehl mit dem Zielbrowser und Dateipfad ausgeführt wird. TestCafe startet automatisch die ausgewählte Browser-Instanz und beginnt mit der Ausführung des Tests. Die Kommandoausgabe sieht dann bei erfolgreicher Testdurchführung so aus:

```
C:\Users\erikanguepi\Desktop\Praktikum\Projektsoftwaretesting>node tests/Test.js
 Running tests in:
 - Chrome 91.0.4472.164 / Windows 10

 login
start
 √ login


 1 passed (21s)
Tests failed: 0
```

QUELLE: Eigene Darstellung

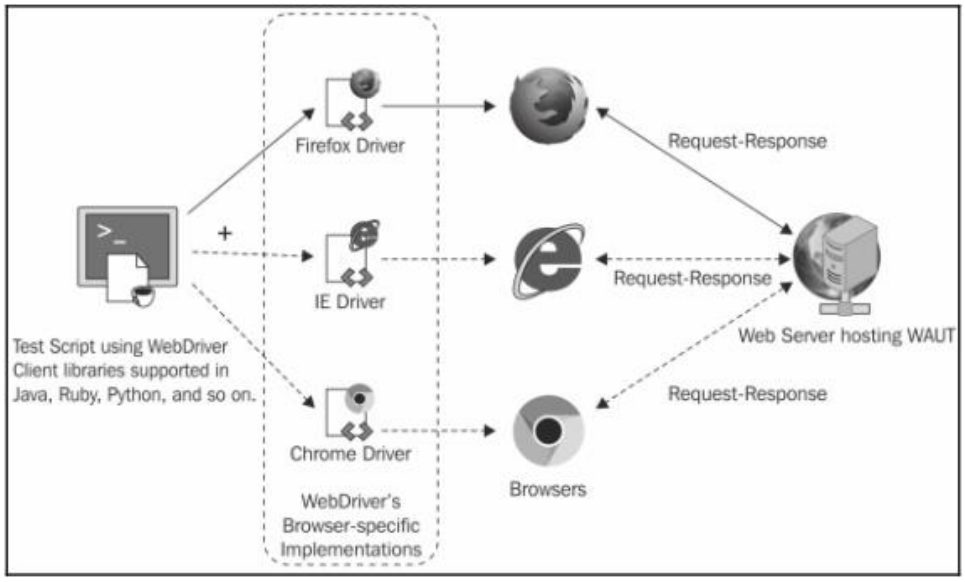
ABBILDUNG 13: AUSGABE NACH DER TESTDURCHFÜHRUNG

6.1.2 Selenium Webdriver

Selenium Webdriver ist ein Framework für automatisierte Tests von Webanwendungen und wurde von einem Programmierer-Team der Firma ThoughtWorks entwickelt. Es simuliert Nutzeraktionen und steuert somit den Browser fern. Selenium WebDriver nimmt Befehle über das JSON-Wire Protokoll, auch Client-API genannt, und sendet sie an einen Browser, der von der spezifischen Driverklasse (wie ChromeDriver, FirefoxDriver oder

IEDriver) gestartet wird. Dies wird durch einen browserspezifischen Browsertreiber implementiert, der mit der folgenden Reihenfolge arbeitet (Gundecha & Avasarala (2018), S. 8 f.):

- Der Driver hört auf die Befehle von Selenium.
- Er setzt diese Befehle in die native API des Browsers um.
- Er nimmt das Ergebnis der nativen Befehle und sendet zurück an Selenium.



QUELLE: Gundecha & Avasarala (2018)

ABBILDUNG 14: TESTDURCHFÜHRUNG MIT SELENIUM WEBDRIVER

Mittels Selenium Webdriver kann eine robuste, browserbasierte Regressionstestautomatisierung erstellt werden, Skripte über viele Browser und Plattformen skaliert und verteilt werden und Skripte in der gewünschten Programmiersprache geschrieben werden. Zusätzlich stellt Selenium WebDriver eine Sammlung von sprachspezifischen Client-Bibliotheken zur Verfügung, um den Browser zu steuern (Vgl. Gundecha & Avasarala (2018), S. 8 f.).

In diesem Abschnitt wird wieder die Kino Webseite mittels Selenium Webdriver getestet.

- **Projekt in Eclipse mit Maven unter Verwendung von Java einrichten**

  Eine integrierte Entwicklungsumgebung (IDE) oder ein Code-Editor werden benötigt, um ein neues Java-Projekt zu erstellen. Eclipse und Maven werden verwendet, um Selenium WebDriver-Testframework von einem einzigen Fenster aus zu erstellen.

Ein wichtiger Vorteil der Verwendung von Maven ist, dass alle Selenium-Bibliotheksdateien und deren Abhängigkeiten durch Konfiguration der pom.xml-Datei erhalten werden können. Maven lädt beim Erstellen des Projekts automatisch die notwendigen Dateien aus dem Repository herunter. Auf die Konfiguration von Eclipse und Maven für die Selenium WebDriver Testentwicklung wird hier nicht näher eingegangen. Am Ende sieht das Ergebnis wie im folgenden Screenshot aus:

```
1  package firsttest.selenium.demo;
2
3  import java.util.concurrent.TimeUnit;
4
5  import org.openqa.selenium.By;
6  import org.openqa.selenium.WebDriver;
7  //import org.openqa.selenium.WebElement;
8  import org.openqa.selenium.chrome.ChromeDriver;
9  //import org.openqa.selenium.interactions.Actions;
10
```

QUELLE: Eigene Darstellung

ABBILDUNG 15: IMPORT PACKAGE

- **Browser festlegen**

Im unterstehenden Code wird den Pfad der ausführbaren Chromdriver-Datei gesetzt und eine Instanzvariable „WebDriver-driver“ deklariert, die später im Test verwendet wird, um den Browser zu starten und zur Website zu navigieren. Das Objekt „driver“ wird nun genutzt, um den Browser fernzusteuern. Meist macht es Sinn, im zweiten Schritt eine URL im Browser aufzurufen.

```
1  package firsttest.selenium.demo;
2
3  import java.util.concurrent.TimeUnit;
4
5  import org.openqa.selenium.By;
6  import org.openqa.selenium.WebDriver;
7  //import org.openqa.selenium.WebElement;
8  import org.openqa.selenium.chrome.ChromeDriver;
9  //import org.openqa.selenium.interactions.Actions;
10
11 public class demo {
12
13     public static void main(String[] args) {
14         // Connect Chrome with my programm
15         System.setProperty("webdriver.chrome.driver", "C:\\chromedriver.exe");
16         WebDriver driver = new ChromeDriver();
17
18         // implicit wait
19         driver.manage().timeouts().implicitlyWait(20, TimeUnit.SECONDS);
20         driver.get("http://localhost/erika/admin/anmeldung.php");
```

QUELLE: Eigene Darstellung

ABBILDUNG 16: WEBDRIVER

- **Elemente untersuchen in Google Chrome (siehe TestCafe)**

- **Die Methode „findElement“**

Um mehrere Elemente auf einer Webseite zu finden, die den gleichen Suchkriterien entsprechen, kann die „findElements()“ Methode verwendet werden. Sie gibt eine Liste von „WebElementen“ zurück, die für einen bestimmten Suchvorgang gefunden wurde. Die Methodendeklaration der Methode „findElements()“ lautet „java.util.List-findElements(By by)“.

Der Eingabeparameter ist der gleiche wie bei der Methode findElement(), nämlich eine Instanz der „By-Klasse“. Der Unterschied liegt im Rückgabetyp. Hier wird eine leere Liste zurückgegeben, wenn kein Element gefunden wird, und wenn mehrere WebElemente vorhanden sind, die den Suchvorgang erfüllen, werden sie alle in einer Liste an den Aufrufer zurückgegeben (Vgl. Gundecha & Avasarala (2018), S. 26).

- **Verwendung des „By-Lokalisierung“**

„By“ ist der Ortungsmechanismus, der an die „findElement()“ Methode übergeben wird, um das/die entsprechende(n) WebElement(e) auf einer Webseite zu holen. Es gibt acht verschiedene Möglichkeiten zur Identifizierung eines HTML-Elements auf einer Web-Seite. Sie werden über „By.id“, „By.name“, „By.className“, „By.tag-Name“, „By.linkText“, „PartialLinkText“, „By.xPath“ und „By.css-Selektor“ gefunden (Vgl. Gundecha & Avasarala (2018), S. 33). Mit den dadurch ausgewählten Elementen kann im nächsten Schritt interagiert werden. Beispielsweise sind das Ausführen von Mausklicks sowie Tastatureingaben möglich. Ein vollständiger Test des Login-Formulars der Kino-Webseite sieht schließlich wie folgt aus:

```java
package firsttest.selenium.demo;

import java.util.concurrent.TimeUnit;

import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
//import org.openqa.selenium.WebElement;
import org.openqa.selenium.chrome.ChromeDriver;
//import org.openqa.selenium.interactions.Actions;

public class demo {

    public static void main(String[] args) {
        // Connect Chrome with my programm
        System.setProperty("webdriver.chrome.driver", "C:\\chromedriver.exe");
        WebDriver driver = new ChromeDriver();

        // implicit wait
        driver.manage().timeouts().implicitlyWait(20, TimeUnit.SECONDS);
        driver.get("http://localhost/erika/admin/anmeldung.php");

        // Find login element
        driver.findElement(By.id("email")).sendKeys("erika@yahoo.fr");
        driver.findElement(By.id("password")).sendKeys("123");
        driver.findElement(By.name("anmelden")).click();
```

QUELLE: Eigene Darstellung

ABBILDUNG 17: SELENIUM SCRIPT

- **Test ausführen**

Um den Test ausführen zu lassen, wird mit der rechten Maustaste in den Code-Editor geklickt und „Ausführen als“ geklickt. Dadurch wird ein neues Google Chrome-Browser-Fenster gestartet und die Website wird aufgerufen. Der Test validiert den Seitentitel, und das Browserfenster wird am Ende des Tests geschlossen.

6.1.3 Selenium-IDE

Selenium-IDE ist das Tool, mit dem Selenium-Testfälle entwickelt werden können. Es ist ein einfach zu bedienendes Tool und im Allgemeinen der effizienteste Weg, um Testfälle zu erstellen, welche dann wieder im Browser ausgeführt werden können (Vgl. Thoughtworks - Selenium Dokumentation (2021)). Selenium IDE kann verwendet werden, um:

- schnelle und einfache Skripte mit Aufzeichnung und Wiedergabe zu erzeugen,
- Skripte zur Unterstützung von automatisierungsgestützten explorativen Tests zu erstellen,
- die Ausführung sich wiederholender Aufgaben auf Webseite zu erstellen (Vgl. Gundecha & Avasarala (2018), S. 9).

Es wird hier nicht auf die Installation des Selenium IDE eingegangen. Nach der Installation kann Selenium IDE unter „Extras“ gestartet werden. Danach öffnet sich ein Fenster, wie es in der folgenden Abbildung zu erkennen ist. Ab diesem Punkt werden alle Aktionen des Benutzers aufgezeichnet, und die Tests können durchgeführt werden. Hier wird wieder das Login der Kino-Webseite mit Selenium IDE getestet. Dafür wird folgendes Szenario durchgeführt:

- Öffne die Webseite http://localhost/erika/admin/anmeldung.php!
- Schreibe die Anmeldedaten für das Eingabe-Login!
- Schreibe das Passwort für die Passworteingabe!
- Klicke auf den Anmelde-Button!
- Klicke bei der Selenium IDE auf den roten Knopf zum Beenden der Aufnahme!

