Freie Universität Berlin

**Bachelorarbeit am Institut für Informatik der Freien Universität Berlin**

# Bachelorarbeit

über das Thema

## Konzeption und Evaluation der Qualitätssicherung einer Multi-Plattform Publishing Suite mit Testautomatisierung


**Autor:** Patricia Jahnke
p.jahnke@fu-berlin.de

**Gutachter:** Prof. Dr. Lutz Prechelt

**Zweitgutachterin:** Prof. Dr. Margarita Esponda-Argüero


Berlin, 04. November 2015

# Eidesstattliche Erklärung

Ich versichere hiermit an Eides Statt, dass diese Arbeit von niemand anderem als meiner Person verfasst worden ist. Alle verwendeten Hilfsmittel wie Berichte, Bücher, Internetseiten oder ähnliches sind im Literaturverzeichnis angegeben, Zitate aus fremden Arbeiten sind als solche kenntlich gemacht. Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungskommission vorgelegt und auch nicht veröffentlicht.

Datum: .....

(Unterschrift)

# Kurzfassung

Diese Arbeit begleitet einen Teil des Entwicklungsprozesses des von der sprylab technologies GmbH entwickelten Produktes „Purple Publishing Suite“. Dies ist eine Produktionsumgebung, mit der digitale Publikationen erstellt werden können und die aus einer Desktopanwendung, einer Webanwendung und einer mobilen Anwendung besteht. Das Ziel dieser Arbeit war die Konzeption und Evaluation einer Testautomatisierung von Purple, wobei die Schwierigkeit war für alle drei Komponenten eine eigene Lösung zu finden.

Bei der Webanwendung – dem Purple Manager – wurden Oberflächentests mithilfe des Testwerkzeugs Selenium automatisiert. Dies war sehr zeitaufwendig aufgrund des hohen Funktionsumfangs der Webanwendung und weil noch ein paar kleinere Codeänderungen nötig waren. Selenium hat allerdings eine gute Elementidentifikation und ist dadurch gut wartbar, sodass weitere Arbeiten an der Testautomatisierung weniger zeitintensiv werden. Die Abdeckung der Testfälle ist hoch.

Auch die Testautomatisierung der mobilen Anwendung, des Purple Kiosks, basiert auf Oberflächentests, wofür MonkeyTalk benutzt wurde. Da sprylab genug Testgeräte besitzt, konnte die Automatisierung direkt in der Firma durchgeführt werden. Aufgrund der deutlich verschiedenen Displaygrößen und der damit verbundenen anderen Darstellung der Inhalte wurden unterschiedliche Tests für Smartphones und Tablets entwickelt. Die größten Probleme bei der Entwicklung der Tests traten bei Wartezeiten und den nicht eindeutigen IDs der Elemente auf. Während der Tests entstanden gewollte Wartezeiten, die je nach Gerät und Internetverbindung unterschiedlich lang waren, wofür aber auch eine einheitliche Lösung gefunden werden musste. Insgesamt war die Abdeckung auch hier hoch und die Tests ließen sich aufgrund des deutlich geringeren Funktionsumfangs relativ schnell umsetzen.

Der Purple Composer ist die Mac-Desktopanwendung, bei der die Exportfunktion der Datei getestet wurde, die innerhalb der Anwendung zusammengestellt wurde. Hierbei werden bereitgestellte Dateien über ein Skript importiert, exportiert und anschließend mit einer Referenzdatei verglichen. Hier mussten zuerst gewollte Unterschiede (beispielsweise die Versionsnummer) gefunden und aus dem Vergleich herausgenommen werden. Da hierbei nur der Datenstrom und nicht die Oberfläche getestet wird, war dies die schnellste Automatisierung.

# Abstract

This thesis accompanied a part of the development process of the product Purple Publishing Suite developed by the sprylab technologies GmbH. This is a production environment for creating digital publications and consists of a desktop application, a web application and a mobile application. The aim of this thesis was creating a design and later evaluating a test automation of Purple, the difficulty was to find an own solution for each of the three components.

In the web application – the Purple Manager – surface tests were automated using the testing tool Selenium. This was very time-consuming due to the high functionality of the web application and because a few minor code changes were necessary. However, Selenium has a good element identification and is therefore maintainable, so further work on the test automation will be less time-consuming. The coverage of the test cases is high.

The test automation of the mobile application, the Purple Kiosk, is based on surface tests using MonkeyTalk. Since sprylab has enough test devices, automation could be performed directly in the company. Due to the significantly different display sizes and the associated varied representation, different tests for smartphones and tablets have been developed. The biggest problems in the development of the tests occurred in waiting times and the non-unique IDs of the elements. During the tests, intentional delays arose, which were of different length depending on the device and the internet connection, for which also a uniform solution had to be found. Overall, the coverage was also high here and the tests could be implemented relatively quickly due to the significantly lower functional scope.

The Purple Composer is the Mac desktop application, where the export function of the file has been tested, which was compiled within the application. The hosted files are imported, exported and then compared to a reference file via a script. First, planned differences (for example, the version number) had to be found and taken out of the comparison. Since only the data stream and not the surface was being tested, this was the fastest automation.

# Urheberrecht

# Inhaltsverzeichnis

# 1 Einleitung

Ziel dieser Bachelorarbeit ist die Entwicklung eines Konzepts zur Testautomatisierung für das von der sprylab technologies GmbH entwickelte Produkt „Purple Publishing Suite“. Zu Beginn dieser Arbeit gab es bereits eine lauffähige, aber vereinfachte Version der Purple Publishing Suite. Der weitere Entwicklungsprozess wurde vom Autor begleitet, um die Arbeit darin zu integrieren.

Die Purple Publishing Suite ist eine Produktionsumgebung, mit der digitale Publikationen erstellt werden können. Insgesamt besteht das Produkt aus drei Anwendungen, die jeweils für einen Teil der Umsetzung zuständig sind. Das Ziel der Anwendung ist, aus statischen Print-Medien aktive, animierte Elemente zu schaffen, um eine interaktive Nutzung von digitalen Publikationen zu ermöglichen. Die Purple Publishing Suite bietet hierfür sowohl eine speziell für diesen Zweck entwickelte Designplattform (Mac-Anwendung), einen umfangreichen Verwaltungsbereich (Webanwendung) und eine mobile Anwendung für den Verkauf und die Nutzung der digitalen Publikationen.

Die sprylab technologies GmbH ist eine 2007 gegründete Softwarefirma, die sich auf die Bereiche der mobilen Anwendungen und der Webanwendungen spezialisiert hat. Neben der Umsetzung von Projekten nach den Vorstellungen der Kunden entwickelt sprylab auch Produkte, mit denen Unternehmen in den Bereichen Digital Publishing, Mobile Strategy und Location-Based Edutainment Anwendungen selbst erzeugen und aktualisieren können. Insgesamt besteht die Firma aktuell aus ca. 45 Mitarbeitern an zwei Standorten, darunter auch Freelancer und Studenten, wobei die Anzahl der Mitarbeiter ständig wächst.

## 1.1 Problemstellung

Das Problem dieser Arbeit bestand im Entwurf einer Qualitätssicherung (QS) mit Testautomatisierung für drei unterschiedliche Plattformen, die zu umfangreich für manuelle Tests sind. Problematisch war das, weil zum einen für jede Plattform eine eigene Lösung gefunden werden musste und zum anderen jede Komponente so umfangreich war, dass vorher abgewägt werden musste, welcher Teil automatisiert wird. Es mussten für jede Plattform mehrere Fragen beantwortet werden:

1. Was soll (nicht) automatisiert werden?

   Es muss entschieden werden, welche Teile des Systems nicht automatisiert werden sollen beziehungsweise was nicht automatisiert werden kann. Außerdem muss entschieden werden, welche automatisierten Systemteile die meisten Vorteile bringen.

2. Wie soll automatisiert werden?

   Die in diesem Fall Beste von vielen möglichen Methoden zur Qualitätssicherung muss herausgefunden werden.

3. Womit soll automatisiert werden?

   Die Auswahl eines geeigneten Testwerkzeugs muss getroffen werden.

Nach Beantwortung dieser Fragen konnte mit der Umsetzung der Testautomatisierung begonnen werden.

## 1.2 Aufbau der Arbeit

Im zweiten Kapitel gibt es eine Einführung in die Softwarekomponenten der Purple Publishing Suite. Kapitel 3 behandelt einige Grundlagen zur Testautomatisierung, anschließend werden die für diese Arbeit aufgestellten Kriterien für die Auswahl eines geeigneten Testwerkzeugs vorgestellt. Die darauffolgenden drei Kapitel behandeln jeweils eine Komponente mit einer Einleitung, den möglichen Vorgehensweisen, der Umsetzung und der Bewertung der Umsetzung. Zuerst wird die Webanwendung vorgestellt, danach die mobile Anwendung und zum Schluss die Desktopanwendung. Den Abschluss dieser Arbeit stellt das siebte Kapitel mit dem Thema Fazit und Ausblick dar.

# 2 Softwarekomponenten von Purple

Für das Verständnis der Purple Publishing Suite ist es wichtig, zwischen Kunden und Konsumenten zu unterscheiden. Kunden sind diejenigen, die Purple zum Erstellen von digitalen Publikationen nutzen. Konsumenten sind diejenigen, die sich die vom Kunden erstellte Anwendung aus dem App Store[1] laden und die Ausgaben kaufen.

Die Purple Publishing Suite besteht aus insgesamt drei Teilen: Composer, Manager und Kiosk. Der erste Schritt ist die Benutzung der Mac-Anwendung „Purple Composer“, was eine professionelle Designplattform ist, die speziell für digitales Publizieren entwickelt wurde. Nachdem man seine Photoshop, Indesign oder PDF Dateien im Composer importiert und bearbeitet oder aber direkt eigene Inhalte erstellt hat, wird das Ergebnis als pkar-Datei exportiert und steht somit der nächsten Anwendung der Purple Publishing Suite zur Verfügung.

In der Webanwendung „Purple Manager“ kann man die erstellte Datei importieren und sich um die Verwaltung kümmern. Es werden unter anderem Publikationen und Ausgaben angelegt, Anwendungen generiert und Einstellungen festgelegt. Die mobile Anwendung wird hier geplant, vom Namen bis zu detaillierten Layouteinstellungen.

Der „Purple Kiosk“ ist die mobile Anwendung, die Kunden im Manager erstellen und anschließend daraus herunterladen können. In dem Kiosk befinden sich alle Ausgaben, die im Manager zu dieser Anwendung hinzugefügt wurden. Der Kunde, der die mobile Anwendung erstellt hat, kann sich diese aus dem Manager herunterladen und beispielsweise im Google Play Store zur Verfügung stellen. Der Konsument lädt sich die Anwendung dann aus dem entsprechenden App Store herunter und sieht die enthaltenen Ausgaben. Interessiert er sich für eine, kann er sie – je nach Einstellung – kaufen oder herunterladen (es wird eine Verbindung zum Purple Manager aufgebaut und die Ausgabe von dort auf das Gerät geladen). Der Konsument hat auch die Möglichkeit ein Abonnement abzuschließen.

Die Testautomatisierung dieser Arbeit betrachtet die drei Komponenten als eigenständig und unabhängig voneinander, die Tests beziehen sich also ausschließliche auf jeweils eine Plattform.

[1] Hiermit ist nicht der gleichnamige App Store von Apple gemeint, sondern allgemein ein Marktplatz für mobile Anwendungen.

# 3 Testautomatisierung

Das Ziel von Testautomatisierung ist eine effiziente und langfristig ökonomische Durchführung [Vig05a]. Dabei muss man beachten, dass die Automatisierung an sich auch ein Programm ist und somit Fehler enthalten könnte. Damit ein Fehler nicht gleich viele Tests stoppt, sollten diese möglichst unabhängig voneinander laufen können. Dafür stellt man vor jedem Test die notwendige Testumgebung her und baut diese hinterher wieder ab [Vig05b].

Für die Skripte nennt Vigenschow fünf Grundtechniken [Vig05c].

**Lineare Skripte:**

Ein solches Skript erhält man nach der Nutzung eines Aufnahme und Wiedergabe Werkzeugs. Es ist je nach Werkzeug unterschiedlich weit heruntergebrochen (teilweise bis auf einzelne Tastenanschläge und Mausklicks). Der Testfall verläuft linear, sodass man keine Programmierkenntnisse benötigt, allerdings ist er auch fehleranfällig gegenüber Änderungen.

**Strukturierte Skripte:**

Diese Skripte enthalten Sequenzen, Selektionen und Iterationen und können somit flexibler auf unvorhergesehene Ablaufänderungen reagieren. Sie sind robuster als die linearen Skripte, besser wartbar und kürzer, allerdings benötigt der Ersteller des Skriptes Programmierkenntnisse.

**Verteilte Skripte:**

Bei verteilten Skripten wird die Funktionalität auf verschiedene Dateien separiert, was eine Wiederverwendung der Skripte ermöglicht. Dadurch reduziert sich zwar der Wartungsaufwand, man hat aber eine komplexe Datenstruktur.

**Datengetriebene Skripte:**

Hier werden die Testdaten von den Abläufen innerhalb eines Skriptes getrennt. Das bedeutet einerseits einen erhöhten Programmieraufwand, andererseits kann man die gleichen Tests leicht mit unterschiedlichen Daten ausführen.

**Schlüsselwortgetriebene Skripte:**

In diesen Skripten werden datengetriebene Tests über Schlüsselwörter gesteuert, sodass man die Testfälle unabhängig von Programmierdetails behandeln kann. Die Anzahl der Testfälle wächst somit nur noch mit der Funktionalität der zu testenden Software, was den Wartungsaufwand verringert. Der initiale Programmieraufwand dagegen ist allerdings –

im Vergleich zu den anderen Skriptarten – maximal, diese Automatisierung muss wie ein richtiges Softwareprojekt behandelt werden.