Danach werden die einzelnen Schritte aufgezeichnet, wie sie auch in folgender Abbildung zu sehen sind. Es ist auch möglich, den von der IDE automatisch erzeugten Quellcode direkt zu modifizieren, wofür allerdings Programmierkenntnisse erforderlich sind. Auch die Programmiersprache, in dem die Testfälle gespeichert werden sollen, kann ausgewählt werden. Die Auswahlmöglichkeiten sind darunter HTML, JAVA, C#, Perl, Python, PHP, Ruby. Nach dem Aufzeichnen sieht das Ergebnis wie in der folgenden Abbildung aus:

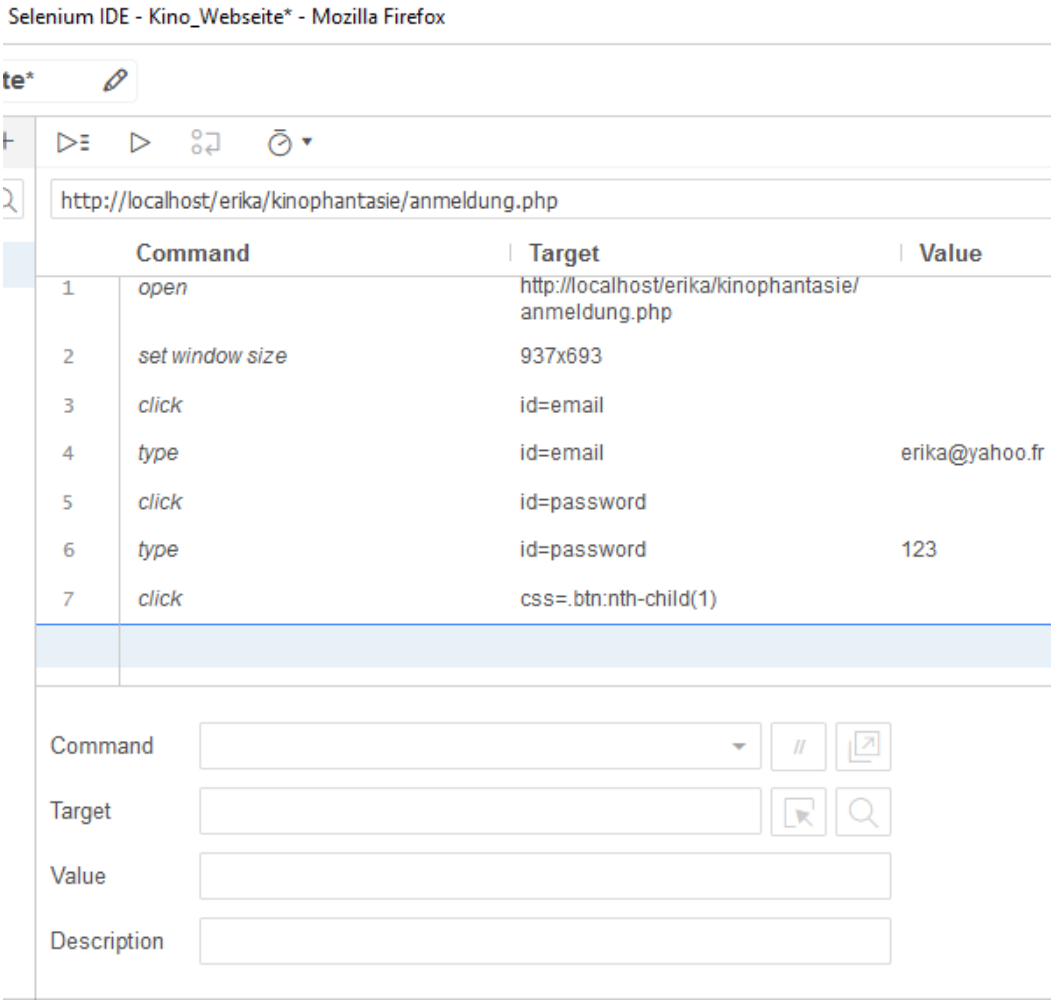


ABBILDUNG 18: TESTAUSFÜHRUNG SELENIUM IDE

Um die Testfälle oder Szenarien wiederabzuspielen, muss einfach die grüne Schaltfläche gedrückt werden. Dadurch werden die Befehle abgearbeitet und Mozilla Firefox oder Google Chrome öffnet die Seite und klickt die Links automatisch bis zum Ende des Testfalls. Das Speichern oder Exportieren der Tests ist ebenfalls möglich, wobei die Programmiersprache, wie bereits oben erläutert, ausgewählt werden kann.

In den Eingabefeldern Befehl („Command“), Ziel („Target“) und Wert („Value“) wird der aktuell ausgewählte Befehl mit seinen Parametern angezeigt. Es handelt sich um Eingabefelder, in denen der aktuell ausgewählte Befehl geändert werden kann (Vgl. ThoughtWorks – Selenium Dokumentation (2021))

Wenn ein Testfall wieder abgespielt wird, werden im unteren Fenster in der Registerkarte „Log“ (Protokoll) automatisch, wie es unten zu sehen ist, Fehlermeldungen und Informationen über den Fortschritt angezeigt. Diese Meldungen sind oft für das Debugging von Testfällen nützlich.

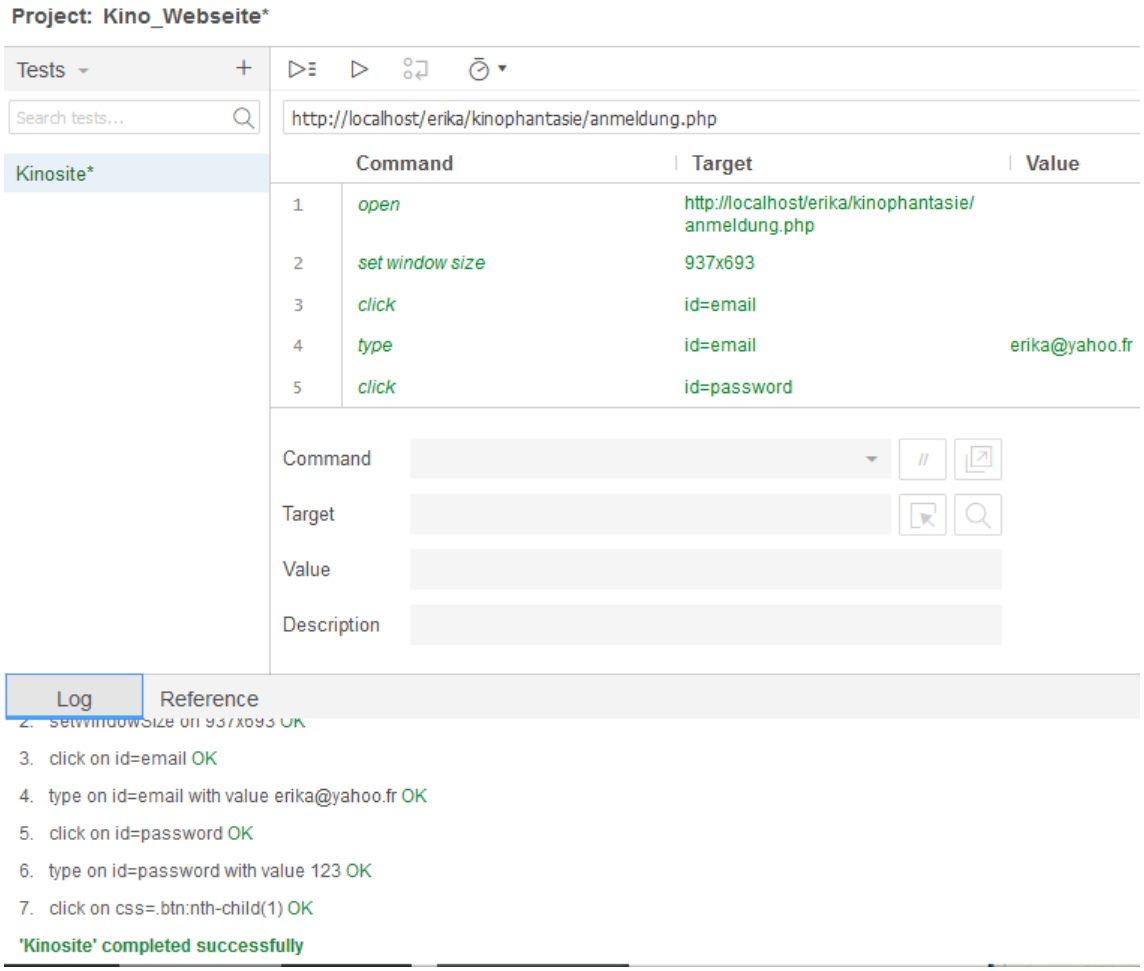


ABBILDUNG 19: AUSGABE SELENIUM IDE

6.1.4 QF-Test

QF-Test ist eine plattformübergreifende Software der Firma Quality First Software GmbH zum automatischen Testen von grafischen Benutzeroberflächen (Java, Web und Windows Anwendungen). Bei der Komponenten-Erkennung erkennt QF-Test zuverlässig auch komplexe Elemente, wie dynamische Bäume und Tabellen. Da die Tests tolerant bezüglich Änderungen an der grafischen Oberfläche sind, ergibt sich daraus ein geringer Wartungsaufwand und eine hohe Wiederverwendbarkeit von Tests. Durch diese Komponenten sind Objekte, wie Buttons oder Textfelder stark abgegrenzt und leicht zu finden (Vgl. Quality First Software GmbH-Dokumentation (2020)). QF-Test ist auf Englisch und Deutsch verfügbar und besitzt jeweils eine passende Dokumentation und Support.

Mittels Record/Replay, wie bei Selenium die, können Tests erstellt und abgespielt werden, um automatische Tests ohne Programmierung zu erzeugen: Dabei können die Aufzeichnungen jederzeit ergänzt und geändert werden. Auch die Programmiersprache, in der die Testfälle gespeichert werden soll, kann ausgewählt werden. Dabei sind Jython,

Groovy, und JavaScript die Skriptsprachen (Vgl. Quality First Software GmbH-Dokumentation (2020)). Die Abbildung 20 zeigt die grafische Oberfläche von QF-Test.

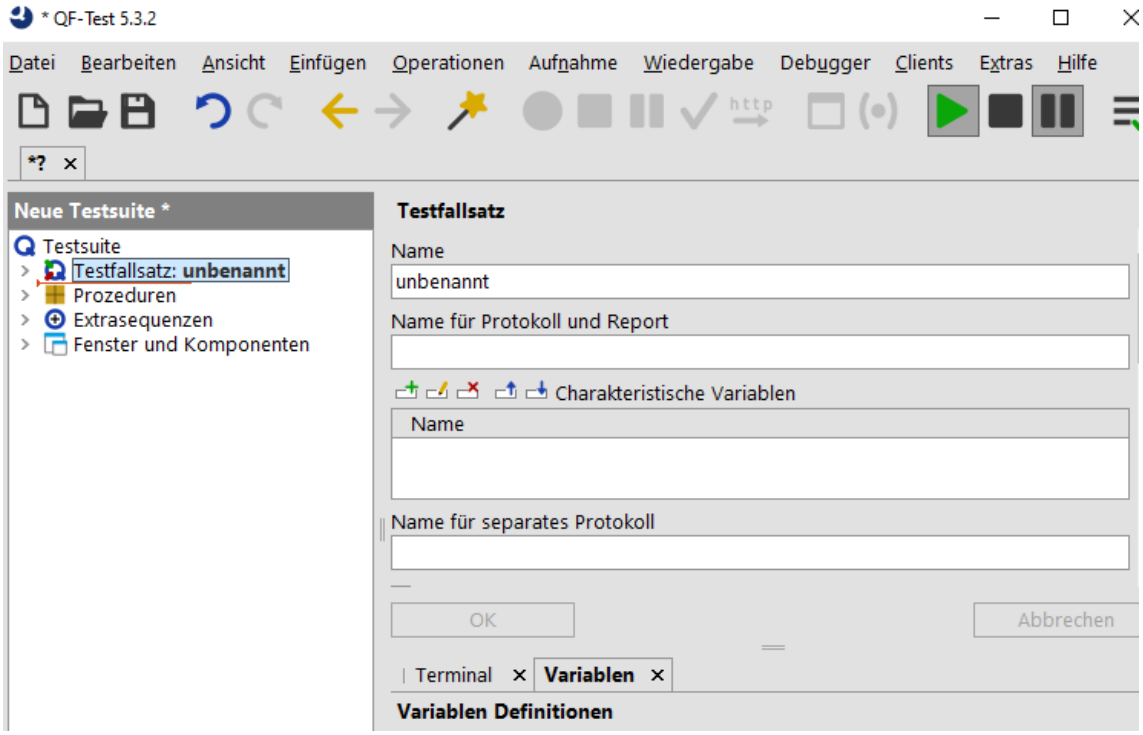


ABBILDUNG 20: GRAFISCHE OBERFLÄCHE VON QF-TEST

Die Abbildung 21 zeigt das Hauptfenster von QF-Test. Auf der linken Seite ist eine Testsuite in Form eines hierarchischen Baums repräsentiert, der Kontrollstrukturen und Testdaten vereint und eine übersichtliche Verwaltung sowie einfache Bedienung ermöglicht. Auf der rechten Seite ist die Detailansicht mit allen Attributen des aktuell ausgewählten Knotens im Baum. Nach der Installation von QF-Test muss in einer TestSuite zunächst die Verbindung zum SUT aufgebaut werden. Schließlich kann über die Symbolleiste mit Hilfe des Record-Buttons ein Test aufgezeichnet und gestoppt werden. Weiterhin erscheint dieser Test im linken Fenster von QF-Test und kann sofort bearbeitet oder wiedergegeben werden, was die Aufzeichnung des Tests schnell und einfach macht. Nach einem Test kann ein Protokoll geöffnet werden, das alle Schritte des durchgeführten Tests auflistet.