## 3.1 Kriterien für Werkzeuge zur Testautomatisierung

Für diese Arbeit wurden einige Kriterien für die Auswahl der Werkzeuge zur Testautomatisierung zusammengestellt. Diese Kriterien und deren Relevanz wurden vom Autor dieser Arbeit im Interesse von sprylab in Bezug auf die Testautomatisierung der Purple Publishing Suite entworfen und sind somit nicht allgemein gültig.

Das wichtigste Kriterium für sprylab ist, dass die Software Open Source ist, damit das Projekt nicht von vorne begonnen werden muss, sollte das zum Testwerkzeug gehörige Unternehmen die Entwicklung einstellen. Zum anderen kann das Testwerkzeug beliebig erweitert und angepasst werden, wenn der Quellcode zur Verfügung steht. Ebenfalls wichtig ist ein aktuelles Releasedatum, da man daran erkennen kann, wie aktiv noch an der Software gearbeitet wird. Auch ein guter Support kann hilfreich sein, ist aber weniger wichtig als die bereits genannten Kriterien.

Beim Erstellen des Tests wird die Handhabung beziehungsweise Einarbeitung noch als relevant erachtet, damit nicht Zeit in das Werkzeug an sich investiert werden muss, sondern man möglichst effizient an den Tests arbeiten kann.

Der nächste Bereich ist der Quellcode des Tests. Hier sind die Darstellungsform und die verfügbaren Programmiersprachen relevant, wichtiger ist aber die Möglichkeit selbst programmieren und somit den Code erweitern zu können.

Beim Protokoll der Testausführung ist die Darstellungsform interessant und auch die Darstellung der Oberfläche des gesamten Testwerkzeugs, damit auch diejenigen schnell Informationen daraus extrahieren können, die nicht direkt an der Testautomatisierung beteiligt waren.

| | **Kriterium** | **Relevanz** |
|---|---|---|
| **Produkt** | Release | ++ |
| | Support | + |
| | Open Source | +++ |
| **Testvorgang** | Handhabung/Einarbeitung | ++ |
| **Quellcode** | Darstellungsform | + |
| | Programmiersprachen | + |
| | Eigene Programmierung | ++ |
| **Protokoll** | Darstellungsform | + |
| **Oberfläche** | Darstellungsform | + |

Tabelle 1: Kriterien für die Testwerkzeuge

Alle genannten Kriterien und ihre Relevanz wurden für eine bessere Übersichtlichkeit in Tabelle 1 zusammengefasst.

# 4 Purple Manager

Der Purple Manager ist eine Webanwendung, über die die Publikationen mit ihren Ausgaben, die Anwendungen und diverse Einstellungsmöglichkeiten verwaltet werden können.

Der erste Schritt des Nutzers im Purple Manager ist die Registrierung (siehe Abb. 2). Hat sich der Nutzer angemeldet, sieht er unter anderem die folgenden Menüpunkte oben auf der Seite: Apps[2], Publikationen und Einstellungen.

Unter dem Menüpunkt „Apps“ befinden sich alle bereits angelegten Anwendungen, sowohl die bereits generierten als auch die nicht generierten (siehe Abb. 3). Durch Klicken auf den Namen einer Anwendung gelangt man auf die entsprechende Seite mit Details und weiteren Einstellungsmöglichkeiten (siehe Abb. 6).

Eine Publikation ist der Name einer Zeitschrift (zum Beispiel „Die Zeitschrift“). In jeder Publikation können mehrere Ausgaben hinzugefügt werden (zum Beispiel „Die Zeitschrift“, Ausgabe 01/2015)(siehe Abb. 4). Eine Ausgabe bekommt ihren Inhalt, indem man ihr die im Composer erstellte pkar-Datei zuweist (siehe Abb. 7). Jede Publikation kann einer oder mehreren Anwendungen zugeordnet werden, in denen die Ausgaben der Publikation dann erhältlich sind. Dieser Menüpunkt ist analog zu den Apps, er beinhaltet eine Liste der Publikationen mit Verlinkungen zu den entsprechenden Detail-Seiten und den Anwendungen, in denen die Publikation verwendet wird.

„Einstellungen“ hat diverse Unterpunkte, unter anderem Zahlungsarten und Umsatzsteuer. Bei den Zahlungsarten kann man ebendiese hinzufügen und bearbeiten, wobei man mehrere Variablen hat wie beispielsweise Typ (Abonnement oder pro Veröffentlichung), Preis und Periode (Tag, Woche, Monat, ...). Der Punkt Umsatzsteuer enthält eine Liste aller Länder, denen jeweils eine Umsatzsteuer zugeordnet werden kann.

Diese Seiten beinhalten überwiegend eine klickbare Fläche zum Erstellen eines neuen Datensatzes per Formular sowie eine Tabelle aller Datensätze. Ab sechs Einträgen erscheint unter der Tabelle eine Seitennavigation, da die Tabelle dann aus mehreren Seiten besteht (fünf Einträge pro Seite). Neben dem Primärschlüssel des Eintrages sind in der Tabelle weitere ausgewählte Eigenschaften zu sehen, nach denen die Einträge teilweise auch auf- beziehungsweise absteigend sortiert werden können.

[2] In diesem Kontext ist eine App eine Anwendung für ein mobiles Gerät. Da diese Anwendung auch im Purple Manager als App bezeichnet wird, wird die umgangssprachliche Bezeichnung auch in dieser Arbeit verwendet.

## 4.1 Mögliche Vorgehensweisen

In diesem Kapitel geht es zuerst um die möglichen Testarten und anschließend werden neun Testwerkzeuge kurz beschrieben und ein Werkzeug ausgewählt.

## 4.2 Testarten

An dieser Stelle muss abgewogen werden, welche Art Test durchgeführt werden soll. Man kann grob zwischen White-Box- und Black-Box-Testverfahren unterscheiden: Bei White-Box-Tests bezieht man sich auf den Code und prüft ihn gezielt auf Schwachstellen. Der Black-Box-Test dagegen kennt den Code nicht, hier bezieht man sich nur auf die Anforderungen, nicht auf die Implementierung.

Ein Beispiel für einen White-Box-Test ist der Modultest (Unittest). Hier werden die einzelnen Module einer Anwendung getestet, also kleine, autarke Teile des Systems. Der Modultest wird in der Regel vom Entwickler selbst schon während des Entwicklungsprozesses durchgeführt, ist also eines der ersten Tests, die gemacht werden [Vig05d].

Beim Black-Box-Test kann der Oberflächentest als Beispiel genannt werden. Die Funktionen werden aus Sicht des Benutzers getestet, sodass ein solcher Test erst in den späteren Zügen der Entwicklung, wenn die Oberfläche nicht mehr viel verändert wird, sinnvoll ist [Vig05e].

Für den Purple Manager gibt es zwar Unittests, die von den Entwicklern geschrieben wurden und auch stetig weiterentwickelt werden. Diese beschränken sich aber auf die Business Logik im Backend-Bereich, für das Frontend gibt es also keine White-Box-Tests, die automatisiert werden könnten. Was also gemacht werden kann, ist eine Überprüfung der Oberfläche, um sicherzustellen, dass auch nach kleineren Verbesserungen der Anwendung bei der Benutzung keine Fehler auftreten.

Wenn man regelmäßig Oberflächentests durchführen möchte, besteht die Gefahr, dass der Tester durch die monotone Arbeit schnell gelangweilt wird und Testfälle übersieht, sodass keine gleichbleibende Qualität der Tests gegeben wird. Daher ist es sinnvoll, diese Tests zu automatisieren. So können sie beliebig oft und mit konstanter Präzision ausgeführt werden, außerdem bekommen die Entwickler schnelleres Feedback. Allerdings muss man beachten, dass man bei der Erstellung der Tests einmalig einen hohen Aufwand hat und dass die Tests eventuell bei jedem Update der Anwendung neu angepasst werden müssen.

Der Vorteil von manuellen Oberflächentests ist, dass der Tester, der wie ein Endbenutzer agiert, auch Anforderungsfehler, Sonderfälle und Layoutfehler feststellen kann. Da ein Testwerkzeug nur unter großem Aufwand in der Lage ist, beispielsweise das Aussehen der

Webseite oder die Erreichbarkeit einer Klickfläche zu beurteilen, ist es trotzdem wichtig, weiterhin regelmäßige manuelle Tests zu machen.

Im nächsten Kapitel werden einige Testwerkzeuge vorgestellt, die Oberflächentests automatisieren und für den Zweck dieser Arbeit in Frage kommen könnten.

### 4.2.1 Testwerkzeuge für Webanwendungen

Die in diesem Kapitel genannten Werkzeuge wurden zunächst ohne Beachtung der in Kapitel 3.1 festgelegten Kriterien ausgewählt, um einen Einblick in die aktuell verfügbaren Werkzeuge zu bekommen. Nach einer kurzen Beschreibung zu jedem Testwerkzeug folgt am Ende dieses Kapitels die Auswahl.

Die Grundanforderung dieser Arbeit an die im Nachfolgenden vorgestellten Testwerkzeuge ist das folgende Funktionsprinzip: Man nimmt einen Testfall auf, indem man die gewünschten Aktionen einmal manuell durchführt (beispielsweise den Registrierungsprozess). Das Testwerkzeug speichert diese Aktionen, sodass sie beliebig oft abgespielt werden können. Da bei sprylab die Qualitätssicherungsabteilung – also nicht die Entwickler – das Automatisieren übernehmen soll, ist es außerdem sehr wichtig, dass die Tests ohne Programmierkenntnisse erstellt werden können.

Das **Telerik Test Studio** [Telb] ist mit 2499$ das teuerste der hier vorgestellten Testwerkzeuge. Die Handhabung soll intuitiv und einfach sein, außerdem wird die Voraussetzung, dass alles ohne Programmerkenntnisse möglich sein muss, erfüllt. Positiv fällt auf, dass das Testwerkzeug auch eine Umgebung für manuelle Tests bietet, sodass man sowohl die automatisierten als auch die manuellen Tests an einem Ort hat, was ein organisiertes und produktives Arbeiten ermöglichen soll. Es wird auch eine Plattform für die Zusammenarbeit des QS-Teams untereinander und mit den Entwicklern geboten, sodass sowohl die Entwickler als auch die Qualitätssicherung in ihrer gewohnten Umgebung bleiben können.

Sehr nützlich ist außerdem, dass die einzelnen Testfälle frei zu Testsuiten zusammenstellbar und auch mehrfach verwendbar sind, sodass häufig verwendete Aktionen wie zum Beispiel der Login nur einmal erstellt werden müssen. Zum Ausführen der Tests steht ein Scheduler zur Verfügung. Während der Testausführung wird von jedem Schritt ein Bildschirmfoto aufgenommen, die anschließend zusammen mit optionalen Anmerkungen als Dokumentation exportiert werden können. Telerik bietet einen mit 430 Videos in 5 Jahren sehr aktiven YouTube-Kanal an, auf dem man sich neben generellen Einführungen in das Testwerkzeug Tutorials zu verschiedenen Themen sowie andere informative Videos ansehen kann. Dies ist eine gute Anlaufstelle für Fragen, um sofort Antworten zu bekommen ohne auf den Support warten zu müssen.

Das Testwerkzeug **TestingWhiz** [Inf] ist genau wie die anderen kostenpflichtigen Werkzeuge nicht Open Source, ein Preis ist allerdings nur auf Anfrage zu erhalten. TestingWhiz fällt durch seine vielfältigen Aufnahmetechniken auf: schlüsselwort- oder datengetrieben, objektbasiert oder mit Excel und Java Skripting. Mithilfe von Excel-Tabellen kann man dynamische Testdaten mit mehreren Eingabewerten nutzen und es können sogar Captchas automatisiert werden. Auch bei diesem Testwerkzeug kann man Testfälle wiederverwenden, indem man sie gruppiert und als Methoden speichert. Die Funktion zum Aufnehmen von Bildschirmfotos und der Scheduler zum Planen der Testausführung fallen ebenfalls positiv auf.

**Ranorex** [Rana] folgt mit 1990 € direkt hinter Telerik auf der Preisliste. Neben den Bildschirmfotos, die bei fehlgeschlagenen Tests aufgenommen werden, ist die marktführende GUI-Objekterkennung die einzige Eigenschaft, die bei diesem Testwerkzeug heraussticht.

Das für Webautomatisierung geeignete Testwerkzeug im Paket von **TestComplete** [Smaa] des Unternehmens Smartbear kostet 1778 €. Es zeichnet sich durch mehrere positive Eigenschaften aus: Zum einen arbeitet TestComplete auf Objektebene, weshalb die Tests auch nach GUI-Änderungen noch funktionieren. Des Weiteren werden Checkpoints bereitgestellt beziehungsweise können selber erstellt werden (zum Beispiel ein Bildvergleich Pixel für Pixel) und Bildschirmfotos sowie Informationen über Objekte und Eigenschaften werden gespeichert. Der Bericht enthält detaillierte Informationen und kann um Nachrichten, Bilder, Dateien etc. erweitert werden. Auch bei diesem Testwerkzeug gibt es einen Scheduler für Regressionstests, die auch parallel ausgeführt werden können, außerdem kann eine externe Datenquelle für die Tests genutzt werden. Smartbear hat seit sechs Jahren einen YouTube-Kanal, auf dem zurzeit 241 Videos veröffentlicht wurden, unter anderem über TestComplete.

Auf der Webseite von **UFT** (ehemals QTP) [HP] des Herstellers HP gibt es nur wenige Informationen über das Testwerkzeug. Die einzige herausragende Eigenschaft ist die Wiederverwendbarkeit der Testkomponenten.

**Selenium** [B$^+$] ist ein kostenloses Open Source Testwerkzeug, das mit vielen Programmiersprachen und Frameworks arbeiten kann. Die IDE ist für Anfänger geeignet und so simpel, dass man die ersten Tests in sehr kurzer Zeit erstellen kann, vor allem, da es eine Autovervollständigung für die häufig genutzten Befehle gibt. Die erstellten Tests sind in vielen Formaten speicherbar.

**Canoo WebTest** [Canb] ist ebenfalls Open Source und kostenlos und einfach zu verstehen und zu benutzen. Es ist leicht erweiterbar und die Tests können in Hintergrund laufen. Der Nachteil ist, dass das Testwerkzeug nicht mit einem richtigen Browser, sondern nur

mit einer Simulation arbeitet.

**Watir** [Wata] ist ein simples, flexibles und kostenloses Open Source Testwerkzeug. Man kann Datenbanken verknüpfen und Dateien lesen, außerdem steht eine aktive und wachsende Community dahinter. Allerdings funktioniert das Werkzeug nur auf Windows und besitzt keine Aufnahme-Funktion.

**SahiPro** [Sof] ist mit 695$ das günstigste Testwerkzeug, das nicht Open Source ist. Der größte Vorteil dieses Werkzeugs ist, dass es keine wait-Statements benötigt, da diese vom Testwerkzeug selbst gemacht werden. Außerdem ist eine parallele und verteilte Ausführung der Tests möglich.

Tabelle 2 zeigt eine Übersicht aller in diesem Abschnitt genannten Testwerkzeuge inklusive Informationen zu den in Kapitel 3.1 festgelegten Kriterien.

Da die für sprylab wichtigste Eigenschaft eines Testwerkzeugs ist, dass es Open Source sein muss, können alle proprietären Testwerkzeuge ausgeschlossen werden. Infrage kommen also nur noch Selenium, Canoo WebTest und Watir. Auch hier kann leicht nach dem Ausschlussverfahren vorgegangen werden, da Watir nicht die vorausgesetzte Wiedergabe-Funktion besitzt und Canoo WebTest nicht mit echten Browsern arbeitet.

Übrig bleibt also Selenium, das mit seiner Einfachheit und einer großen Community, die an der Weiterentwicklung arbeitet und bei Problemen helfen kann, überzeugt. Der letzte Release der Selenium IDE war diesen Monat (Stand Oktober 2015), sodass alle Kriterien zum Thema Produkt erfüllt sind. Wie bereits erwähnt, ist die Handhabung einfach, sodass die Einarbeitungszeit auch dank der Dokumentation kurz ist.

Beim Quellcode kann man zwischen zwei verschiedenen Darstellungen wählen: entweder als Tabelle oder in der Struktur der gewählten Programmiersprache. Die Tabellenansicht ist klar strukturiert, da es sich um ein lineares Skript handelt, und Kommentare werden durch die IDE farblich hervorgehoben. Wählt man die andere Ansicht, bekommt man zwar die klare Struktur von beispielsweise HTML-Code, allerdings ohne Syntaxhighlighting. In beiden Darstellungsformen hat man die Möglichkeit, manuell Befehle einzugeben.

Das Protokoll wird im Reiter Log angezeigt. Pro Codezeile gibt es die Information, ob die Zeile ausgeführt wurde oder ob ein Fehler aufgetreten ist. Allerdings gibt es für den Tester bis auf eine knappe Meldung keine weiteren nützlichen Informationen und auch keine Verlinkungen zu Fehlermeldungen.

Die Oberfläche der IDE ist sehr übersichtlich und leicht verständlich, da die verwendeten Symbole eindeutig und die Struktur klar sind. Somit erfüllt Selenium die bereits genannten

[3] [Tela] [PR] [Ranb] [Smab] [Ban15] [Sel] [Cana] [Watb] [Sah]

| | | Telerik | TestingWhiz | Ranorex | TestComplete | UFT | Selenium | Canoo | Watir | SahiPro |
|---|---|---|---|---|---|---|---|---|---|---|
| **Produkt** | Release | 10/2015 | 08/2015 | 10/2015 | 09/2015 | 07/2015 | 10/2015 | 12/2014 | 04/2013 | 06/2015 |
| | Support | + | + | + | + | + | + | O | + | + |
| | Open Source | Nein | Nein | Nein | Nein | Nein | Ja | Ja | Ja | Nein |
| **Testvorgang** | Hand-habung | + | + | + | + | + | + | + | O | O |
| **Quellcode** | Darstellung | + | + | + | + | + | + | ? | O | - |
| | Progr.-sprachen | + | ? | O | + | ? | + | ? | - | - |
| | Eigene Programme | Ja | Ja | Ja | Ja | Ja | Ja | Ja | Ja | Ja |
| **Protokoll** | Darstellung | + | + | + | + | ? | - | O | ? | + |
| **Oberfläche** | Darstellung | + | + | + | + | + | + | O | ? | O |

Tabelle 2: Kriterienübersicht für alle Testwerkzeuge des Purple Managers[3]

Kriterien und eignet sich zur Automatisierung des Purple Managers.

## 4.3 Umsetzung

In diesem Kapitel geht es um die Planung der Testfälle mithilfe eines webbasierten Testmanagement-Systems[4] und das Erstellen der Tests.

### 4.3.1 Testfallkatalog im Testmanagement-System

Der erste Schritt zur Testautomatisierung war das Anlegen eines Testfallkatalogs im Testmanagement-System. Die Testfälle wurden dafür zunächst in Gruppen unterteilt, die sich am Aufbau der Webseite des Managers orientieren. Jeder Testfall wurde dann möglichst genau spezifiziert, also welches Verhalten welchen Effekt erzielen soll.

Der erste Bereich ist ein allgemeiner, in dem es um die Login-Seite, die Registrierung und diverse Verlinkungen wie zum Beispiel zum Impressum, Datenschutz, Social Media und zur Homepage von Purple geht.

Die zweite Testfallgruppe ist die umfangreichste, hier geht es um den Bereich der Apps. Neben dem Erstellen von Apps und dem Testen der Tabellenfunktionen (Anordnung ändern, Suche, Navigation) fallen in diesen Bereich auch die vielfältigen Einstellungsmöglichkeiten für das Aussehen der App. Auch das Generieren der Anwendung wird hier getestet.

In den Bereichen Publikationen, Nutzer, Coupons und Einstellungen geht es hauptsächlich darum, ein neues Objekt zu erstellen, gegebenenfalls zu editieren und die Tabellenfunktionen zu testen.

Die Testfallgruppen Berichte, Downloads und Profil sind relativ simpel, sie enthalten viele Verlinkungen und nur wenige Einstellungsmöglichkeiten.

Der Bezahlungsprozess ist wieder eine größere und vor allem sehr wichtige Testfallgruppe aus Sicht der Kunden, da diese wollen, dass der Bezahlvorgang reibungslos funktioniert. Dementsprechend werden hier alle einzelnen Schritte sowie die verschiedenen Bezahlungsmodelle ausführlich beschrieben.

### 4.3.2 Testautomatisierung

Als erstes wurde eine Zeitplanung gemacht, um einschätzen zu können, wie lange die Automatisierung dauern wird. Wegen der gut geordneten Struktur und der ausführlichen Beschreibungen im Testfallkatalog konnte dies schnell umgesetzt werden. Anschließend wurde damit begonnen, die Testfälle gruppenweise mit der Selenium IDE zu automatisieren, indem sie zuerst aufgenommen und dann manuell angepasst wurden. Sollten dabei

[4] Bei sprylab wird das Testmanagement-System TestLink verwendet [Tes].

noch Details gefunden werden, die noch nicht im Testfallkatalog standen, wurden diese umgehend ergänzt. Teilweise mussten für die Automatisierung kleinere Codeänderungen vorgenommen werden. Besonders Element-IDs mussten häufig angepasst werden, da nicht alle statisch waren, sondern zur Laufzeit automatisch generiert wurden. Da Selenium diese IDs jedoch nutzt, um die Elemente zu finden, mussten sie einen festen Wert haben.

Der nächste Schritt war das Importieren in WebInLoop, einem von sprylab entwickelten Testframework. Die Selenium Tests erlaubten nur eine Ausführung in Firefox, außerdem mussten die Testsuits der IDE von Hand ausgeführt werden. WebInLoop ermöglicht eine noch höhere Automatisierungsstufe, indem es die Testfälle in beliebigen Webbrowsern automatisch ausführt und dazu Testberichte und Bildschirmfotos erstellt. Außerdem erleichtert WebInLoop die Integration in Continuous Integration-Umgebungen, was eine automatische Ausführung der Tests nach einem festgelegten Zeitplan erlaubt.

## 4.4 Bewertung der Umsetzung

Das folgende Kapitel behandelt die Bewertung der Umsetzung und umfasst eine Abschätzung der Aufwände, des Nutzen und der Abdeckung.

### 4.4.1 Aufwände

Insgesamt wurde ein Zeitaufwand von etwa 190 Stunden für das Abarbeiten des gesamten Testfallkatalogs (zu diesem Zeitpunkt etwa 55 Testfälle) mit der IDE und die Übertragung in WebInLoop geschätzt. In dieser Zeit konnte zwar ein Großteil der Arbeit erledigt werden und es gab dann auch zu jedem Testfall einen Test, allerdings gab es noch viele Fehlschläge, die fast ausschließlich mit Problemen bei wait-Anweisungen[5] zu tun hatten. Es kamen somit noch etwa 50 weitere Stunden hinzu, in denen diese Fehler behoben werden mussten. Das Erstellen der Tests war also ein großer Aufwand, auch am Code mussten einige Stellen geändert werden, um die in Kapitel 4.3.2 erwähnten Element-IDs anzupassen.

Man muss jedoch auch den langfristigen Aufwand bedenken, da die Purple Publishing Suite auch in Zukunft noch weiterentwickelt werden soll. Die Zeit zum Erstellen neuer Testfälle ist gering und auch der Wartungsaufwand ist nicht hoch. Wenn beim Entwickeln

[5] An einigen Stellen kommt es aufgrund von hohem Datenverkehr zu längeren Ladezeiten, denen man mit wait-Anweisungen begegnen muss. Dabei muss man sich genau überlegen, auf welches Element man warten möchte. Wartet man beispielsweise auf das Verschwinden des Ladebildschirms (Befehl waitForNotText), muss man bedenken, dass es einen Moment dauert, bis der Ladebildschirm erscheint. Ist Selenium zu schnell, wird der Befehl erfolgreich abgearbeitet bevor der Ladebildschirm überhaupt erscheint. Diese Zeiten sind abhängig von der Geschwindigkeit des ausführenden Computers (bestimmt die Geschwindigkeit der Befehlsabarbeitung) und der Internetverbindung (bestimmt die Geschwindigkeit des Purple Managers).

neuer Funktionen in Zukunft beispielsweise auf das Setzen einer Objekt-ID geachtet wird, sollte auch am Code nichts mehr geändert werden müssen.

Im Vergleich dazu steht der Aufwand bei manuellen Tests, der bei jedem erneuten Testdurchlauf immer ungefähr gleich hoch ist. Die Seleniumtests benötigen für einen gesamten Testdurchlauf etwa eine Stunde. Um die Dauer der manuellen Tests abschätzen zu können, wurden stichpunktartig einige Tests ausgesucht und manuell getestet. Die dabei gemessene Zeit wurde anschließend mit der Zeit des automatisierten Tests verglichen. Insgesamt lässt sich sagen, dass die manuellen Tests inklusive dem Lesen der Testfallspezifikation dreimal so lange dauern wie die Automatisierten. Daraus lässt sich Abbildung 1 ableiten.

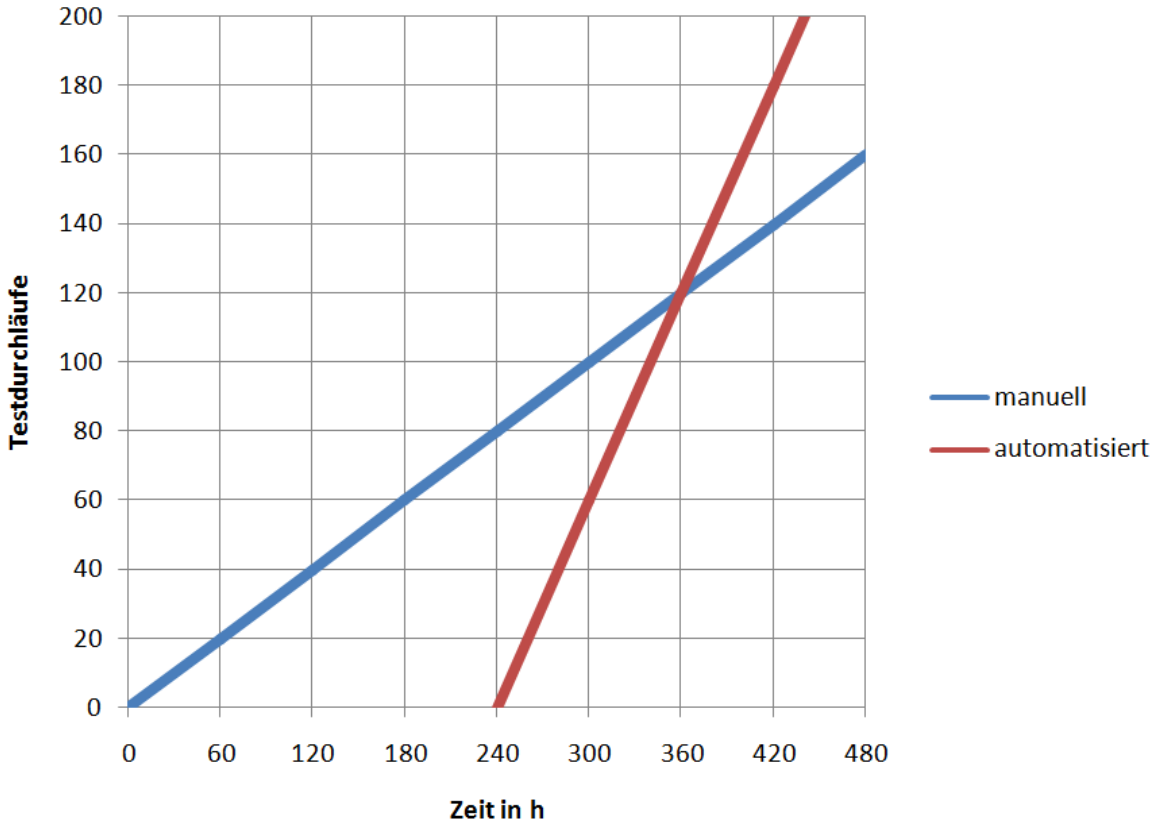


Abbildung 1: Zeit über Testdurchlauf

Obwohl die automatisierten Tests erst nach 240 Stunden begonnen wurden, liegt der Schnittpunkt beider Geraden bei 360 Stunden. Das ist der Zeitpunkt, ab dem sich die Testautomatisierung lohnt, weil diese nun deutlich zeitsparender arbeitet. Bei täglicher Ausführung entsprechen 120 Testdurchläufe etwa vier Monaten, sodass man sagen kann, dass sich in diesem Fall eine Testautomatisierung ab einer Testzeit von vier Monaten rentieren würde.