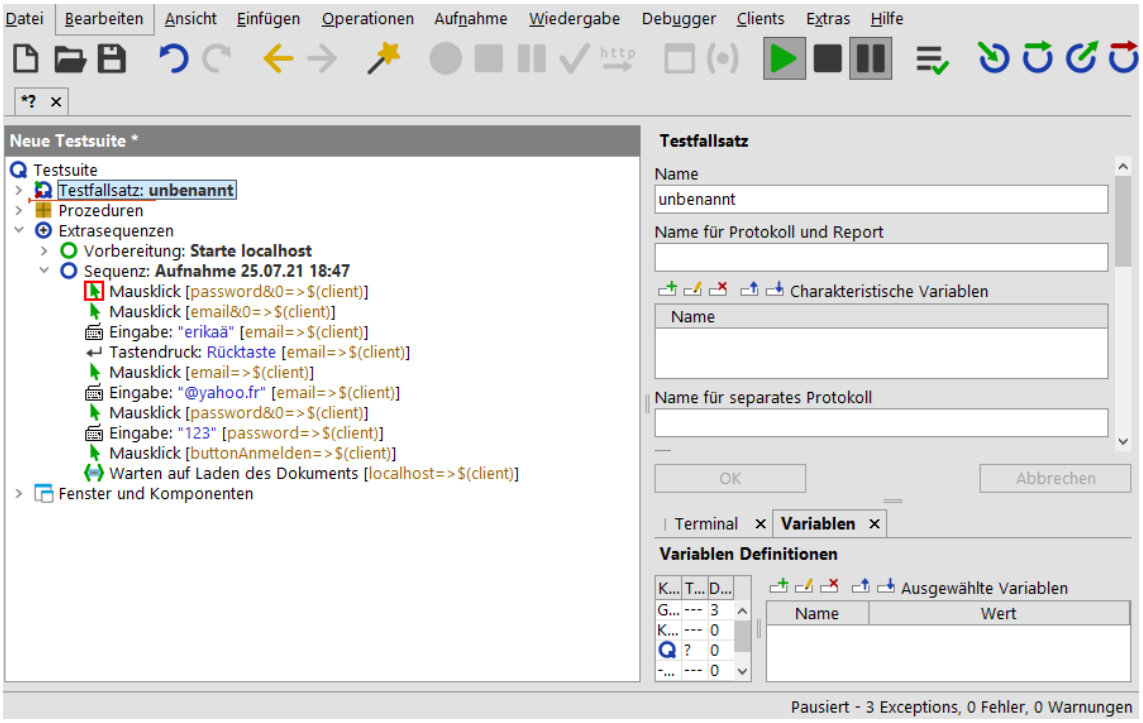


ABBILDUNG 21: TESTAUSFÜHRUNG MIT QF-TEST

6.1.5 Appium

Appium ist ein Open-Source-Tool zur Automatisierung von nativen, mobilen Web- und hybriden Anwendungen auf den Plattformen iOS Mobile, Android Mobile und Windows Desktop. Appium ermöglicht die Nutzung und Erweiterung des bestehenden Selenium WebDriver Frameworks, um mobile Tests zu erstellen. Da es Selenium WebDriver zur Steuerung der Tests verwendet, können mit jeder Programmiersprache Tests für eine Selenium-Client-Bibliothek erstellt werden (Vgl. Tom Christie Appium-Dokumentation (2014)).

Für das Erstellen von Appiumtests müssen Android Studio und Appium Desktop Client auf dem Computer installiert sein. Mittels diesem Tool wird eine ausgesuchte Applikation auf dem Android getestet. Der Test mit Appium erfolgt wie folgt:

- **Einrichten des Android-SDKs (Software Development Kit)**

  Ein Android SDK (Android-Softwareentwicklung) ist der Prozess, durch den mobile Apps für Geräte mit dem Betriebssystem Android erstellt werden. Es muss ein Android SDK (Software Development Kit) von https://developer.android.com/studio/ installiert werden. Dann wird das installierte Android Studio gestartet. Es wird ein beliebiges Android heruntergeladen, dessen API-Level 28 ist, und wird installiert. Dazu wird zu Tools | SDK Manager navigiert und es wird schließlich etwas Ähnliches zu sehen sein, wie im folgenden Screenshot gezeigt wird.

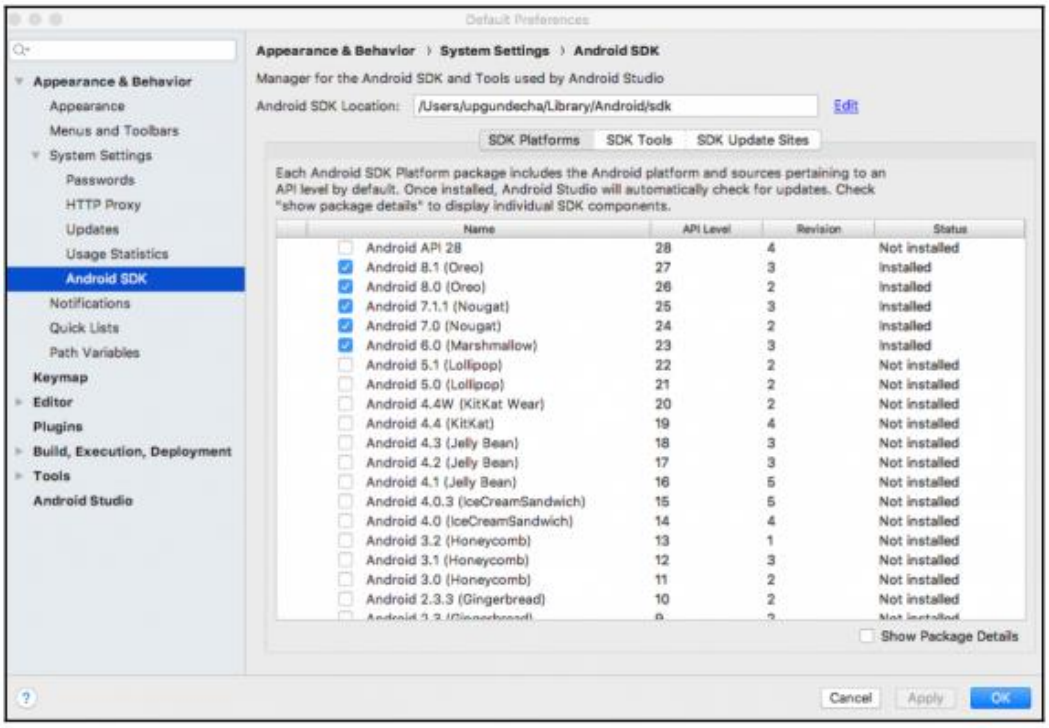


ABBILDUNG 22: INSTALLATION ANDROID 8.1

- **Erstellen des Android-Emulators**

Falls die Testskripte auf einem Android-Emulator ausgeführt werden, muss einen Emulator erstellt werden. Zum Erzeugen von einem Android Emulator sind die folgenden Schritte auszuführen:

In Android Studio den AVD-Manager (Android Virtual Device) öffnen, indem zu Extras | AVD Manager navigiert wird. Es startet den AVD-Manager, wie in der folgenden Darstellung gezeigt wird. Um ein neues virtuelles Gerät oder einen Emulator zu erstellen, muss auf die Schaltfläche „Device Definition“ geklickt werden. Es wird sich ein Fenster öffnen, das alle notwendigen Informationen von dem Android abfragt.

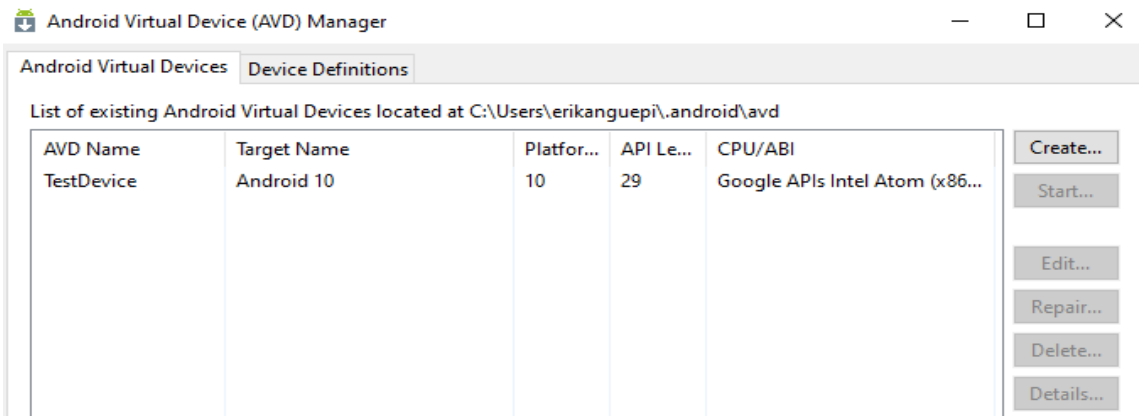


ABBILDUNG 23:VIRTUAL DEVICE ANLEGEN

- **Installation von Appium**

Appium kann von http://appium.io/ heruntergeladen und gestartet werden. Um zu starten, muss auf die Schaltfläche Start Server geklickt werden. Standardmäßig startet er auf http://localhost:4723. Dabei handelt es sich um die Remote-URL, an die die Testskripte die Testbefehle richten sollen.

QUELLE: http://appium.io/

ABBILDUNG 24: APPIUM SERVER

Testfall erstellen

Als Android wird hier ein echtes Gerät verwendet anstelle eines Emulators (ein Simulator wird in der Android-Community als Emulator bezeichnet). Für dieses Beispiel wird das Android-Gerät Samsung Galaxy S8 verwendet. Es muss die „Apk Info“ installiert werden, um Informationen über die App abzulesen, falls sie nicht auf dem Gerät vorinstalliert ist. Als nächstes muss das Gerät mit dem Rechner verbunden werden, auf dem der Appium-Server läuft. Der Befehl „ ./adbdevices “ muss ausgeführt werden, um eine Liste der Emulatoren oder Geräte zu erhalten, die mit dem Gerät verbunden sind.

Die Android Debug Bridge (ADB) ist ein im Android-SDK verfügbares Befehlszeilen-Tool, mit dem man mit einem tatsächlichen Android-Gerät kommunizieren kann, das an dem Computer angeschlossen ist. Der Befehl „ ./adbdevices“ zeigt eine Liste aller Android-Geräte an, die mit dem Host verbunden sind, wie die folgende Ausgabe zeigt:

List of devices attached
ce0617169mh8e640520c device

Im folgenden Beispiel wird der Wert der Fähigkeit plattformName (DesiredCapabilities) auf Android gelegt, die von Appium verwendet wird, um Tests auf Android auszuführen. Da die Tests in Chrome für Android ausgeführt werden, wird Chrome im Abschnitt Browser-Fähigkeit des Codes erwähnt. Die andere wichtige Änderung, die vorgenommen wird, ist die Verwendung der AndroidDriver-Klasse aus den Appium Java-Client-Bibliotheken.