Man sieht, dass es einen bestimmten Zeitpunkt gibt, ab dem sich die Automatisierung mit Selenium lohnt. Dieser Zeitpunkt ist abhängig vom jeweiligen Projekt, denn nicht jedes Projekt lässt sich gleich gut automatisieren. Daher ist es definitiv sinnvoll, sich vor Beginn einer Testautomatisierung Gedanken zu machen, ob sich das überhaupt lohnt. Da die Purple Publishing Suite aber wie bereits erwähnt ein langfristiges Projekt ist und keine großen Codeänderungen für die Testautomatisierung benötigte, war der Schritt zur Testautomatisierung in diesem Fall die richtige Entscheidung.

### 4.4.2 Nutzen

Die automatisierten Tests werden jede Nacht ausgeführt, sodass man morgens einen fertigen Testbericht hat. Das spart zum einen Zeit, da sich kein Mitarbeiter darum kümmern muss. Gleichzeitig ist auch eine gleichbleibend hohe Qualität der Tests gesichert, indem sie immer absolut identisch ausgeführt werden. Wird der Manager also verändert, gibt es mit der Testautomatisierung nun eine verlässliche Methode, um die Korrektheit der Grundanforderungen sicherzustellen.

### 4.4.3 Abdeckung

Der Purple Manager hat einige Funktionen, die nicht mit Selenium automatisiert werden können. Dazu gehören beispielsweise Bilder und Diagramme, das Öffnen einer neuen Seite (zum Beispiel Verlinkung zur Produkthomepage) und das Prüfen von generierten Gutscheincodes. Diese Funktionen können aber problemlos manuell getestet werden, da sie nur weniger umfangreiche Funktionen des Managers sind. Von insgesamt 52 relevanten Testfällen konnten 46 mit Selenium umgesetzt werden, also etwa 88%. Dementsprechend ist die Testfallabdeckung durch die Testautomatisierung hoch.

# 5 Purple Kiosk

Der Purple Kiosk ist eine mobile Anwendung, die für Smartphones und Tablets entwickelt wurde. Sie kann für Android, iOS und Kindle erstellt werden. Auf der Startseite befinden sich neben einem Menü- und Teilen-Symbol eine horizontal scrollbare Leiste mit Reitern und darunter eine vertikal scrollbare Auflistung aller im ausgewählten Reiter befindlichen Ausgaben (siehe Abb. 9). Neben den Reitern „Meine Ausgaben“ und „Alle Ausgaben“ gibt es einen für jede Publikation, die im Purple Manager mit der Anwendung verbunden wurde. Hat man einen Reiter ausgewählt, erscheint eine Liste aller verfügbaren Ausgaben in dieser Publikation mit jeweils einer Klickfläche zum Öffnen des Inhaltsverzeichnisses und einer Klickfläche mit dem Namen „Lesen“, „Kaufen“ oder „Download“ (siehe Abb. 10). „Alle Ausgaben“ ist initial bei jedem Start der Anwendung der ausgewählte Reiter und ist eine Zusammenfassung aller Ausgaben. „Meine Ausgaben“ beinhaltet alle bereits heruntergeladenen beziehungsweise gekauften Ausgaben (siehe Abb. 9).

Das Menü lässt sich in der oberen linken Ecke ausklappen und enthält neben Links zu externen Seiten, beispielsweise zur Homepage, diverse Verlinkungen zu Informationstexten, beispielsweise zum Datenschutz, Impressum oder zu allgemeinen Informationen über die Anwendung (siehe Abb. 11). Zusätzlich gibt es noch den Menüpunkt „Abonnement“, unter dem verschiedene Abonnements abgeschlossen werden können.

## 5.1 Mögliche Vorgehensweisen

In diesem Kapitel geht es zuerst um die möglichen Testarten und anschließend werden fünf Testwerkzeuge kurz beschrieben und ein Werkzeug ausgewählt.

### 5.1.1 Testarten

Beim Testen mobiler Anwendungen stellt sich als erstes die Frage, ob man überhaupt in der Lage ist zu testen. Dies ist deswegen nötig, weil es inzwischen eine extrem große Anzahl an Geräten unterschiedlicher Hersteller mit unterschiedlichen Betriebssystemversionen gibt. Insgesamt gab es im Februar 2015 45,6 Millionen Smartphone-Nutzer in Deutschland [stab], davon hat Android mit 68% den größten Marktanteil [dei15]. Im August 2014 gab es 18.796 verschiedene Geräte, auf denen Android installiert war [Ope14], außerdem gibt es viele Versionen des Betriebssystems. Zusammengefasst gibt es – die für Wearables angepasste Version „Wear“ ausgenommen – 21 für die Softwareentwicklung unterschiedliche Android-Versionen [Wik]. Um herauszufinden, welche davon noch relevant sind und dementsprechend getestet werden sollten, wurde eine aktuelle Statistik verwendet. Darin hat ein Statistik-Portal die Daten von Android-Smartphones in der ersten

Septemberwoche 2015 mithilfe des Google Playstores gesammelt und veröffentlicht. Daraus lässt sich erkennen, dass die Versionen 1.0 bis 2.1 gar nicht mehr verwendet werden, die Version 2.2 wird nur von 0,2% genutzt [staa]. Zum Testen relevant sind also nur die Versionen ab 2.3, was insgesamt 12 Unterschiedliche sind, eine weitere Version (Android 6.0 Marshmallow) ist seit August 2015 als Developer-Preview verfügbar [Wik].

Allein diese Zahlen und die Tatsache, dass dies nur ein Teil der aktuell verwendeten Geräte ist, machen deutlich, dass es nahezu unmöglich ist, dass eine Softwareentwicklungsfirma all diese Geräte zum Testen besitzt – aktuell (Stand September 2015) gibt es fast 10.000 Geräte, die mit Google Play kompatibel sind [Goo]. Dies ist auch nicht nötig, allerdings funktioniert die entwickelte Anwendung umso zuverlässiger beim Endnutzer, je vielfältiger die Testgeräte sind.

Hat ein Unternehmen nicht genug Testgeräte zur Verfügung, gibt es diverse Anbieter, von denen man Tests durchführen lassen kann. Dabei bietet sich Testautomatisierung besonders an, um die Geräte effektiv zu nutzen und die Kosten gering zu halten. Ein Anbieter wird im folgenden Kapitel kurz vorgestellt. Sind genug Testgeräte im Unternehmen vorhanden, können die Tests vor Ort durchgeführt werden.

### 5.1.2 Testwerkzeuge für mobile Anwendungen

Die in diesem Kapitel genannten Werkzeuge wurden zunächst ohne Beachtung der in Kapitel 3.1 festgelegten Kriterien ausgewählt, um einen Einblick in die aktuell verfügbaren Werkzeuge zu bekommen. Nach einer kurzen Beschreibung zu jedem Testwerkzeug folgt am Ende dieses Kapitels die Auswahl.

**Ranorex** [Rana] und **SilkMobile** [Bor] sind kostenpflichtige Werkzeuge, die nicht Open Source sind. Beide funktionieren nach dem Aufnahme-Wiedergabe-Prinzip, also ohne Programmierkenntnisse. Ranorex bietet hierbei Bildschirmfotos innerhalb des Berichts und ist als Paket zusammen mit Web- und Desktop-Automatisierungswerkzeugen verfügbar. SilkMobile kann etwas mehr: Es liefert Performance Informationen zu jedem Test, hat eine vollständige Gesten-Unterstützung und ermöglicht eine parallele Ausführung der Tests auf mehreren Geräten.

**TestDroid** [K$^+$] ist ein kostenpflichtiger Dienst, bei dem die Tests beim Anbieter ausgeführt werden. Man entwirft also Tests und sendet diese zusammen mit der zu testenden Anwendung an TestDroid und wählt aus, auf welchen Geräten getestet werden soll. Derzeit (Stand Oktober 2015) werden 402 Testgeräte mit den Betriebssystemen iOS und Android zur Verfügung gestellt. Besonders für kleine Firmen oder für Start-ups ist ein Anbieter wie TestDroid nützlich, um eine hohe Abdeckung an Testgeräten zu erreichen.

**Appium** [App] und **MonkeyTalk** [Clo] gehören zu den Open Source Testwerkzeugen. Appium ist einfach zu bedienen und verfügt über eine große Community, die bei Fragen und Problemen weiterhelfen kann. Die Tests können in diversen Sprachen geschrieben werden, es gibt aber auch ein Werkzeug, mit dem das Aufnehmen und anschließende Abspielen von Tests möglich sind. Für die Automatisierung sind keine Codeänderungen in der Anwendung nötig und auch um plattformspezifische Besonderheiten kümmert sich der Server.

MonkeyTalk hat ebenfalls eine große Community, die sich um den Support kümmert. Es verfügt über eine einfache Aufnahme und Wiedergabe Funktion und unterstützt auch touch- und gestenbasierte Interaktionen. Android und iOS haben dabei die gleichen Kommandos, sodass man einen auf Android aufgenommenen Test auch auf iOS abspielen kann und umgekehrt. Außerdem ist die Sprache an natürliche Sprache angelehnt und somit leicht zu verstehen, es gibt aber auch eine API für Java- und Javaskript-Befehle. MonkeyTalk arbeitet objektbasiert und kann somit alle Elemente einer Anwendung erreichen. Redundanz in den Testfällen wird reduziert, indem ein Testfall mit vielen Daten durchlaufen werden kann dank einer CSV Unterstützung. In dem detaillierten Bericht sind ebenso wie bei Appium Bildschirmfotos enthalten.

Tabelle 3 zeigt eine Übersicht aller in diesem Abschnitt genannten Testwerkzeuge inklusive Informationen zu den in Kapitel 3.1 festgelegten Kriterien.

| | | **Ranorex** | **Silk Mobile** | **Appium** | **Monkey Talk** |
|---|---|---|---|---|---|
| **Produkt** | Release | 10/2015 | ? | 09/2015 | 12/2014 |
| | Support | + | + | + | + |
| | Open Source | Nein | Nein | Ja | Ja |
| **Testvorgang** | Handhabung/ Einarbeitung | + | ? | + | + |
| **Quellcode** | Darstellungsform | + | + | + | + |
| | Programmier-sprachen | O | + | + | + |
| | Eigene Programmierung | Ja | Ja | Ja | Ja |
| **Protokoll** | Darstellungsform | + | + | + | + |
| **Oberfläche** | Darstellungsform | + | + | + | + |

Tabelle 3: Kriterienübersicht für alle Testwerkzeuge des Purple Kiosks[6]

Da sprylab über eine ausreichend große Auswahl an Testgeräten verfügt (etwa 60 Stück), wird ein Anbieter wie TestDroid nicht benötigt. Auch kommen wieder nur die Open

[6] [Ranb] [Vir15] [Sch14]

Source Werkzeuge infrage. Die Entscheidung zwischen Appium und MonkeyTalk wurde auf Basis der vorhandenen Erfahrung getroffen. In der Qualitätssicherung wurde in der Vergangenheit bereits ein Projekt mit MonkeyTalk getestet, sodass für dieses Werkzeug schon Vorkenntnisse vorhanden waren.

## 5.2 Umsetzung

In diesem Kapitel geht es um die Planung der Testfälle mithilfe eines webbasierten Testmanagement-Systems und das Erstellen der Tests. Im Rahmen dieser Bachelorarbeit wurde nur die Version der Anwendung für das Betriebssystem Android getestet. Die Oberfläche auf iOS-Geräten sieht zum einen anders aus, was komplett eigene Tests erfordert hätte, zum anderen war die iOS-Version der Anwendung zu diesem Zeitpunkt noch nicht für MonkeyTalk geeignet.

### 5.2.1 Testfallkatalog im Testmanagement-System

Der Funktionsumfang des Kiosks ist deutlich geringer als der des Managers, deswegen gibt es nicht sehr viele Testfälle. Die Schwierigkeit bei der Testfallerstellung bestand eher darin, möglichst viele kleine Testfälle zu erstellen, die unabhängig voneinander testbar sind. Das Design der Anwendung kann sich nämlich stark ändern, sodass sich die Funktionen nach einem Update an einer ganz anderen Stelle befinden. Hat man dann diese Funktion unabhängig von ihrer Position innerhalb der Anwendung beschrieben, muss an diesem Testfall nichts verändert werden.

Der wichtigste Teil des Purple Kiosks ist der Kaufvorgang. Wenn der Konsument den Kaufvorgang nicht abschließen kann, verkauft der Kunde, also der Herausgeber der Publikationen, der den Kiosk kostenpflichtig betreibt, keine Ausgaben mehr und wird infolge dessen den Kiosk nicht weiter zum Verkauf nutzen wollen. Wie in [Wal07] in Abb. IV-10 zu sehen, sind die mit deutlichem Abstand häufigsten Gründe für den Abbruch des Kaufvorgangs während der Bestellung oder Bezahlung Sicherheitsbedenken oder mangelndes Vertrauen. Deswegen ist es besonders wichtig, dass der Kaufvorgang fehlerfrei ablaufen kann, sodass diese Tests besonders detailliert entworfen wurden.