Appium verwendet das erste Gerät aus der Liste der Geräte, die „adb“ zurückgibt, wie im folgenden Screenshot zu sehen ist.

```
1  package appiumtests;
2  import java.net.URL;
3  import org.openqa.selenium.By;
4  import org.openqa.selenium.remote.DesiredCapabilities;
5  import io.appium.java_client.AppiumDriver;
6  import io.appium.java_client.MobileElement;
7
8  public class calculator {
9      static AppiumDriver<MobileElement> driver;
10     public static void main(String[] args) {
11         try {
12             taschenrechnerÖffnen();
13         }catch(Exception exp) {
14             System.out.println(exp.getCause());
15             System.out.println(exp.getMessage());
16             exp.printStackTrace();
17         }
18     }
19     public static void taschenrechnerÖffnen()throws Exception{
20         //Set the desired capabilities for Samsung 8
21         DesiredCapabilities cap = new DesiredCapabilities();
22         cap.setCapability("deviceName", "SM-G950F");      //Gerätename
23         cap.setCapability("udid", "ce0617169mh8e640520c");  //GeräteID
24         cap.setCapability("platformName", "Android");
25         cap.setCapability("platformVersion", "9.0");
26         cap.setCapability("automationName", "UiAutomator1");
27         cap.setCapability("appPackage", "com.sec.android.app.popupcalculator");
28         cap.setCapability("appActivity", "com.sec.android.app.popupcalculator.Calculator");
29         //Send DesiredCapabilities to the Appium server
30         URL url = new URL ("http://127.0.0.1:4723/wd/hub");
31
32         driver = new AppiumDriver<MobileElement>(url, cap);
33         System.out.println("Application started...");
```

```
34
35          //Elemente finden
36          MobileElement two = driver.findElement(By.id,"").click;
37          MobileElement plus = driver.findElement(By.id,"").click;
38          MobileElement three = driver.findElement(By.id,"").click;
39          MobileElement equals = driver.findElement(By.id,"").click;
40          MobileElement plus = driver.findElement(By.id,"").click;
41          MobileElement result = driver.findElement(By.className("")).click;
42          System.out.println("Ergebnis ist: " + result.getText());
43      }
44  }
```

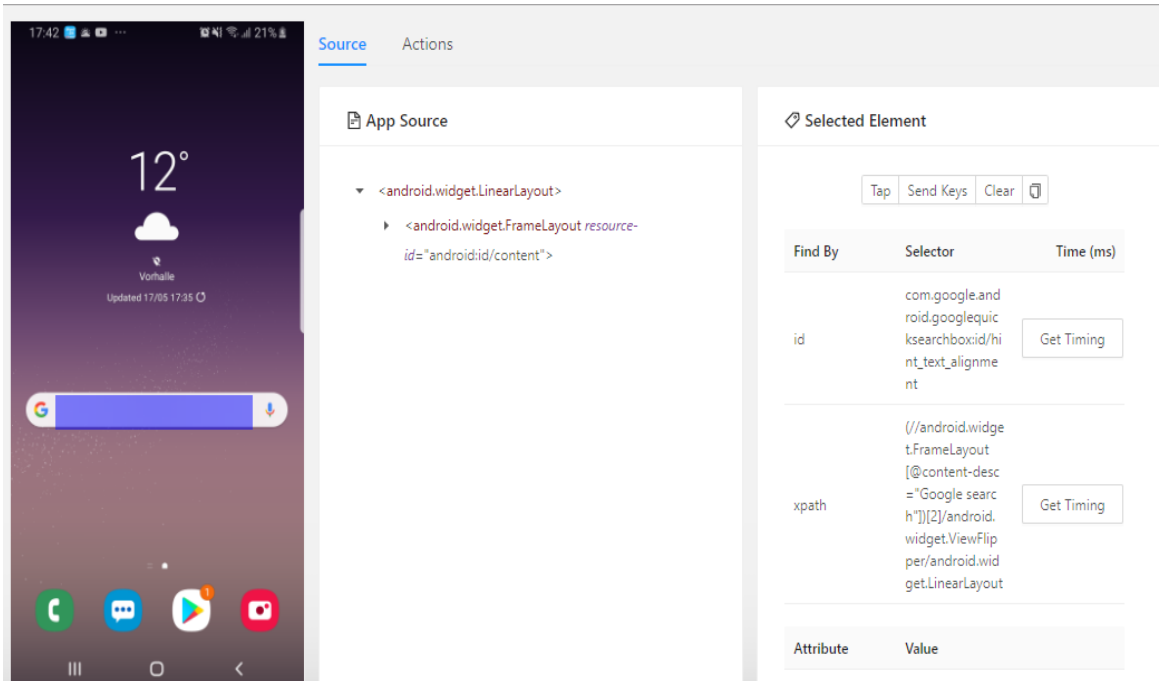
QUELLE: Eigene Darstellung

ABBILDUNG 25: SELENIUM SCRIPT

- **Elemente in Appium untersuchen**

Mit Hilfe von Appium Server wird jedes Element und seine Locators gefunden. Damit diese Aktion durchgeführt wird, muss auf das Element im Screenshot-Bild geklickt oder es im Quellcode-Baum lokalisiert werden. Für das Starten der Testautomatisierung müssen folgende Punkte beachtet werden:

1. Befehl „adbdevices“ auf der Kommandozeile prüfen
2. Gerät bereitmachen
3. Entwicklermodus aktivieren
4. USB-Debugging einschalten
5. Gerät über ein USB-Kabel mit dem Computersystem verbinden
6. USB-Debugging auf Wunsch aktivieren



QUELLE: Eigene Darstellung

ABBILDUNG 26: ELEMENTE UNTERSUCHEN

6.1.6 Ranorex Studio

Ranorex Studio ist ein einfaches Testwerkzeug für automatisierte Tests und wurde von der Firma Ranorex GmbH entwickelt. Es ist eine bessere Lösung als andere Testtools, da es Anwendungen aus der Sicht eines Benutzers testet. Ranorex unterstützt das Testen von Desktop-, Web- und mobilen Anwendungen auf einer Vielzahl von Plattformen, wie Android, iOS und Windows. Außerdem werden Testskripte in C #, Python, C++, VB.net und XML geschrieben. Ranorex bietet als Funktionen die wiederverwendbaren Testcodes, Integration mit verschiedenen Tools, GUI-Erkennung, Aufzeichnung und Wiedergabe, Fehlererkennung. Zusätzlich bietet es Kapazitäten für die Durchführung von Regressionstests und garantiert die Wiederverwendbarkeit von Testaktivitäten.

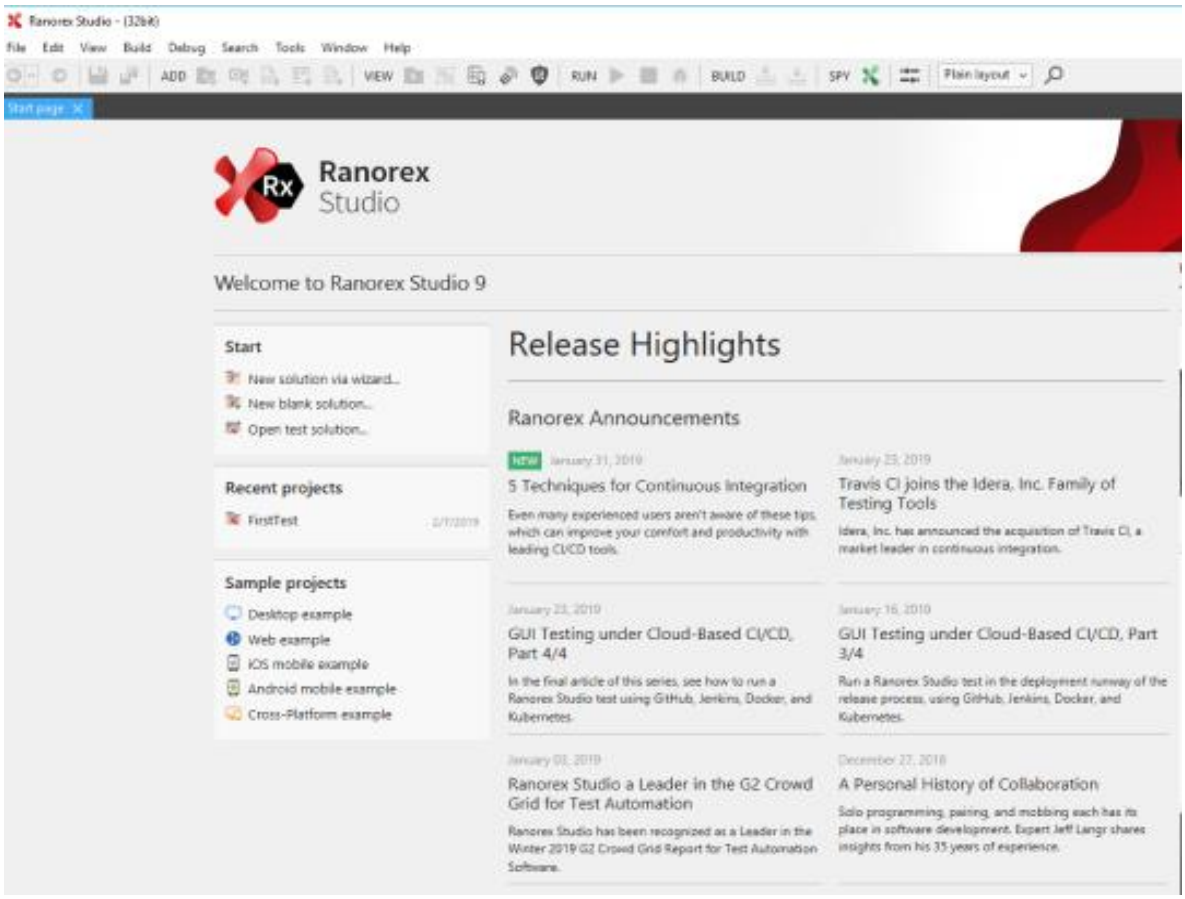
Nach dem Start von Ranorex Studio stellt die Software eine übersichtliche Menüführung und eine Reihe von Hilfedialogen zur Verfügung, die den Benutzer durch das Programm begleiten, wie in der folgenden Abbildung dargestellt ist. Die Dokumentation und die Software selbst sind nur auf Englisch verfügbar (Vgl. Ranorex GmbH-Dokumentation (2021)). Für die Erstellung eines Testfalls führt das Programm den Benutzer durch verschiedene Masken. Der Benutzer kann festlegen, ob das zu erstellende Projekt in Programmiersprache erstellt und verwaltet werden soll.

Ranorex bietet eine breite Reihe von Werkzeugen und Methoden an, um so viele Testherausforderungen wie möglich abzudecken. Einem Projekt können mehrere sogenannte Lösungen untergeordnet werden und das Ganze ist in einer Art Klassenstruktur strukturiert. Darüber hinaus gibt es Dateien und Ordner für die Konfiguration, Testberichte, Referenzen zu Plugins und die eigentlichen Testschritte.

Ein Testschritt ist hier nicht Aktion wie ein Klick auf eine Oberfläche, sondern entspricht einem übergeordneten Schlüsselwort, wie zum Beispiel „Login“. Diese Schlüsselwörter können als Bestandteile einer Maske zu neuen Interaktionen/Testfällen per Drag & Drop oder per Copy & Paste zusammengesetzt werden. Um einen Testfall einfach anzulegen, muss er manuell getestet und dann mit dem eingebauten Rekorder von Ranorex aufgezeichnet werden. Der Rekorder unterstützt hierbei Modi wie die bild- oder textbasierte Erkennung und die manuelle oder automatische Mausverfolgung. Standardmäßig verfolgt der Rekorder automatisch alle Aktionen mit der Maus und der Tastatur.