### 5.2.2 Testautomatisierung

Auch bei diesen Tests war der Testfallkatalog die Vorlage, nach der die Tests geschrieben wurden. Allerdings wurden diese Testfälle nochmal so weit aufgeteilt, bis die Ebene einzelner Klickflächen erreicht war. Der Testfall „Öffnen und Verifizieren des Impressums“ wurde so zum Beispiel unterteilt in „Öffnen des Menüs“, „Öffnen des Impressums“ und „Verifizieren des Impressums“. Das Skript, das dem oben genannten Testfall entsprechen

soll, besteht also nicht mehr aus einzelnen Anweisungen, sondern nur noch aus Aufrufen der genannten Skripte, wobei die richtige Reihenfolge der Aufrufe notwendig für eine korrekte Ausführung ist.

Das größte Problem bei MonkeyTalk war, dass die IDs nicht eindeutig, also viele Elemente gleich benannt waren. Befindet man sich beispielsweise in der Übersicht seiner Ausgaben, heißt die Klickfläche für das Inhaltsverzeichnis bei allen Ausgaben gleich. Übergibt man MonkeyTalk also einfach diese ID, wird das erste Inhaltsverzeichnis genommen, das auf dem Bildschirm des Gerätes sichtbar ist. Wenn man das nicht möchte, hat man die Möglichkeit per Nummer an ein Element zu kommen. MonkeyTalk nummeriert alle auf dem Testgerät aktuell sichtbaren Elemente und lässt diese Nummern auch als ID zu. Das Problem ist natürlich, dass sich die Nummerierung immer ändert, je nachdem, ob das gewünschte Element beispielsweise ganz oben oder ganz unten auf dem Bildschirm zu sehen ist.

Dazu kommt noch, dass die Testgeräte unterschiedlich große Displays haben und sich somit auch anders verhalten (beispielsweise unterschiedlich weites Scrollen). Deswegen sollte die Ausgabe, die man anklicken möchte, entweder die erste oder die letzte sein. Der Aufwand für eine beliebige andere Position der Ausgabe würde deutlich höher sein und keinen weiteren Nutzen bringen.

Ein weiterer Punkt, der beachtet werden musste, waren die unterschiedlichen Wartezeiten der einzelnen Geräte. Die Rechenleistung der Geräte untereinander unterschied sich deutlich und damit die Tests auf möglichst vielen Geräten funktionieren, mussten sie auf die langsameren angepasst werden. Doch auch die Anzahl der Geräte, die zu einem Zeitpunkt getestet wurden, hatte Auswirkungen auf die Wartezeit beim Herunterladen von Ausgaben. Zum einen spielte die Auslastung des WLANs eine Rolle bei der Geschwindigkeit, aber auch der Server muss je nach Auslastung unterschiedlich viele Anfragen gleichzeitig bedienen. Hierfür musste ein Kompromiss gefunden werden, sodass lange genug gewartet wird, aber die Tests auch nicht unnötig in die Länge gezogen werden. Die Lösung war, dass bei einem sehr langsamen Gerät die Downloadzeit gemessen und anschließend verdreifacht wurde, um möglichst keinen Downloadprozess unnötig abzubrechen. Die Klickfläche „Download“, auf die geklickt werden muss, heißt während des Downloads „Abbrechen“ und anschließend, nachdem die Ausgabe erfolgreich heruntergeladen wurde, „Lesen“. Deshalb konnte MonkeyTalks Timeout-Funktion genutzt werden. Beim Timeout wird der Befehl so lange probiert auszuführen, bis er entweder erfolgreich war oder die angegebene Zeit abgelaufen ist. Es wurde also auf „Download“ geklickt, anschließend hat man mit der Timeout-Funktion die zuvor bestimmte Zeit darauf gewartet, dass die Klickfläche „Lesen“ heißt. Das hat den Vorteil, dass man einerseits auf langsame Geräte wartet, aber

andererseits schnelle Geräte nicht unnötig warten lässt.

Es wurden eigene Tests für Smartphones und Tablets entwickelt, da die Darstellung auf den deutlich größeren Bildschirmen der Tablets anders ist als auf den Smartphones. Nachdem eine erste lauffähige Version der Smartphone-Tests existierte, wurden diese dupliziert und an den notwendigen Stellen für die Tablets angepasst. Das war eine effiziente Vorgehensweise, weil im Anschluss bei beiden Gerätetypen nur kleinere Verbesserungen vorgenommen werden mussten wie zum Beispiel die Erhöhung der Wartezeit oder die Korrektur der ID.

## 5.3 Bewertung der Umsetzung

Das folgende Kapitel behandelt die Bewertung der Umsetzung und umfasst eine Abschätzung der Aufwände, des Nutzen und der Abdeckung.

### 5.3.1 Aufwände

Der Aufwand zur Testautomatisierung des Purple Kiosks war deutlich geringer als der des Purple Managers. Zum einen war eine erste lauffähige Version aufgrund des geringeren Funktionsumfangs schon früh möglich, wodurch man schnell die Schwierigkeiten des Programms entdecken und im weiteren Verlauf vermeiden konnte. Die Erweiterung der Smartphone-Tests auf Tablet-Tests war schnell und einfach. Ein wirkliches Problem stellten die bereits erwähnten IDs dar. Teilweise war die Nummerierung der Elemente nicht wie erwartet, da es bei MonkeyTalk offenbar Probleme bei der Erkennung der sichtbaren Elemente gibt. So begann die Nummerierung der Elemente teilweise bei der ersten Ausgabe, obwohl inzwischen schon beispielsweise bis zur vierten heruntergescrollt wurde.

### 5.3.2 Nutzen

Im Gegensatz zu einer Web- oder Desktopanwendung muss bei einer mobilen Anwendung nicht nur ein Gerät getestet werden. Derzeit gibt es bei sprylab etwa 60 Geräte, auf denen die Anwendung ausgeführt werden kann, was für einen manuellen Test kaum machbar wäre. Durch die Testautomatisierung müssen jetzt nur noch die gewünschten Geräte angeschlossen werden und die Tests laufen auf allen gleichzeitig. Außerdem ist die Zeit zum Herunterladen der Ausgaben im Verhältnis zum gesamten Test relativ hoch, was für den Tester zu einer unnötigen Wartezeit führen würde. Aus diesen Gründen ist der Nutzen durch die Testautomatisierung sehr hoch.

### 5.3.3 Abdeckung

Die Verlinkungen zu externen Webseiten, zum Beispiel zur Homepage der Purple Publishing Suite, können nicht überprüft werden. Das liegt daran, dass MonkeyTalk nur innerhalb

der zu testenden Anwendung funktioniert. Auch die Ausgabe selbst kann nicht getestet werden, da sie für MonkeyTalk nur als ein großes Element vorliegt und nicht auf einzelne Teile zugegriffen werden kann. Das ist unproblematisch, da die Korrektheit der Ausgabe direkt nach dem Erstellen der Datei getestet werden soll und somit zum Composer-Test gehört. Auch die externen Verlinkungen stellen nur einen kleinen Teil der mobilen Anwendung dar und lassen sich zudem schnell und einfach manuell überprüfen. Insgesamt ist die Abdeckung also sehr gut.

# 6 Purple Composer

Der Purple Composer ist eine umfangreiche, professionelle Designplattform, die als Mac-Desktopanwendung genutzt werden kann. Es werden sowohl eine freie Gestaltung als auch ein Import von beispielsweise PDF- oder Photoshop-Dateien ermöglicht. Für beide Möglichkeiten werden diverse Funktionalitäten geboten, so kann man zum Beispiel Seiten und Szenen verwalten, Bilder, Galerien oder eigene Texte einfügen und animieren (siehe Abb. 12, 13).

Um ein interaktives Ergebnis zu erhalten, ist ein zeit- und aktionsbasiertes Verhalten einzelner Objekte möglich, wofür es unter anderem eine Zeitleiste mit diversen Funktionalitäten gibt. Außerdem kann man das Layout einzelner Objekte oder das des gesamten Produkts detailliert anpassen. Auch das Inhaltsverzeichnis kann direkt dort verwaltet werden und auch für die nutzbaren Endgeräte können Einstellungen vorgenommen werden.

Der letzte Schritt ist der Export der Datei. Hierfür werden die erstellten Dateien verpackt und als pkar-Datei exportiert, eingefügte Videos bleiben allerdings separat, um die Dateigröße der pkar zu beschränken.

## 6.1 Mögliche Vorgehensweisen

In diesem Kapitel geht es zuerst um die möglichen Testarten und anschließend werden zwei Testwerkzeuge kurz beschrieben und deren Verwendbarkeit bewertet.

### 6.1.1 Testarten

Die naheliegenste Möglichkeit den Composer zu testen ist ein Oberflächentest. Wie schon bei den anderen Tests wird hierbei auf die Benutzeroberfläche zugegriffen und die Anwendung dadurch so bedient, wie es auch der Nutzer tun würde. Im folgenden Kapitel werden zwei Testwerkzeuge vorgestellt, mit denen ein solcher Test durchgeführt werden könnte.

### 6.1.2 Testwerkzeuge für Mac-Desktopanwendungen

**Sikuli** [Sik] ist ein kostenloses Open Source Testwerkzeug und nutzt Bildschirmausschnitte zur Automatisierung. Hierbei wird der Teil der Anwendung, der angesprochen werden soll, nicht als ID dem Code übergeben, sondern als Bild eingebunden in Pythoncode. Dieses Bild wird dann bei der Testausführung auf dem Bildschirm gesucht und man kann damit interagieren. Dies ermöglicht eine sehr einfache Bedienung und der Code wird auch für Außenstehende leicht verständlich. Allerdings ist das Programm auch sehr fehleranfällig, da es schon bei der kleinsten grafischen Änderung nicht mehr funktioniert.

**Squish GUI Tester** [Fro] von Froglogic ist kostenpflichtig und nicht Open Source, bietet aber mehr Funktionen als Sikuli. Es nutzt das Aufnahme und Wiedergabe Prinzip und bietet mehrere Skriptsprachen an, außerdem ermöglicht es datengetriebenes Testen. Bei Squish kann die Testsuite per Drag and Drop organisiert werden und man kann einzelne Variablen überwachen.

Beide Testwerkzeuge sind ungeeignet, da Sikuli durch das ausschließliche Abgleichen der Benutzeroberfläche mit Bildern zu fehleranfällig und Squish nicht Open Source ist und damit den Anspruch von sprylab nicht erfüllt. Doch auch mit besseren Testwerkzeugen wäre ein Oberflächentest zur Automatisierung zum jetzigen Zeitpunkt nicht die richtige Wahl, da es aufgrund der Vielzahl an Funktionen des Composers zu viel Zeit in Anspruch nehmen würde. Außerdem wird die Oberfläche ständig weiterentwickelt, was bei der Vielzahl an komplexen Elementen eine Wartung der Tests schwierig machen würde. Im folgenden Kapitel wird eine effektivere Methode zur Testautomatisierung des Purple Composers beschrieben.

## 6.2 Umsetzung

Bei diesem Ansatz geht es nicht um das Testen der Benutzeroberfläche, wie es bei den anderen Tests gemacht wurde. Genutzt wird die Tatsache, dass das erstellte Ergebnis beim Export verarbeitet und als Datei zur Verfügung gestellt wird. Diese Datei kann anschließend mit einer Referenzdatei, die den Sollzustand darstellt, verglichen werden.

Der erste Schritt war das Erstellen von Testdateien. Diese wurden in einer korrekt funktionierenden Version des Composers importiert und wieder exportiert, um die Referenzdatei zu erstellen. Danach wurde ein Shell-Skript geschrieben, das mithilfe von Apple Script Befehlen den Composer ansteuern kann.

Apple Script bietet die Möglichkeit, Anwendungen zu steuern. Dazu muss die Anwendung skriptfähig gemacht werden, indem man ihr unter anderem ein Wörterbuch aus Begriffen übergibt und bestimmte Klassen und Methoden implementiert. Hat man das getan, kann diese Anwendung von Apple Script kontrolliert werden. Die Kommandos werden in Form von Apple Events gesendet, die eine Art Interprozessnachricht sind.

Das Skript öffnet den Composer, importiert die Testdateien und exportiert sie wieder. Die erstellte pkar-Datei und die Referenzdatei werden nun mithilfe der Vergleichsfunktion diff aufgerufen. Diff ist in der Lage, nach Unterschieden zwischen Dateien zu suchen und diese anzugeben. Da die Referenzdatei aus den gleichen Testdateien erstellt wurde wie die neue Datei, sollten beide gleich sein. Sind sie es nicht, bedeutet das einen Fehler bei der Verarbeitung der Daten in der getesteten Version des Composers. Um das Ergebnis des

Vergleichs übersichtlicher zu gestalten, wurde es unter Verwendung von Python in HTML umgewandelt.

Nachdem die erste Ausführung des Tests mit vielen Unterschieden fehlschlug, hat sich gezeigt, dass es auch Dateiunterschiede gibt, die keinen Fehler bedeuten. So wird zum Beispiel die Versionsnummer des verwendeten Composers in der pkar-Datei gespeichert, die dementsprechend unterschiedlich sein muss. Also wurde noch eine Funktion eingefügt, die die Versionsnummern beider Dateien überschreibt, sodass sie gleich sind und kein Fehler mehr angezeigt wird.

Der nächste gefundene Fehler musste ebenfalls behoben werden: Die Reihenfolge der Elemente, die importiert wurden, wurde vom Composer beliebig abgespeichert, sodass auch hier immer Unterschiede zwischen den Dateien angezeigt wurden. Dies musste im Programmcode des Composers selbst behoben werden. Ein weiteres Problem waren die Videos, die außerhalb der pkar-Datei gespeichert werden. Diese bekamen immer einen anderen, automatisch generierten Namen und innerhalb der pkar-Datei wurde dann darauf referenziert. Auch dies wurde im Code geändert.

War der Test erfolgreich, wird die erstellte pkar-Datei als neue Referenzdatei genommen. So wird sichergestellt, dass die älteren Versionen mit den neuen kompatibel sind, was für den Kunden sehr wichtig ist.

## 6.3 Bewertung der Umsetzung

Das folgende Kapitel behandelt die Bewertung der Umsetzung und umfasst eine Abschätzung der Aufwände, des Nutzen und der Abdeckung.

### 6.3.1 Aufwände

Um ein Referenzprojekt zu erstellen und die Automatisierung vorzunehmen, wurde etwa eine Woche benötigt. Die weiteren Referenzprojekte, die noch erstellt werden sollen, werden schneller erstellt werden können, da man sich an der Umsetzung des ersten Projekts orientieren kann. Insgesamt ging die Umsetzung der Testautomatisierung also relativ schnell.

### 6.3.2 Nutzen

Schon nach der ersten Ausführung des Tests war der Nutzen groß, da so die Unstimmigkeiten innerhalb des Composers (beliebige Reihenfolge der Elemente und die Benennung der Videos) entdeckt und behoben werden konnten. Auch im weiteren Verlauf der Testautomatisierung werden die Ergebnisse von Nutzen sein, da die Tests die genaue Stelle zeigen, an der der Fehler auftritt.

### 6.3.3 Abdeckung

Diese Art von Testautomatisierung testet nicht die Nutzeroberfläche. Getestet wird die Korrektheit des Datenstroms, der beim Im- und Export stattfindet. Sollte an dieser Stelle ein Fehler auftreten, wird er aufgrund des Dateivergleichs definitiv gefunden.

# 7 Fazit/Ausblick

Der Arbeitstitel dieser Bachelorarbeit war „Konzeption und Evaluation der ganzheitlichen Qualitätssicherung einer Multi-Plattform Publishing Suite mit Testautomatisierung“. Zu Beginn der Arbeit wurde ein Plan erstellt, mit welchen Methoden die Komponenten am besten automatisiert werden sollten. Schon bei diesem ersten Schritt wurde deutlich, dass eine ganzheitliche Qualitätssicherung im Rahmen einer Bachelorarbeit unmöglich umzusetzen war. Somit wurde der Plan so erstellt, dass jeweils der für den Kunden wichtigste Teil von jeder der drei Komponenten automatisiert werden sollte.

Begonnen wurde dann mit dem umfangreichsten Teil, nämlich dem Purple Manager. Trotz des großen Zeitaufwandes, der für die Automatisierung der Oberflächentests benötigt wurde, war deutlich, dass eine Testautomatisierung der richtige Weg für die Qualitätssicherung war. Die Abdeckung der Testfälle war hoch (etwa 88%) und die Testergebnisse standen bereits am nächsten Morgen zur Verfügung. Manuelle Tests wären bei diesem Funktionsumfang nur unzureichend möglich gewesen, sodass keine gleichbleibend gute Qualität des Produkts gewährt werden könnte. Auch die Entwickler profitieren von schnellen Testergebnissen, da sie unverzüglich entstandene Fehler wieder beheben können bevor darauf aufgebaut wird.

Die Automatisierung der Oberflächentests des Purple Kiosks war ebenfalls nützlich, auch wenn manuelle Tests hier aufgrund des deutlich geringeren Funktionsumfangs möglich gewesen wären. Da aber auch die Testautomatisierung dementsprechend schnell ging, ist auch dies eine große Entlastung für die Qualitätssicherung. Der größte Vorteil an dieser Automatisierung ist, dass viele Geräte bei nahezu gleicher Testdauer gleichzeitig getestet werden können.

Beim Purple Composer wären Oberflächentests im Rahmen dieser Bachelorarbeit wahrscheinlich nicht möglich gewesen. Der Funktionsumfang ist sehr hoch, weswegen die Entscheidung auf eine Testautomatisierung zur Kontrolle der Versionskompatibilität fiel. Dabei wurde ein Skript ausgeführt, das ein Dateipaket in den Composer importiert, exportiert und anschließend mit einer Referenzdatei vergleicht. Hier ist der große Vorteil, dass man innerhalb von sehr kurzer Zeit weiß, ob die neue Version mit der alten kompatibel ist. Sollte dies nicht der Fall sein, sieht man dank des Dateivergleichs auch, wie genau der Fehler entstanden ist. Dies ist besonders für die Entwickler praktisch, da sie jede neue Version sofort testen können und sich einen großen Teil der Fehlersuche sparen.

In Zukunft können die Tests noch ausreichend erweitert werden. So wurde beispielsweise bereits eine externe Firma damit beauftragt, Black-Box-Tests in Bezug auf die Sicherheit des Purple Managers durchzuführen. Auch das Thema Last- beziehungsweise Stresstests

ist im Rahmen dieser Bachelorarbeit unberührt geblieben, eventuell können auch davon noch Teile automatisiert werden. Auch die Zusammenarbeit der Komponenten wurde bisher nicht getestet.

Beim Purple Kiosk werden die zukünftigen Testergebnisse zeigen, ob und wo noch weiterer Automatisierungsbedarf besteht. Hier liegt besonderes Augenmerk auf der Kauffunktion, da ein Fehler an dieser Stelle zum Verlust der Konsumenten und damit auch der Kunden führt.

Insgesamt war dieser Anfang der ganzheitlichen Testautomatisierung der Purple Publishing Suite ein Erfolg.

# A Purple Manager

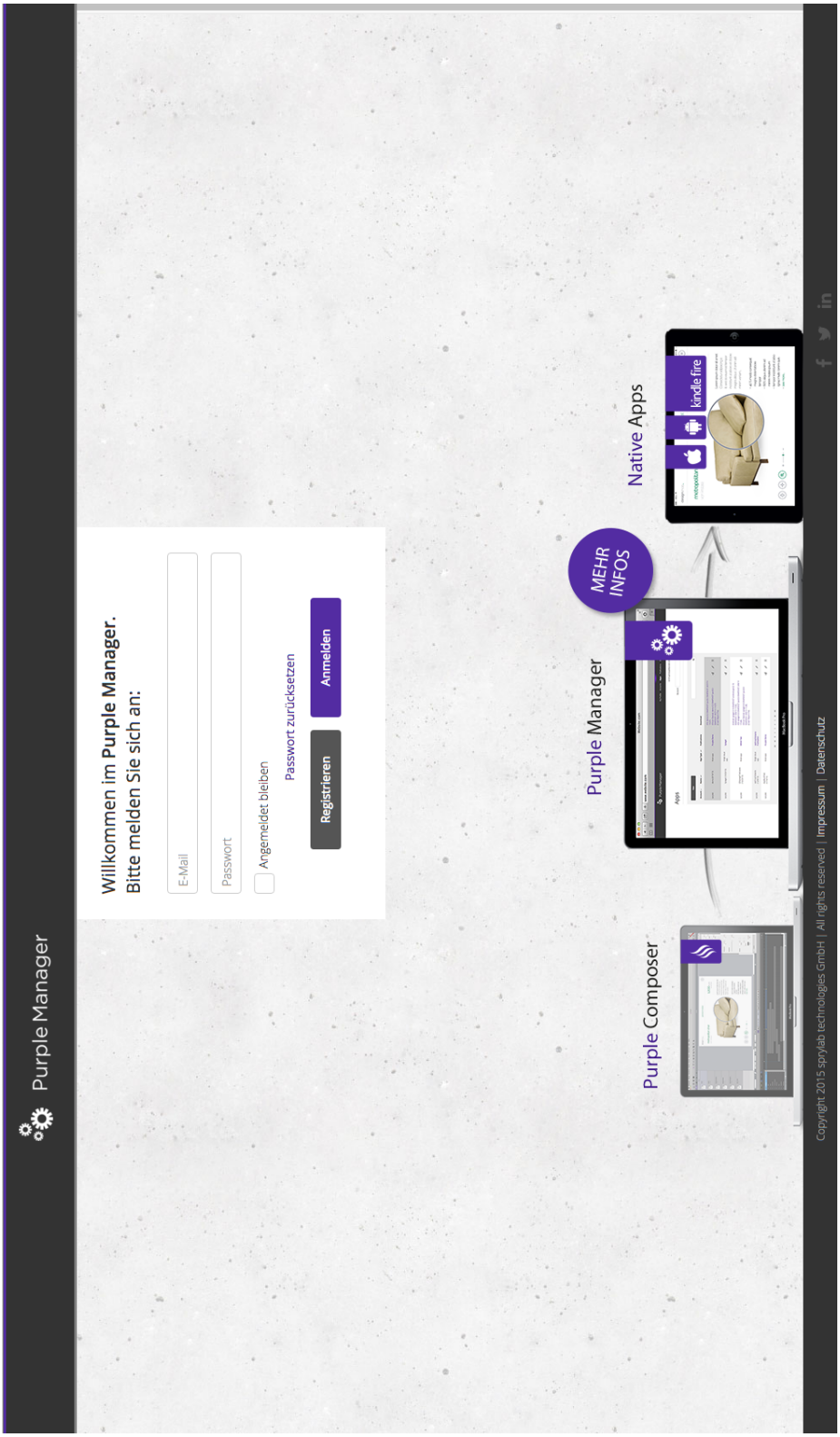


Abbildung 2: Begrüßungsseite mit Anmeldebereich, Registrierung und knappem Überblick über die Komponenten der Purple Publishing Suite

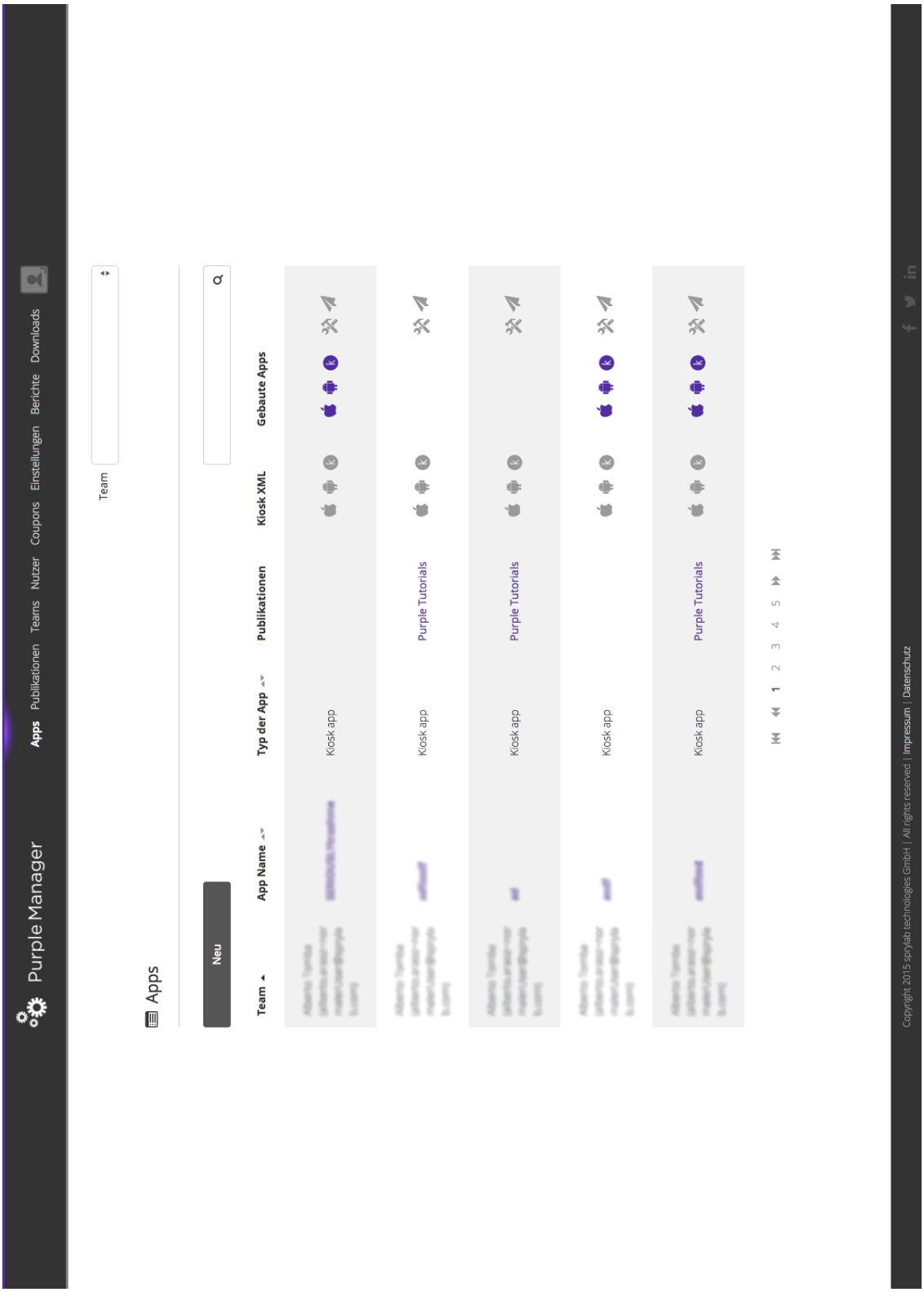


Abbildung 3: Startseite nach der Anmeldung: Überblick über alle Apps als Tabelle mit mehreren Seiten und mit Klickfläche zur Erstellung eines neuen Eintrags