Fertige Aufzeichnungen werden als Code in einer GUI erstellt und können auf Basis der gewählten Programmiersprache oder per Drag & Drop und über Auswahlmenüs geändert

werden. Außerdem erscheinen die Auswahlmenüs jeweils mit Klick auf einen der Testschritte. Zum Beispiel, wenn auf die Aktion "Maus" geklickt werden soll, kann der Benutzer die Bewegung der Maus (klicken, bewegen, halten, loslassen), aber auch die Taste, mit der interagiert werden soll, bearbeiten (Vgl. Ranorex GmbH-Dokumentation (2021)).



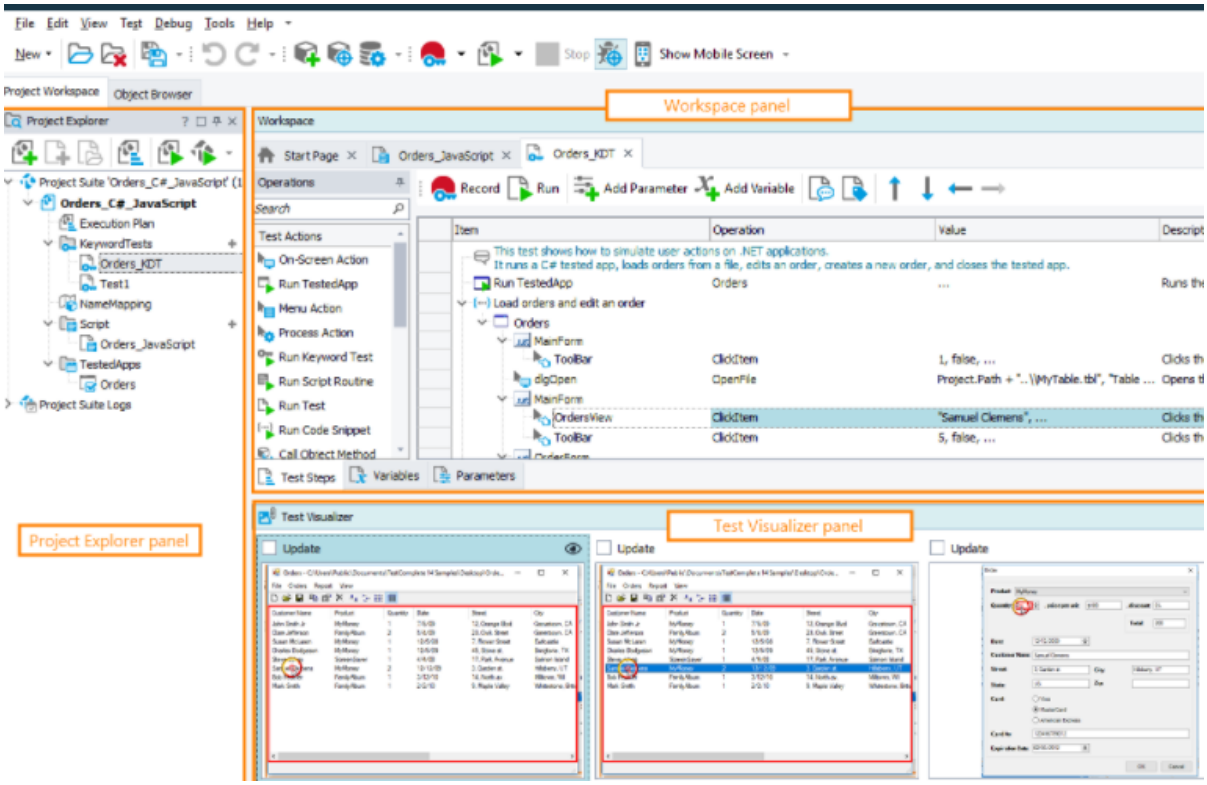
QUELLE: Ranorex GmbH-Dokumentation (2021)

ABBILDUNG 27: STARTANSICHT VON RANOREX STUDIO

6.1.7 TestComplete

TestComplete ist eine Software zur Testautomatisierung von Windows, Web und mobile Anwendungen und wurde von der Firma „SmartBear Software“ entwickelt. Es ist sowohl auf funktionale als auch auf Unit-Tests ausgerichtet. Es bietet hervorragende Unterstützung für tägliche Regressionstests und unterstützt viele andere Arten von Tests: datengesteuerte Tests, verteilte Tests und andere. Die Erstellung von für die Tests verwendeten Schlagworte ist visuell, einfach und erfordert keine Programmierkenntnisse. Diese Tests können aufgezeichnet werden und die automatisierten Skripte können aus Python, C++Script, VBScript oder auch JavaScript-Sprachen zusammengestellt werden (Vgl. SmartBear Software-Dokumentation (2021)).

Zur Erstellung von Tests werden sie aufgezeichnet oder Testbefehle werden in TestComplete-Panels und -Editoren bearbeitet. Die Tests können aus TestComplete heraus ausgeführt werden oder sie können in eine externe Anwendung exportiert und dort ausgeführt werden. Es bietet auch spezielle Checkpunkte, mit denen der Zustand der Anwendung während des Testlaufs leicht überprüft werden kann. Wenn die eingebauten Mittel nicht ausreichen, um Benutzeraktionen in der getesteten Anwendung zu simulieren oder den Zustand der Anwendung zu überprüfen, können die Vorteile des Zugriffs auf die internen Objekte, Methoden und Eigenschaften der Anwendung genutzt werden, um die erforderlichen Aufgaben durchzuführen.



QUELLE: SmartBear Software (2021)

ABBILDUNG 28: GRAFISCHE OBERFLÄCHE VON TESTCOMPLETE

In der Abbildung 28 ist die Benutzeroberfläche von TestComplete in einer Reihe von Bedienfeldern zu sehen. Der „Project-Explorer panel“ auf der linken Seite des Fensters zeigt den Inhalt der Projekte und der Projektsuite an. Es bietet auch Links zu den Testprotokollknoten. Das „Workspace panel“ ist die Arbeitsoberfläche: Es zeigt die Projekt- und Projektelementeditoren an, in denen Tests erstellt, geändert und Testergebnisse angezeigt werden können. In der obigen Abbildung ist zum Beispiel der Editor für den Schlüsselworttest zu sehen, der im Arbeitsbereich geöffnet ist. Unterhalb des Editors befindet sich ein „Test-Visualizer panel“, das Bilder anzeigt, die von der Test-Engine während der Aufzeichnung von Testbefehlen erfasst wurde. Diese Bilder helfen, die Aktionen

zu verstehen und die Testbefehle auszuführen. TestComplete enthält weitere Bereiche wie z. B. „Watch List“, „Locals“, „Breakpoints“ und „Call Stack“ - sie werden zum Beispiel für das Debugging von Tests verwendet. Das Panel "To Do" verwaltet eine Liste von zu erledigenden Aufgaben (Vgl. SmartBear Software-Dokumentation (2021)).

6.1.8 UFT (Unified Functional Testing)

Unified Functional Testing (UFT), früher bekannt als QuickTest Professional (QTP), ermöglicht die Erstellung von GUI Tests und API-Tests und wurde von Micro Focus entwickelt. Es nutzt die Visual Basic Script-Sprache, um die Skripte auszuführen und die Testformen einzutragen. Es arbeitet ebenfalls mit verschiedenen Objekten und steuert das Testen von Anwendungen. UFT bietet der Branche eine nützliche Lösung für die Automatisierung von Regressionstests und funktionalen Tests, die alle wichtigen Implementierungen und Umgebungen von Software abdeckt. UFT wird in der Regel nicht nur für die Automatisierung von UI-basierten Testfällen verwendet, sondern auch einige nicht UI-basierte Testfälle, wie Dateisystemaktivitäten und Datenbanktests, können automatisiert werden.

UFT verfügt über eine IDE (Integrated Development Environment), die benutzerfreundlich und einfach ist und daher auch von Nicht-Programmierern leicht zu bedienen und zu verstehen ist, so dass sie mit Leichtigkeit Testfälle erstellen und hinzufügen können. Es unterstützt verschiedene Addins wie Java, Oracle, SAP aber ein großer Nachteil ist, dass die Lizenz- und Wartungskosten hoch sind, und es können nicht mehrere Prozesse oder Instanzen ausgeführt werden (Vgl. Micro Focus-Dokumentation (2021)).

6.2 Vergleichskriterien

Wenn man mit der Suche nach der richtigen automatisierten Software anfängt, ist es wichtig, eine Liste von Anforderungen zu erstellen, die bei der Auswahl eines Werkzeugs für die Evaluierung helfen kann. Falls keine Liste der Anforderungen vorliegt, kann die Zeit mit dem Herunterladen, Installieren und Evaluierung von Tools, die nur einige der Anforderungen erfüllen oder vielleicht gar nicht erfüllen, vertan werden. Diese Arbeit bewertet acht Tool-Anbieter, nämlich TestCafe, Selenium Webdriver, Selenium IDE, QF-Test, Appium, Ranorex Studio, TestComplete (TC) und Unified Functional Testing (UFT) auf folgende Kriterien: Effizienz der Aufzeichnung, Fähigkeit zur Erstellung von Skripten, Berichte über Testergebnisse, Wiederverwendbarkeit, Werkzeuge Testarten, Kosten. Mittels dieser Kriterien werden die Unterschiede zwischen den Tools vorgestellt.

6.3 Unterschied zwischen den Testtools

Bei Selenium IDE, QF-Test, Ranorex Studio, TestComplete (TC) und Unified Functional Testing (UFT) handelt es sich um Aufnahme- und Wiedergabewerkzeuge. Während der Aufzeichnung werden Befehle eingefügt, um zu prüfen, ob die Anwendung wie vorgesehen funktioniert. Im Gegensatz zu diesen Tools werden bei TestCafe, Selenium Webdriver und Appium-Programme geschrieben, die später durchgeführt werden. Die Unterschiede werden mittels der oben genannten Kriterien behandelt.

- **Effizienz der Aufzeichnung**

  QF-Test kann im Hinblick auf die Erfassung und Wiedergabe überzeugen: Die Erfassung von Aktionen, Prüfungen und die direkte Bearbeitung der erfassten Schritte gehören zu den Grundfunktionalitäten von QF-Test.

  Selenium IDE erfasst lediglich Aktionen und Prüfungen über das Firefox-Plugin, die Bearbeitung der erfassten Schritte ist jedoch eingeschränkt. Außerdem wird Selenium IDE nur zögerlich weiterentwickelt (ThoughtWorks – Selenium Dokumentation (2021)).

  Mit dem Ranorex Recorder können die Tastatur- und Mausaktionen aufgezeichnet werden, die für einen Test der Benutzeroberfläche notwendig sind. Diese Aktionen erscheinen in der Aktionstabelle des Schreibers, wo sie bearbeitet oder weitere hinzugefügt werden können. Auf diese Weise können Aufnahmen erstellt werden, die ihren Testanforderungen entsprechen (Vgl. Ranorex GmbH-Dokumentation (2021)).

  TestComplete erlaubt einen sehr einfachen Zugang zu den Kontrollen. Während der Aufzeichnung ist die TestComplete-Aufnahmefunktion immer in der Anwendung und zeigt an, dass sie die Benutzeraktionen aufzeichnet. Die Aufnahme-Symbolleiste enthält alle Steuerelemente. Daher können leicht die Kontrollpunkte angewendet, Text hinzugefügt und auch der Bildschirm und die Fensterkoordinaten gesehen werden. TestComplete bietet eine einfache Möglichkeit, die Aufnahme in der Mitte zu unterbrechen. So können die zu testende Anwendung manipuliert oder die Umgebung geändert werden, ohne diese Aktionen im Skript aufzuzeichnen (Vgl. SmartBear Software-Dokumentation (2021)).