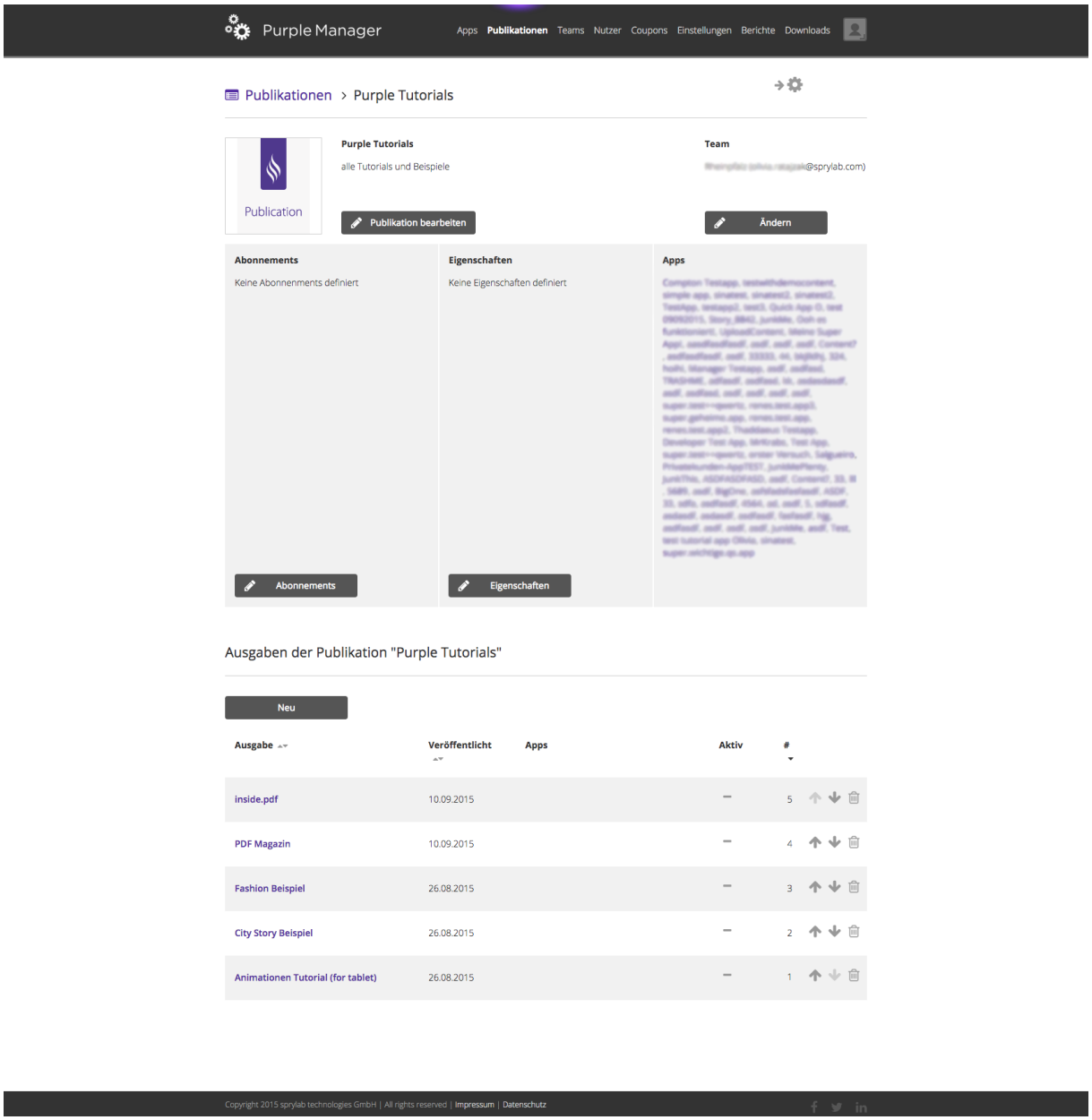


Abbildung 4: Ansicht einer Publikation, unter anderem mit einer Liste von Apps, in denen die Publikation enthalten ist und mit enthaltenen Ausgaben

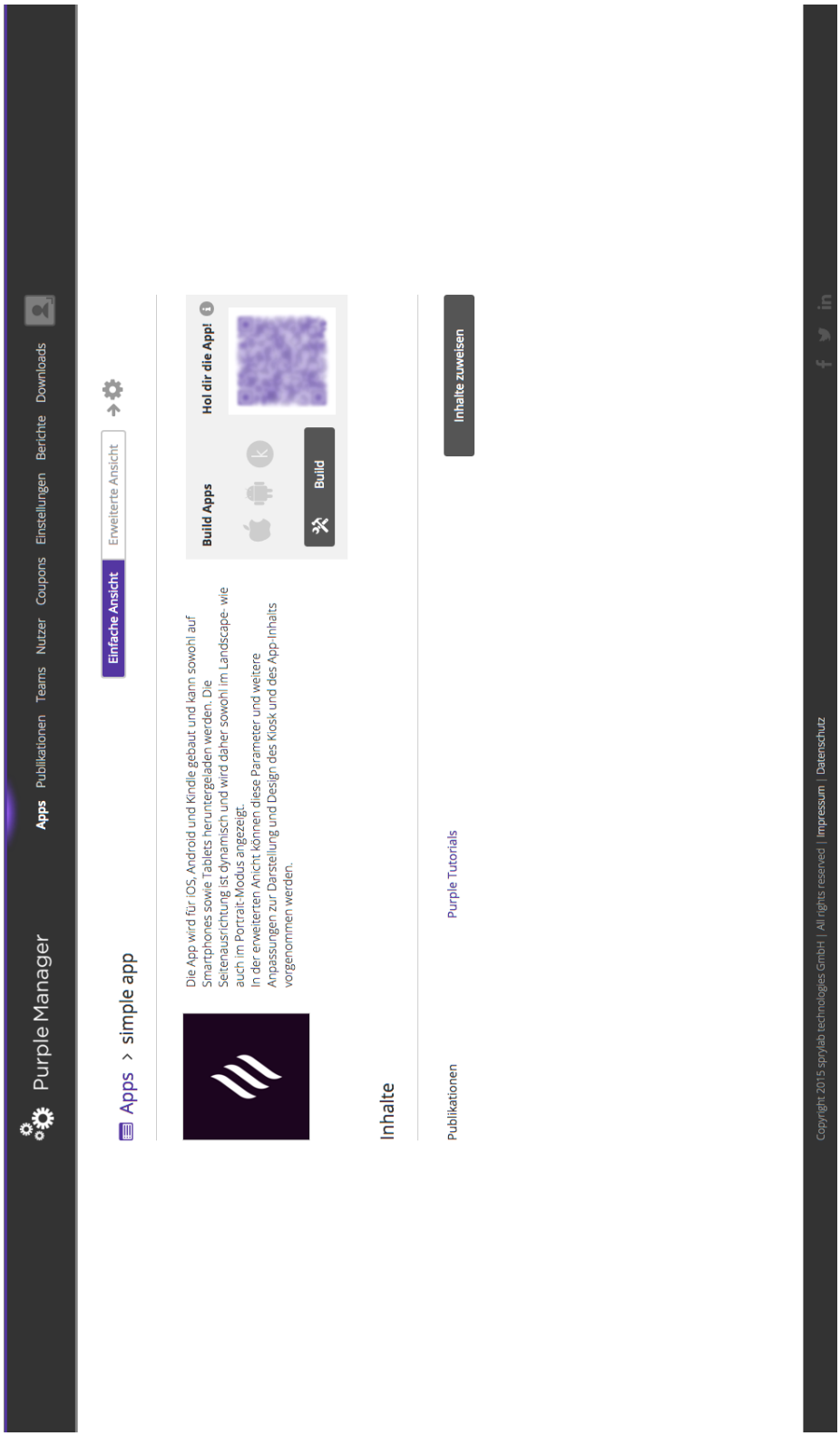


Abbildung 5: Vereinfachte Ansicht einer App mit Beschreibung, Publikationen, QR-Code zum Herunterladen der App und Möglichkeit zum Generieren

Purple Manager
Apps Publikationen Teams Nutzer Coupons Einstellungen Berichte Downloads

Apps > simple app
Einfache Ansicht Erweiterte Ansicht

App Name simple app
Typ der App Kiosk app
App Bezahlmodell -
Template Standard (Development)
Paketname com.sprylabsimpleapp
Plattformen iOS, Android, Kindle
Basiseinstellungen

Team
QS
Ändern

Build Apps
Hol dir die App!
Einstellungen
Build

Push senden
Einstellungen
Senden

App XML

Kiosk XML

Inhalte

Publikationen Purple Tutorials
Inhalte zuweisen

App / Kiosk Design

Purple verwendet Standard- und angepasste, kundenspezifische App-Templates, um Anwendungen zu bauen. Dieser Abschnitt ermöglicht es Ihnen, das Design der App und des Kiosks ausgiebig zu konfigurieren. Sie können hier auch den Promo-Bereich des Kiosks mit Werbeflächen aus dem Purple Composer bestücken.
App / Kiosk Design
Werbefläche

Tracking & Analyse

Nutzungsstatistiken geben wichtige Einblicke in das Nutzerverhalten und Hinweise, um Ihr Angebot zu verbessern. Hier integrieren Sie beliebte Tracking-Dienste wie z.B. Google Analytics oder Flurry.
Tracking & Analyse

Dynamische App-Inhalte

Dynamische Vorschauinhalte: –
Upload Vorschau-Inhalt
Dynamische Live-Inhalte: –
Upload Live-Inhalt

Profiles (Expert)

Neu

| Profile | Basis-einstellungen | Build Einstellungen | Push Einstellungen | App / Kiosk Design | Tracking & Analyse |
|---|---|---|---|---|---|
| Keine Ergebnisse | | | | | |

Transaktionen

| Status | Typ | App / Inhalte | Preis |
|---|---|---|---|
| Keine Ergebnisse | | | |

Copyright 2015 sprylab technologies GmbH | All rights reserved | Impressum | Datenschutz

Abbildung 6: Erweiterte Ansicht einer App mit diversen Einstellungsmöglichkeiten

Purple Manager
Apps Publikationen Teams Nutzer Coupons Einstellungen Berichte Downloads

Publikationen > Purple Tutorials > PDF Magazin

PDF Magazin
Beispiel für den Import und Animation einer pdf-Datei
Ausgabe bearbeiten Coming Soon off

Eigenschaften
Keine Eigenschaften definiert
Eigenschaften

Produkte

Apps

Content-Versionen der Ausgabe "PDF Magazin"
Vollversionen Vorschauversionen

Vorschau-Versionen Ihrer Ausgaben können Ihre Kunden lesen, auch wenn sie die Vollversion der Ausgabe noch nicht gekauft haben. Am Ende der Vorschau wird ein Kaufhinweis für die Vollversion der Ausgabe eingeblendet. Wenn Sie Inhalte aus dem Purple Composer einer Ausgabe zuweisen, können Sie entscheiden, ob die Inhalte für die Vorschau oder die Vollversion verwendet werden sollen.

Neu

| Nummer | Beschreibung | Erstellt | In Release-App veröffentlichen | In Vorschau-App veröffentlichen |
|---|---|---|---|---|
| 10 | | 21.09.2015 | off | on |
| 9 | stephans version | 15.09.2015 | off | off |
| 8 | | 14.09.2015 | off | off |

1 2 3 4

Bundles

Diese Version beinhaltet 1 Referenzen zu fehlenden Zusatzdateien (klicken Sie hier für Details)

Inhaltspaket hochladen

| Name | Plattformen | Geräteklassen | Hochgeladen | Seit Version | Größe |
|---|---|---|---|---|---|
| demo_pdf_tablet_und_phone.pkar | | | 21.09.2015 | | 10 MB |

Assets

Zusatzdatei hochladen Alle löschen

| Name | Plattformen | Geräteklassen | Hochgeladen | Seit Version | Größe |
|---|---|---|---|---|---|
| 4b308eb7a95757743ce86ee9cd4351fa.mp4 | | | 21.09.2015 | | 10 MB |
| 5cb3b076c4b88e839362f403ac4868ca.mp3 | | | 21.09.2015 | | 109 KB |

Copyright 2015 sprylab technologies GmbH | All rights reserved | Impressum | Datenschutz

Abbildung 7: Ansicht einer Ausgabe mit Versionen, pkar-Datei und zusätzlichen Video- und Musikdateien

# B Kiosk

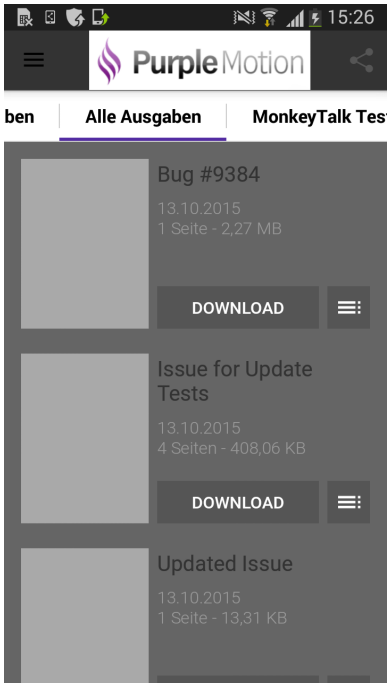


Abbildung 8: Startseite nach Öffnen des Kiosks im Reiter „Alle Ausgaben“ ohne heruntergeladene Ausgabe