  UFT bietet keinen einfachen Zugang zu Steuerelementen. Wenn auf die Aufzeichnungsschaltfläche gedrückt wird, wird die Anwendung gestartet. Alle vom Benutzer durchgeführten Aktionen werden aufgezeichnet. UFT stellt drei Arten von Aufzeichnungen zur Verfügung, nämlich den kontextsensitiven Modus, analoger Modus und

Low-Level-Aufnahme. Es ist nicht möglich, den Test während der Aufzeichnung anzuhalten (Vgl. Micro Focus-Dokumentation (2021)).

- **Fähigkeit zur Erstellung von Skripten**

Selenium und Appium verfügen über sehr viele Programmiersprachen (Java, C#, JavaScript, Python, PHP, Ruby), während TestCafe nur auf JavaScript, TypeScript, CoffeeScript basiert. Das macht Selenium flexibler, da in der favorisierten Sprache gearbeitet werden kann. Anderseits, wenn man in JavaScript erfahren ist, kann man mit beiden Testwerkzeugen arbeiten. Selenium erfordert gute Entwicklungskenntnisse, um die Selenium-IDE und den Selenium-Server konfigurieren zu können. Das ist bei QF-Test anders: Auch Nicht-Entwickler können einfach und schnell mit dem Tool arbeiten. Bei Ranorex Studio sind die Skripte in C#, VBNet zu erzeugen.

TestComplete kann fünf Arten von Skripten erzeugen, nämlich VbScript, Delphi, C++, C# und JScripts (Vgl. SmartBear Software-Dokumentation (2021)). Wenn eine Applikation auf einer dieser Anwendungen basiert, kann TC die entsprechenden Skripte leicht erstellen. Jedoch erzeugt UFT nur VbScripts.

- **Wiederverwendbarkeit von Skripten**

Um Selenium und TestCafe-Tests wiederverwenden zu können, benötigt es spezielle Kenntnisse in objektorientierter Programmierung. Ein flexibles und erweiterbares Entwurfsmuster ist in QF-Test eingebaut.

TC verwendet die Funktion der Namenszuordnung, um die Eigenschaft des Objekts zu ändern. Die Namenszuordnung zeigt alle Objekte in der Baumansicht. Das Objekt wird ausgewählt und die Eigenschaft wird geändert, die auf der rechten Seite angezeigt wird. Nachdem die Änderungen vorgenommen sind, kann das gleiche Skript ausgeführt werden. UFT dagegen hat eine eingebaute Funktionsbibliothek. Sie bildet das Herzstück der Automatisierung im Framework. Die gesamte Kodierungslogik ist in Form eines benutzerdefinierten VB-Skripts. Alle diese Funktionen sind in der Funktionsbibliothek. Es ist der Ort, an dem sich die meisten Skripte befinden und der Ort, an dem Anpassungen im Skript für das Projekt vorgenommen werden. Die gemeinsamen Skripte sind leicht wiederverwendbar. Wenn Anwendung modifiziert wurden, indem einige Eigenschaften der Objekte geändert wurden, kann dasselbe Skript für den neuen Build verwendet werden.

- **Berichte über Testergebnisse**

Nach der Ausführung des Testskripts ist es notwendig, die Ergebnisse der Ausführung für die Durchführung einer effektiven Analyse zu erzielen überprüfen, ob die Testskripte bei der Ausführung einer Testsuite bestanden haben oder nicht.

Während des Testablaufs sammelt TestCafe Informationen über den Testablauf und gibt den Bericht in einer Befehlshell aus. Bei der Durchführung von Tests kann ein Reporter ausgewählt werden, um Testberichte zu erstellen. Dies erfolgt mit der Befehlszeilenoption „-r (--reporter)“ oder mit der API-Methode “ runner.reporter“. Bei Selenium Webdriver und Appium dagegen ist der Bericht auf der Konsole zu sehen. Selenium IDE zeigt Informationen über alle Testschritte.

Beim Abspielen eines Tests erzeugt QF-Test einen Bericht, in dem jede einzelne Aktion vermerkt wird. Die Protokolle der zuletzt ausgeführten Tests sind über das „Wiedergabe“-Menü zugänglich. Die Struktur des Protokolls ist so aufgebaut, dass Knoten bei ihrer Ausführung in das Protokoll aufgenommen werden.

Jeder Testlauf in Ranorex Studio endet mit einem Bericht, um zu wissen, ob der Testlauf erfolgreich war oder nicht. Der Bericht beschreibt den gesamten Testlauf von Anfang bis Ende, d. h. wie viele Testfälle erfolgreich waren, fehlgeschlagen sind oder blockiert wurden, welche Fehler und Warnungen aufgetreten sind. Die Berichte können angepasst werden, um zu ändern, welche Informationen angezeigt werden.

TC zeigt das Ergebnis der Ausführung. Aber es zeigt alle Ereignisse, die während der Wiedergabe aufgetreten sind in einem Fenster. Es gibt keine Informationen über jeden Testschritt und keine grafische Darstellung der Ergebnisse.

UFT liefert auch eine Zusammenfassung des Tests. Es zeigt die Testschritte in einem Hierarchiebaum und bietet außerdem eine Zusammenfassung des einzelnen Schritts - im Gegensatz zu TestComplete. Es liefert auch Informationen über Checkpunkte, die während der Prüfung gelten und UFT gibt die Statistiken über den vorherigen Lauf und den aktuellen Durchlauf in Form von Diagrammen an.

6.4 Ergebnisse

Nach umfangreicher Recherche in der Literatur zu folgenden Testwerkzeugen werden in der nachstehenden Tabelle die benötigten Informationen über die Tools zusammenfassend dargestellt, die auf bestimmten ausgewählten Kriterien basieren. Diese Analyse kann den Testern helfen, das beste Werkzeug zum Testen von Software auszuwählen. In der Tabelle sind noch einmal die Eigenschaften der Tools zusammengefasst.

Kriterien\Tools	TestCafe	Selenium Web-driver	Appium	Selenium IDE	QF-Test	Ranorex	TestComplete	UFT
Programmiersprache	JavaScript, TypeScript, CoffeeScript	Java, C#, Perl, Python, JavaScript, Ruby, PHP	Java, C#, Perl, Python, JavaScript, Ruby, PHP	Java, C#, Perl, Python, JavaScript, Ruby, PHP	Java	C #, Python, C++, VB.net and XML	VbScript, Delphi, C++, C# und JScripts	VbScript
unterstützende Plattform	Web	Web	Mobile, Windows	Web	Java, Web, Desktop(Win)	Mobile, Web, Desktop(Win)	Mobile, Web, Desktop(Win)	Web, Desktop (Win)
Berichte über Testergebnisse	ja	ja	ja	ja	ja	ja	ja	ja
Wiederverwendbarkeit	ja	ja	ja	nein	ja	ja	ja	ja
Aufnahme/Wiedergabe	nein	nein	nein	ja	ja	ja	ja	ja

leicht zu lernen	ja	ja	nein	ja	ja			
Programmierkenntnisse	ja	ja	ja	ja	nein	nein	nein	nein
Werkzeuge Testarten	End-to-End Test	Funktional Unit Test, End-to-End Test	GUI Test, Funktional Unit Test	Funktional Unit Test	GUI Test	GUI Test,	Funktional Unit Test, GUI Test, Unit Test	Funktional Unit Test, Regression Test
Lizenz	Frei	Frei	Frei	Frei	Nicht frei	Nicht frei	Nicht frei	Nicht frei
Quellen	DevExpress - TestCafe Dokumentation (2012-2021)	ThoughtWorks - Selenium Dokumentation (2021)	Tom Christie Appium-Dokumentation (2014)	ThoughtWorks - Selenium Dokumentation (2021)	QF-Dokumentation (2020)	Ranorex GmbH-Dokumentation (2021)	SmartBear Software-Dokumentation (2021).	Micro Focus-Dokumentation (2021)

QUELLE: eigene Darstellung

TABELLE 1: ZUSAMMENFASSUNG VERGLEICH

Aus der Analyse der Werkzeuge ist festzustellen, dass die Mehrheit der Werkzeuge lizenziert sind. Das bedeutet, dass die Tester ein Abonnement abschließen müssen, damit sie für Tests verwendet werden können, obwohl einige Anbieter kostenlose Testtage ermöglichen. Die Open-Source-Tools sind kostenlos und können ohne Abonnement genutzt werden. Die Anwendungen und Plattformen, die jedes Tool unterstützen, sind ebenfalls in der Tabelle zu sehen. Manche Tools unterstützen alle drei Plattformen (Mobile, Web, Desktop), während einige nur eine. Diese zeigen die Art der Anwendungen, die mit diesen Tools getestet werden können. Einige Tools sind einfach zu benutzen und zu erlernen, während andere Programmierkenntnisse erfordern, um sie effektiv nutzen zu können.

Diese Kriterien sollten bei der Auswahl von Testwerkzeugen berücksichtigt werden, da diese sowohl Zeit als auch Kosten sparen, die beim Testen anfallen.

6.5 Gemeinsamkeiten von Testwerkzeugen

- Werkzeuge Testarten

Die in dieser Arbeit vorgestellten Testwerkzeuge sind applikationsbezogene Testarten, da sie die Funktionalität der Software überprüfen.

- Wiederverwendbarkeit von Skripten

TestCafe, Selenium, Appium und Ranorex unterstützten die Möglichkeit, denselben Code für ähnliche Tests wiederzuverwenden, anstatt die Codes vom Grund auf neu schreiben zu müssen.

- Capture/Replay

Die Werkzeuge generieren die automatische Dokumentation der Aktionen, die von den Nutzern durchgeführt werden. Bei einigen der oben genannten Tools handelt es sich um Aufnahme- und Wiedergabewerkzeuge. Während der Aufzeichnung werden Befehle eingefügt, um zu prüfen, ob die Anwendung wie vorgesehen funktioniert.

- Berichte über Testergebnisse

Selenium WebDriver kann in der Ranorex-Umgebung erstellt werden und es können die Möglichkeiten des größten Frameworks für automatisierte Tests genutzt werden. Diese enge Integration ermöglicht es Ranorex, alle Probleme zu lösen, die Selenium noch hat.

6.6 Probleme und Grenzen von Testwerkzeugen

Probleme

Probleme mit den automatisierten Testtools sind die schwierige Installation, die komplizierte Lernkurve, die Schwierigkeit Skripte zu lesen und zu bearbeiten und das Erfordernis von Programmierkenntnissen. Es dauert eine Weile, bis man sich mit dem Tool vertraut gemacht hat, und das Supportmaterial ist nicht sehr hilfreich, da die meisten Tools nur auf Englisch dokumentiert sind. Das heißt, es wird schwer für diejenigen, die kein Englisch verstehen. Die automatisierten Tools können Testfälle in Programmiersprachen aufzeichnen, die für den Tester schwer zu verstehen/bearbeiten sind. Am Ende, wenn der Tester eine gute Bibliothek von Software und eine Menge Erfahrung gesammelt hat, wird

das Werkzeug wirklich einfach zu handhaben, aber er benötigt Zeit, um dieses Ziel zu erreichen.

Grenzen

Selenium benötigt für das Testen mobiler Anwendungen Unterstützung durch Appium, da Selenium nur Webanwendungen unterstützt. Außerdem muss man die Programmiersprache gut beherrschen, um mit Selenium arbeiten zu können (Vgl. ThoughtWorks - Selenium Dokumentation (2021)).

Da TestCafe JavaScript verwendet, kann es manchmal schwierig für Tester sein, die mit JavaScript nicht sehr vertraut sind (Vgl. DevExpress - TestCafe Dokumentation (2012-2021)).

Bei UFT sind die Lizenz- und Wartungsgebühren sehr hoch. TestComplete ist zwar günstiger als UFT, aber nicht so leistungsstark wie UFT (vgl. Micro Focus und SmartBear – UFT und TestComplete Dokumentation).

Ranorex und TestComplete laufen nicht unter Mac OS und erlauben nicht das Testen von Mac-Anwendungen (Vgl. Ranorex GmbH und SmartBear – Ranorex und TestComplete Dokumentation).

7 Einsatzgebiete nach Systemart

Bei der Software kann zwischen verschiedenen Arten von Systemen unterschieden werden. In diesem Kapitel wird beschrieben, welche Aspekte und Möglichkeiten bei der Testautomatisierung berücksichtigt werden müssen.

7.1 Webapplikationen

Webanwendungen stellen einen sehr häufigen und gut dokumentierten Spezialfall von Client-Server-Anwendungen dar. Hierbei gibt es im Allgemeinen keinen spezifischen Client für Applikationen, sondern den Browser. Aufgrund des hohen Standardisierungsgrades der übermittelten Daten (HTTP und HTML) ist es möglich, hier spezifische Methoden anzuwenden, die auf diese Protokolle abzielen und deren Nutzung ausnutzen (z. B. Capture und Replay auf Protokollebene). Viele Werkzeuge unterstützen Webanwendungen explizit. Die Testausführung kann auch in Webanwendungen einfach parallel implementiert werden, da einige Tools nicht über die physische GUI auf die Anwendungsschnittstelle zugreifen können, sondern über JavaScript, oder ihre Tests auf der Ebene der zugrundeliegenden Protokolle und Formate (HTTP und HTML/Text, möglicherweise mit einem In-Memory-Browser für JavaScript-Funktionalität) verarbeiten können, was die Testausführungszeit normalerweise erheblich beschleunigt.

Um die Auswahl eines Tools zu vereinfachen, ist die Frage, ob automatisierte Tests auf verschiedenen Browsern durchgeführt werden sollen, zu beantworten. Dies hängt insbesondere von den Funktionalitäten ab, die innerhalb der Browser zur Verfügung gestellt werden, d. h. von JavaScript, Ajax und ähnliche Techniken. Automatisierte Tests der grafischen Darstellung selbst sind im Allgemeinen ungeeignet, so dass automatisierte Browsertests vor allem dann sinnvoll sind, wenn ein Teil der Funktionalität mit Hilfe dieser Techniken implementiert wird (Vgl. Busics et al. (2015) S. 118).

7.2 Mobile Applikationen

Bei diesen Ansatzbereichen können einige spezielle Herausforderungen auftreten, die eine gesonderte Betrachtung rechtfertigen. Im Folgenden werden diese Herausforderungen und einige Lösungsansätze genau erörtert:

Herausforderungen

- Auswahl der Testplattformen

Ein großes Problem bei der Auswahl der Testplattformen ist die sinnvolle Auswahl einer Teilmenge von Geräten für die Testausführung. Das Hauptproblem für Testautomatisierungsprojekte ist die starke Fragmentierung von Geräteherstellern und Betriebssystemen, während unterschiedliche Versionen von Geräten oder Betriebssystemen relativ einfach zu bedienen sind. Der Grund dafür ist, dass im zweiten Fall lediglich die Testdurchführung auf einem anderen Gerät stattfindet, die automatisierten Testskripte hingegen nicht angepasst werden müssen. Wenn ein Testfall jedoch auf einem anderen Betriebssystem ausgeführt werden soll, muss in den meisten Fällen ein neues Automatisierungsskript geschrieben werden.

- Spezielle Probleme bei der GUI-Testautomatisierung von mobilen Apps

Derzeit befindet sich die Automatisierung mobiler Anwendungen in einem relativ frühen Entwicklungsstadium. Das lässt sich daran erkennen, dass viele Hersteller immer noch auf Bilderkennungsmethoden für die Identifizierung von GUI-Objekten setzen. Das bedeutet, dass selbst kleine Änderungen am Erscheinungsbild einer Anwendung einen erheblichen negativen Einfluss auf die Stabilität der Testautomatisierung haben. Die Erfahrung mit der Testautomatisierung auf Desktop-PCs hat gezeigt, dass eine nachhaltige Testautomatisierung mit dieser oder ähnlichen Methoden nur schwer zu erreichen ist.

- Umgang mit Interrupts (Unterbrechungen)

Unterbrechungen (z. B. eingehende Anrufe, SMS, Push-Benachrichtigungen) stellen sowohl für die Hersteller von Tools als auch für die Testautomatisierer eine große Herausforderung dar. Werden für die Testausführung Emulatoren oder Simulatoren eingesetzt, lassen sich Unterbrechungen relativ einfach simulieren (Vgl. Busics et al (2015) S. 119 ff.).

- Unterschiedliche Hardware der Endgeräte

Insbesondere bei mobilen Geräten gibt es eine Vielzahl von Geräten mit unterschiedlicher Hardware, die sich in verschiedenen Komponenten unterscheiden. Je nach Applikation können diese Variationen für den Test relevant oder auch nicht sein. Für die Automatisierung bedeutet dies, dass in der Regel eine große Anzahl von Endgeräten unterstützt werden müssen und Tests an der Hardware allein nicht besonders effektiv sind. Damit die Automatisierung einen echten Mehrwert für das Projekt bringt, muss es möglich sein, dieselben Testfälle auf mehreren Geräten auszuführen. Dies bedeutet jedoch, dass entweder in physische Endgeräte investiert werden oder Emulatoren und Simulatoren für Tests verwendet werden müssen. Die Vielfalt von Bildschirmgröße, Auflösung und Punktdichte ist in dieser Form auch nur auf mobilen Geräten zu finden. Diese Faktoren sind für die Automatisierung von Desktop-Anwendungen weniger relevant.

Lösungsansätze

- Einsatz von Emulatoren und Simulatoren

Obwohl dieser Ansatz bedeutet, dass nur ein kleiner Teil der funktionalen Anforderungen an einem physischen Gerät getestet werden muss, kann dieser Teil des Prozesses immer noch sehr zeitaufwendig und teuer sein. Ein vielversprechender Weg, diese Probleme zu vermeiden, ist die Verwendung von Emulatoren und Simulatoren als Ersatz für reale Endgeräte. Hauptziele sind sowohl die Senkung der Kosten als auch die Gewährleistung einer flexiblen Geräteauswahl mit der Möglichkeit, neue Geräte schnell und mühelos bei der Durchführung der Tests zu berücksichtigen.

- Multi-Layer-Test

Der Multi-Layer-Test ist einer der wichtigsten Ansätze bei der Automatisierung von mobilen Applikationen. Wegen der Vielfalt an Zielplattformen wird ein Großteil der funktionalen und nicht funktionalen Anforderungen nicht auf dem Endgerät selbst, sondern auf einer technischen Schicht getestet wird. Da diese technische Schicht gerätneutral definiert und implementiert wird, kann davon ausgegangen werden, dass eine auf diesen Ebenen verifizierte Anforderung auch am Endgerät funktioniert. Dazu gehören z. B. Tests der Service oder Business-Layers. Zusätzlich spielen bei diesem Ansatz auch Tests wie Unit Tests, Komponententests eine große Rolle. Dadurch sollten schon viele der Grundfunktionalitäten, wie z. B. Berechnungen, Businessregeln und Ähnliches ausführlich verifiziert werden. Zum Schluss ist durch diesen Ansatz

eine effiziente Testautomatisierung zu erreichen, ohne das Risiko von unerkannten Fehlerzuständen wegen einer zu geringen Testtiefe zu erhöhen.

- Ansatz von Cloud-Services

Durch den Ansatz von Cloud-Services wird die Verwendung von verschiedenen mobilen Endgeräten strukturierter und einfacher durchgeführt. Dabei wird ein Cloud-Service angesprochen, der den Test auf dem gewünschten Gerät, Emulator oder Simulator, durchführt und die Testergebnisse an den Tester zurückgibt. Weiterhin kann mit Cloud-Service eine parallele Testdurchführung auf mehreren Geräten gleichzeitig stattfinden, ohne vorher die Endgeräte physisch über den Computer vorzubereiten. Die Effizienz des Testens wird dadurch erhöht und der Tester wird durch Verringerung der organisatorischen Tätigkeiten entlastet (Vgl. Busics et al. (2015) S. 124 f.).

7.3 Desktop Applikationen

Ein im Softwarebereich immer seltener anzutreffender Fall, der aber immer noch einen großen Teil der bestehenden Softwaresysteme ausmacht, vor allem im Bereich der kleinen Anwendungen, sind Standalone-Desktop-Applikationen. Das sind Anwendungen, die in sich geschlossen sind und außer dem Betriebssystem keine weiteren wichtigen Schnittstellen zu anderen Systemen haben.

In diesen Fällen ist ein automatisiertes Testen über die Benutzeroberfläche oft sinnvoll, zusätzlich zum Entwicklertest auf Komponentenebene. Zu diesem Zweck können Tools verwendet werden, die die Technologie der Anwendung oder ihre Benutzeroberfläche unterstützen. Darüber hinaus kann es erforderlich sein, auf die Testdaten über das Dateisystem (und jedes von der Anwendung verwendete Format) zuzugreifen oder sie zu erstellen.

Außerdem kann es notwendig oder sinnvoll sein, zusätzliche Funktionen in die Anwendung für Testzwecke aufzunehmen, die beispielsweise bei der Vorbereitung des Tests oder der Überprüfung der Ergebnisse helfen. Bei dieser Art von Softwaresystemen ist die Automatisierung viel einfacher, da die Abhängigkeiten von anderen Softwaresystemen begrenzt sind und daher eine begrenzte Anzahl von Schnittstellen wahrscheinlich ausreicht, um einen Funktionstest zu automatisieren (Vgl. Busics et al. (2015) S. 115 f.).

8 Empfehlungen

In der Softwareentwicklung ist die Qualität das wichtigste Ziel eines jeden Projekts, daher wird empfohlen, bei der Auswahl eines Werkzeugs den Umfang des Projekts und die für das Testen veranschlagten Kosten zu berücksichtigen, Auch die Plattform, auf der das Projekt eingesetzt werden soll, sollte sich bei den Kriterien für die Auswahl eines Testwerkzeugs widerspiegeln. Aufgrund der in dieser Arbeit erzielten Ergebnisse werden Ranorex, TestComplete, UFT und QF-Test für das Testen auf allen Plattformen und bei großen Projekten empfohlen. Da es sich dabei um lizenzierte Tools handelt, sollte das Budget für das Testen berücksichtigt werden. Appium wird für mobile Anwendungen empfohlen, während TestCafe, Selenium Webdriver, Selenium IDE für Web-Test zur Verfügung stehen (Vgl. F. Okezie et al. 2019 S. 9).

- **Zielgruppe ohne Programmierkenntnisse**

  Für Benutzer ohne Programmierkenntnisse empfiehlt sich die Verwendung von QF-Test, Ranorex, TestComplete, UFT und Selenium IDE, da diese Software keine Programmierkenntnisse erfordern. Die Tools sind zum Aufzeichnen von automatisierten Testfällen gedacht und verfolgen das gleiche Ziel.

  Diese Tools sind benutzerfreundlich und gut bedienbar und die Benutzeroberfläche ist übersichtlich und erklärt sich von selbst. Dies macht es für Anfänger und unerfahrener Tester einfach. Bei Selenium IDE z. B. werden die möglichen Eingaben mittels Popup vorgeschlagen, falls Kommandos per Hand in den Tab „Table“ eingefügt werden müssen. Zudem sind die möglichen Eingaben auf der Webseite von jedem Tool gut dokumentiert. Zusätzlich kann der Quellcode auch nachträglich verändert werden, was das Tool auch für Tester interessant macht. Des Weiteren können die Tests sofort nach der Aufzeichnung erneut ausgeführt werden, so dass die korrekte Funktion der Tests auch für unerfahrene Benutzer leicht zu überprüfen ist.

- **Zielgruppe mit Programmierkenntnissen**

  Für Benutzer mit Programmierkenntnissen empfiehlt sich TestCafe, Selenium Webdriver, Appium zu verwenden. Hier wird es der persönlichen Entscheidung des Benutzers überlassen, da dies eine Frage der eigenen Interessen ist.

  Zu empfehlen ist es unbedingt, eine kostenlose Testversion anzufordern, um sich vor Beginn der offiziellen Tests mit dem Ablauf vertraut zu sein. Wie in dieser Arbeit zu sehen, bieten nicht alle Anbieter eine breite Palette an Skriptsprachen zur Auswahl, so dass die Tools, die die bevorzugte Technologie nicht anbieten, schnell abgelehnt

werden können. Da der Markt für Mac-Benutzer besonders klein ist, sollte die Anschaffung eines Virtualisierungsprogramms in Betracht gezogen werden.

- **Investition in Testwerkzeuge**

Die Einführung eines neuen Werkzeugs ist mit Kosten für Auswahl, Anschaffung und Wartung der Werkzeuge verbunden. Zusätzlich können Kosten für Hardware und Mitarbeiterschulungen hinzukommen. Abhängig von der Komplexität des Werkzeugs und Anzahl der auszustattenden Arbeitsplätze kann die Investition schnell im sechsstelligen Bereich liegen. Wichtig ist natürlich, wie bei jeder Investition, den Zeitrahmen für die Amortisation des neuen Testwerkzeugs zu definieren.

Bei Werkzeugen zur Automatisierung der Testausführung lässt es sich leicht abschätzen, wie viel Aufwand durch einen automatisierten Testlauf im Vergleich zur manuellen Ausführung eingespart werden kann. Der neue Aufwand für die Programmierung des Tests muss noch abgezogen werden, so dass nach nur einem automatisierten Testlauf die Kosten-Nutzen-Bilanz meist negativ ist.

Bei der Investition muss auch berücksichtigt werden, inwieweit sich die Testqualität durch den Einsatz des neuen Werkzeugs erhöht, was dazu führt, dass mehr Fehler gefunden und behoben werden. Die Kosten für Entwicklung, Support und Wartung werden dadurch gesenkt. Allerdings ist das Einsparpotenzial hier höher und daher interessanter (Vgl. Spillner & Linz. (2019) S. 223 f.).

Falls es sich bei der zu testenden Software um ein kleines Projekt handelt, können Open-Source-Tools verwendet werden, anstatt mehr Kosten für den Kauf eines lizenzierten Tools zu investieren. Wer nur ein sehr geringes Budget zur Verfügung hat, dem lassen sich die kostenlose Software TestCafe, Selenium Webdriver, Selenium IDE, Appium empfehlen.

9 Fazit

Ziel dieser Arbeit war es, einzelne Testwerkzeuge anhand verschiedener Kriterien im Bereich des Softwaretests miteinander zu vergleichen. Für den Vergleich wurden Testtools vom Internet ausgewählt und mittels einer Kino-Webseite getestet, die im Rahmen des Moduls „Webtechnologie“ während des Studiums programmiert wurde.

Es ist für einen Tester unerlässlich, einige wichtige Begriffe aus dem Bereich Softwaretests und ihre Definitionen zu kennen. Dazu zählt der Begriff Softwarequalität, weil die Qualität und ihre Sicherung auch zu den Aufgaben des Testers gehören. Ebenso muss ein Tester die Definition von Tests beherrschen, um die Arbeit mit Wissen erledigen zu können. Auch der Begriff des Fehlers, der unterschiedliche Definitionen haben kann, je nachdem worauf sich der Fehler bezieht, muss bekannt sein. Da ein Tester in einem Projekt nur eine bestimmte Zeit zum Testen hat, muss er in der Lage sein, die Dauer des Tests abzuschätzen. Darüber hinaus unterscheiden sich die wichtigsten Arten von Tests, die sich für die Durchführung eignen, in der Zielsetzung. Dazu gehören die Ziele und Methoden der funktionalen Tests, bei denen die Funktionalität des Systems, wie sie in einer Spezifikation beschrieben ist, geprüft wird. Umgekehrt gibt es auch die nicht funktionalen Tests, die sich mit Aspekten wie der Benutzerfreundlichkeit und der Akzeptanz durch den Benutzer befassen. Damit das Testen durchgeführt werden kann, werden Verfahren benötigt, um Testfälle zu erstellen bzw. eine Sicht auf das Testobjekt bekommen zu können. Dabei sind Verfahren wie das Blackbox-Verfahren, welches das Testen aufgrund von Spezifikationen durchführt, und das Whitebox-Verfahren, welches den Programmtext als Referenz für die Tests hernimmt, zu betrachten. Bei beiden Verfahren können verschiedene Ansichten des Testobjekts betrachtet werden, die sich stark unterscheiden und einen guten Überblick über das System geben.

Softwaretests sind ein wichtiger Teil des Lebenszyklus der Softwareentwicklung. Sie stellen sicher, dass die auf dem Markt bereitgestellte Software frei von Fehlern ist. Deswegen muss einen Tester die verschiedenen Software-Testing-Methoden sowie die Kategorien von Werkzeuge und deren Anwendungsgebiete unterscheiden können. Tester sollte sich immer der Verwendung dieser Werkzeuge im Entwicklungsprozess bewusst sein.

Während der Arbeit konnte festgestellt werden, dass einige Tools nur zum Testen von Webanwendungen geeignet sind und andere zum Testen von Web, Mobile, Desktop - Anwendungen verwendet werden können. Auch die meisten Tools waren „Capture/Replay Tools“, die auch von Benutzern ohne Programmierkenntnisse verwendet werden können.

Was einen guten Tester ausmacht, ist die gesammelte Erfahrung im Bereich Softwaretesten, denn nur durch viel Erfahrung kann ein Tester ein gutes Gefühl für das Auffinden von Fehlern in Software entwickeln. Eine weitere Voraussetzung für einen guten Tester ist das Mitbringen von vielen Programmierskills, weil die Skripte in unterschiedlichen Programmiersprachen erzeugt werden. Da ein Tester derjenige ist, der den von einem anderen verursachten Fehler an den Entwickler überbringt, muss er eine gewisse Sensibilität haben, um die Entwickler davon zu überzeugen, den von ihm gefundenen Fehler zu beheben.

Wegen der in dieser Arbeit behandelten Punkte ist der Softwaretest nicht mehr in der Softwareentwicklung wegzudenken. Als Softwaretester ist man verantwortlich für die Sicherstellung der Qualität des Produkts sowie der Dokumentation. Dadurch können sich die Entwickler auf die Programmierung, den Entwurf und die Korrektur von Fehlern im Programm konzentrieren. Sie müssen die Fehler nicht mehr selbst suchen, sondern sie bekommen sie durch die Tester vorgestellt. Deswegen muss die Zusammenarbeit und das gegenseitige Unterstützung zwischen ihnen gut funktionieren.

Diese Arbeit wäre für Tester und Entwickler nützlich, um zu wissen, welches Werkzeug für ein bestimmtes Projekt am besten geeignet ist, mehr Werkzeuge können verglichen werden und mehr Kriterien für den Vergleich herausstellen. Dies würde Softwaretestern die Möglichkeit geben, perfekte Tools für das Testen von Anwendungen auszuwählen. Dadurch wird mehr Zeit gespart und die Kosten werden reduziert, wenn das richtige Testwerkzeug ausgewählt wird. Daraus lässt sich die Schlussfolgerung ziehen, dass es nicht das eine perfekte Werkzeug für das Testen gibt, sondern für einen bestimmten Testzweck, können je nach Größe des Projekts Kompromisse geschlossen werden, um das beste Tool auszuwählen, das die geplanten Kosten für das Testen berücksichtigt und die Plattform der Anwendung und auch die Sprache, in der das Projekt entwickelt wird, ermöglicht.

Literaturverzeichnis

al, F. O. (2019). A Critical Analysis of Software Testing Tools. *Journal of Physics: Conference Series*, 1.

Axel Kalenborn, T. W. (2006). Java-basiertes automatisiertes Test-Framework. *WI – Innovatives Produkt*, 1&2.

Busics&Baumgartner&Seidl&Gwihs. (2015). *Basiswissen Testautomatisierung.* 2. aktualisierte und überbearbeitete Auflage, Heidelberg.

Cleff, T. (2010). *Basiswissen Testen von Software.* Herdecke/Witten 2.Auflage.

Franz, K. (2007, 2015). *Handbuch zum Testen von Web- und Mobile-Apps.* 2., aktualisierte und erweiterte Auflage, Berlin Heidelberg.

Gundecha&Avasarala. (2018). *Selenium WebDriver 3 Practical Guide.* Birmingham.

Hoffmann, D. W. (2008, 2013). *Software-Qualität.* 2., aktualisierte und korrigierte Auflage, Berlin Heidelberg .

Kleuker, S. (2013, 2019). *Qualitätssicherung durch Softwaretests.* 2., Erweiterte und aktualisierte Auflage, Wiesbaden.

Manfred Baumgartner, M. K. (kein Datum). *Agile Testing.* 2. Auflage, München.

Peter, L. (2009). *Software-Qualität.* 2.Auflage, Heidelberg.

Pilorget, L. (2012). *Testen von Informationssystemen.* Wiesbaden GmbH.

Shpakovskyi, D. (2020). *Modern Web Testing with TestCafe.* Birmingham.

Siteur, M. M. (2005). *Automate your testing! Sleep while you are working.* Amsterdam: Centraal Boekhuis.

Spillner&Linz. (2019). *Basiswissen Softwaretest.* 6., überbearbeitete und aktualisierte Auflage, Heidelberg.

Stefan, G. (2013). *Software-Test für Embedded Systems.* 1.Auflage, Heidelberg.

Thaller, G. E. (2002). *Software-Test.* 2., aktualisierte und erw. Auflage, Hannover.

Witte, F. (2016, 2019). *Testmanagement und Softwaretest.* 2., erweiterte Auflage, Wiesbaden.

Witte, F. (2020). *Strategie, Planung und Organisation von Testprozessen.* Wiesbaden.

Appium Dokumentation (Tom Christie) [Online]: http://appium.io/docs/en/about-appium/intro/ [Abgerufen am 28.07.2021]

Certified Tester Foundation Level Syllabus, Version 2018 V3.1, International Software Testing QualificationsBoard (ISTQB) (Klaus Olsen (chair), Meile Posthuma, Stephanie Ulrich) [Online]: ISTQB-CTFL_Syllabus_2018_V3.1.pdf [Abgerufen am 30.06.2021]

QF-Test Dokumentation (Quality First Software GmbH) [Online]: https://www.qfs.de/en/support/documentation.html [Abgerufen am 01.08.2021]

Ranorex Dokumentation (Ranorex GmbH) [Online]: https://www.ranorex.com/help/latest/ [Abgerufen am 19.08.2021]

Selenium Dokumentation (ThoughtWorks) [Online]: https://www.seleniumhq.org/ [Abgerufen am 28. 07.2021]

Software Testingtools (Unbekannt) [Online]: http://www.softwareqatest.com [Abgerufen am 28.07.2021]

TestCafe Dokumentation (DevExpress) [Online]: https://testcafe.io/documentation/402665/reference/test-api/testcontroller [Abgerufen am 20.07.2021]

TestComplete Dokumentation (SmartBear Software) [Online]: https://support.smartbear.com/testcomplete/docs/ [Abgerufen am 26.08.2021]

Unified Functional Testing Dokumentation (Micro Focus) [Online]: Welcome to UFT One (microfocus.com) [Abgerufen am 26.08.2021]

Eidesstattliche Versicherung

Hiermit versichere ich, an Eides statt, dass die an diese Versicherung angefügte schriftliche Ausarbeitung selbstständig und ohne jede unerlaubte Hilfe angefertigt wurde, dass sie noch keiner anderen Stelle zur Prüfung vorgelegen hat und dass sie weder ganz noch im Auszug veröffentlicht worden ist.

Die Stellen der Arbeit, die anderen Werken und Quellen (auch Internetquellen) dem Wortlaut oder dem Sinn nach entnommen sind, habe ich in jedem einzelnen Fall als Entlehnung mit exakter Quellenangabe kenntlich gemacht. Zudem versichere ich, dass ich keine anderen als die angegebenen und bei Zitaten kenntlich gemachten Quellen und Hilfsmittel benutzt habe.

Mir ist bekannt, dass diese Arbeit auch auf elektronischem Wege auf Einhaltung wissenschaftlicher Standards überprüft wird und im Falle eines Plagiats als Täuschungsversuch bewertet werden kann.

_____ _____

Ort, Datum Unterschrift

.