Abbildung 9: Reiter „Meine Ausgaben“ ohne heruntergeladene Ausgabe

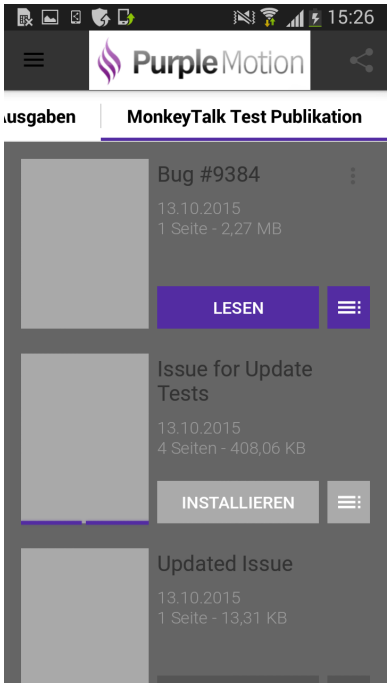


Abbildung 10: Erste Ausgabe heruntergeladen, zweite wird installiert

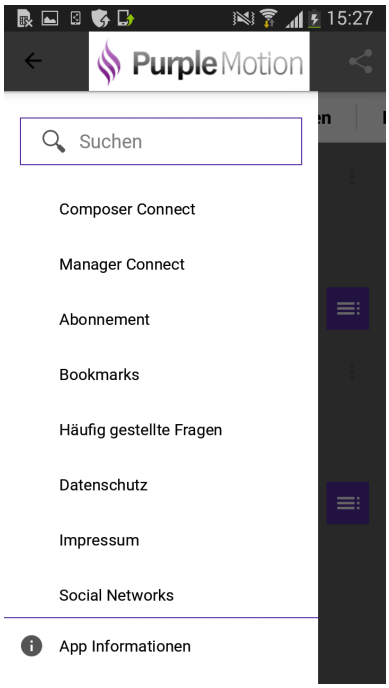


Abbildung 11: Menü

# C Purple Composer

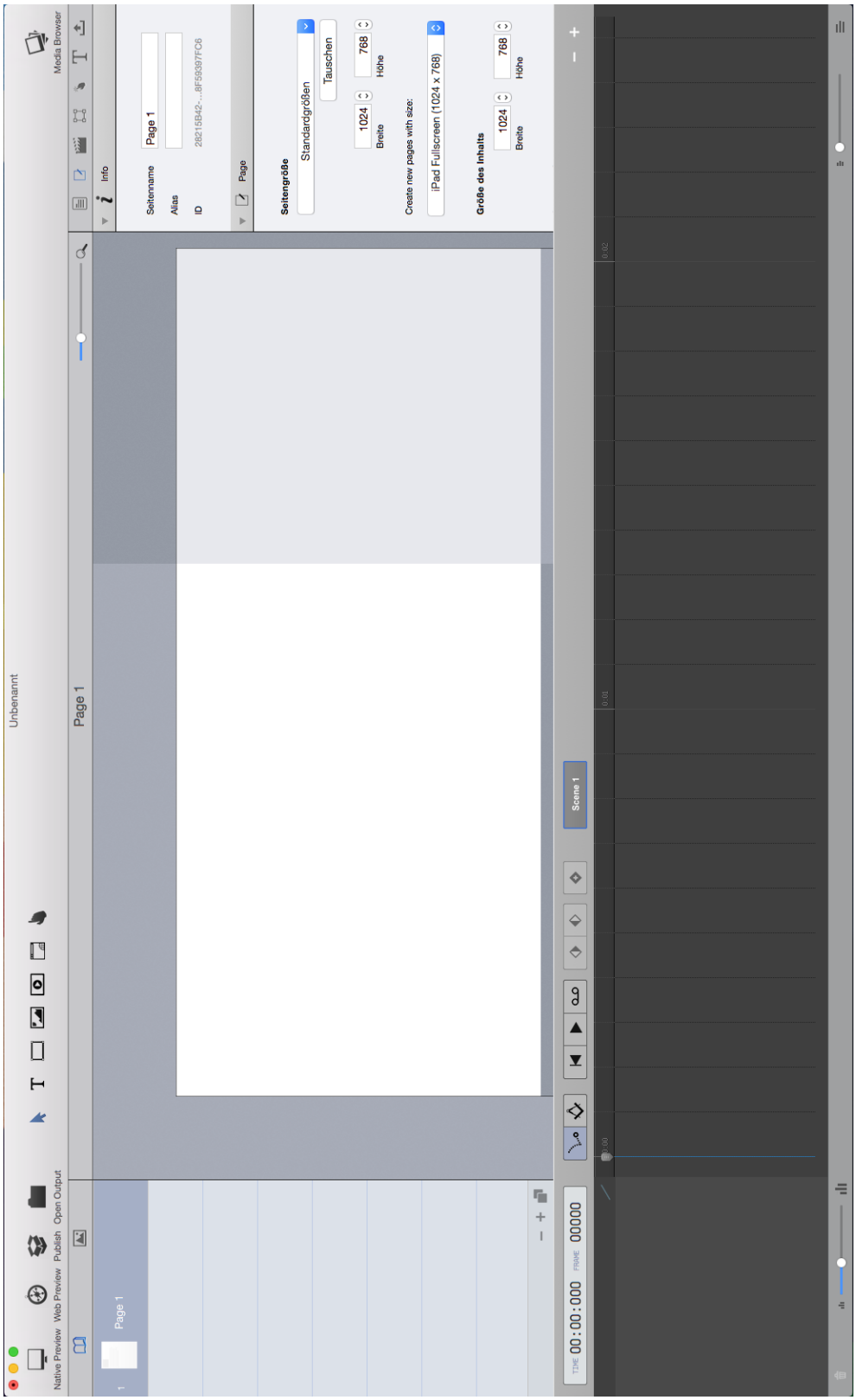


Abbildung 12: Ansicht des Composers nach dem Start der Anwendung

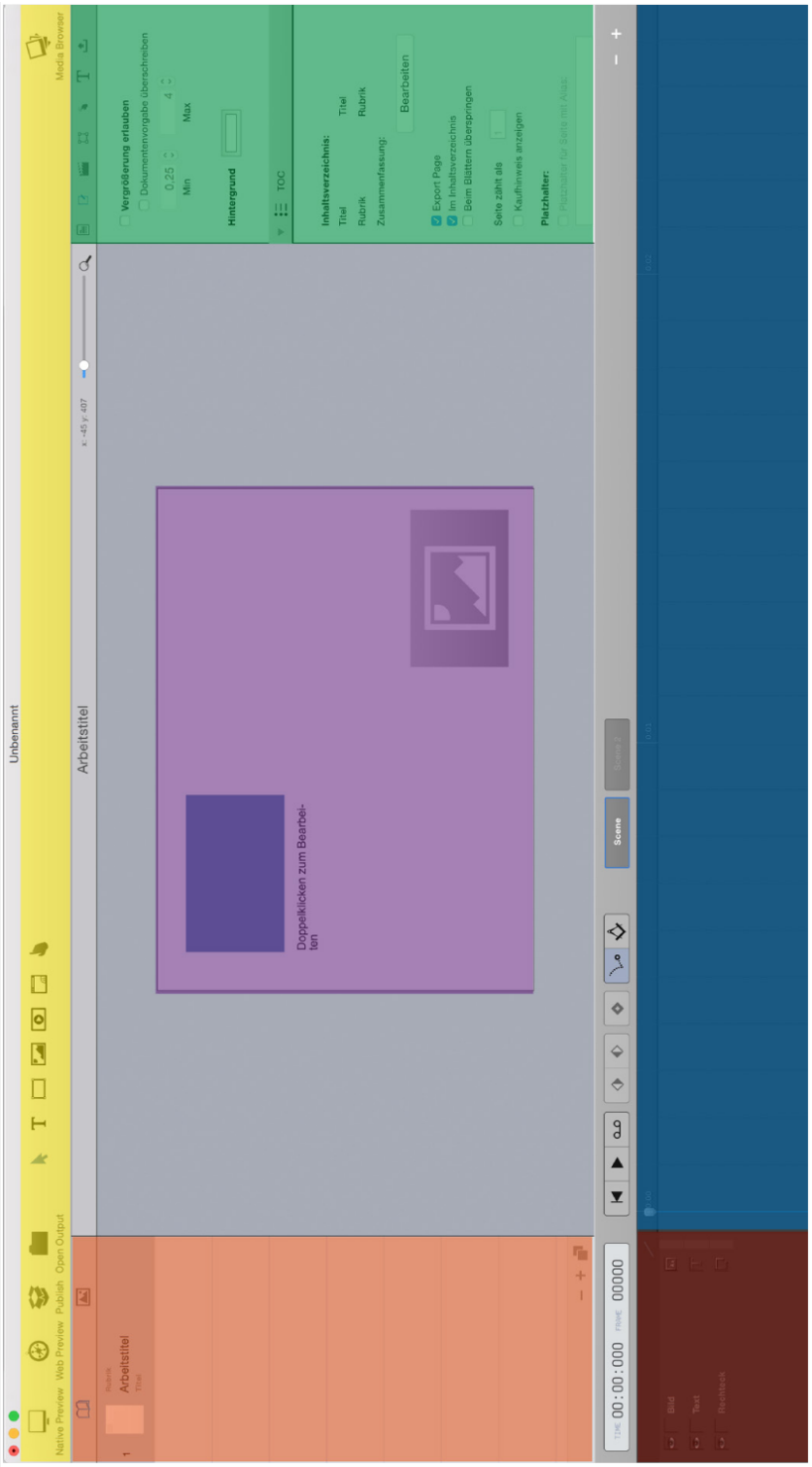


Abbildung 13: Gelb: links: Klickfläche für Vorschau und Export des Inhalts; oberhalb der Bühne links: Werkzeugleiste; rechts: Medienbibliothek zum schnellen Einfügen von Mediendateien; Orange: Seitenübersicht und verwendete Medien; Grün: Inspektor für Eigenschaften (Projekt, Seite/Bühne, Szene, Layout, Objekt, Text, Export); Lila: Bühne, der Arbeitsbereich; Rot: Objektliste (Listenansicht aller Objekte, die sich auf der Bühne befinden); Blau: Zeitleiste (Steuerung von Animationen), darüber Szenensteuerung

# Verzeichnisse

## I Literatur- und Quellenverzeichnis

[App] APPIUM: *Appium.* – Zugriff: 28.10.2015, Archiviert mit WebCite®: http://www.webcitation.org/6cdXwl9vq

[B+] BADLE, Samit u. a.: *SeleniumHQ.* – Zugriff: 28.10.2015, Archiviert mit WebCite®: http://www.webcitation.org/6ccQuzcrV

[Ban15] BANET, Amir: *LeanFT and UFT 12.5 are now available to accelerate your application delivery.* Juli 2015. – Zugriff: 28.10.2015, Archiviert mit WebCite®: http://www.webcitation.org/6cddkt9IU

[Bor] BORLAND: *Silk Mobile.* – Zugriff: 28.10.2015, Archiviert mit WebCite®: http://www.webcitation.org/6cdXNebKn

[Cana] CANOO: *Canoo WebTest Downloads.* – Zugriff: 28.10.2015, Archiviert mit WebCite®: http://www.webcitation.org/6cde3rRYU

[Canb] CANOO: *WebTest.* – Zugriff: 28.10.2015, Archiviert mit WebCite®: http://www.webcitation.org/6ccR9eFmw

[Clo] CLOUDMONKEY: *MonkeyTalk.* – Zugriff: 28.10.2015, Archiviert mit WebCite®: http://www.webcitation.org/6cdY8KpnP

[dei15] DEINPHONE: *Aktueller Stand des Smartphone Marktes in Deutschland und der Welt [Infografik].* Juni 2015. – Zugriff: 28.10.2015, Archiviert mit WebCite®: http://www.webcitation.org/6cdWxCg1m

[Fro] FROGLOGIC: *Squish GUI Tester.* – Zugriff: 28.10.2015, Archiviert mit WebCite®: http://www.webcitation.org/6cimNUGzw

[Goo] GOOGLE: *Unterstützte Geräte.* – Zugriff: 28.10.2015, Archiviert mit WebCite®: http://www.webcitation.org/6cilQG17S

[HP] HP: *Unified Functional Testing.* – Zugriff: 28.10.2015, Archiviert mit WebCite®: http://www.webcitation.org/6ccPxb7bO

[Inf] INFOTECH, Cygnet: *TestingWhiz.* – Zugriff: 28.10.2015, Archiviert mit WebCite®: http://www.webcitation.org/6ccPbj4fI

[K+] KAASILA, Marko u. a.: *TestDroid.* – Zugriff: 28.10.2015, Archiviert mit WebCite®: http://www.webcitation.org/6cdXYMMZv

[Ope14] OPENSIGNAL: *Android Fragmentation Visualized.* August 2014. – Zugriff: 28.10.2015, Archiviert mit WebCite®: http://www.webcitation.org/6cdX8S6rq

[PR] PR: *TestingWhiz 4.5 Released: 360 Grad Test Automation with New Features and More.* – Zugriff: 28.10.2015, Archiviert mit WebCite®: http://www.webcitation.org/6cdcwDCYo

[Rana] RANOREX: *Ranorex.* – Zugriff: 28.10.2015, Archiviert mit WebCite®: http://www.webcitation.org/6ccPkGRE6

[Ranb] RANOREX: *Release Notes.* – Zugriff: 28.10.2015, Archiviert mit WebCite®: http://www.webcitation.org/6cddEKfpH

[Sah] SAHI: *Sahi Pro Installers Archive.* – Zugriff: 28.10.2015, Archiviert mit WebCite®: http://www.webcitation.org/6cdeMePpY

[Sch14] SCHWARZ, E.: *MonkeyTalk Professional Edition V2.0.10.beta released.* Dezember 2014. – Zugriff: 28.10.2015, Archiviert mit WebCite®: http://www.webcitation.org/6cdeZLo28

[Sel] SELENIUM: *Downloads.* – Zugriff: 28.10.2015, Archiviert mit WebCite®: http://www.webcitation.org/6cddrfGI5

[Sik] SIKULI: *Sikuli Script.* – Zugriff: 28.10.2015, Archiviert mit WebCite®: http://www.webcitation.org/6cim8TE2u

[Smaa] SMARTBEAR: *TestComplete.* – Zugriff: 28.10.2015, Archiviert mit WebCite®: http://www.webcitation.org/6ccPodygC

[Smab] SMARTBEAR: *What's New in TestComplete 11.11.* – Zugriff: 28.10.2015, Archiviert mit WebCite®: http://www.webcitation.org/6cddOvHwh

[Sof] SOFTWARE, Tyto: *SahiPro.* – Zugriff: 28.10.2015, Archiviert mit WebCite®: http://www.webcitation.org/6ccRzdkPl

[staa] STATISTA: *Anteil der verschiedenen Android-Versionen an allen Geräten mit Android OS weltweit im Zeitraum 29. September bis 05. Oktober 2015.* – Zugriff: 28.10.2015, Archiviert mit WebCite®: http://www.webcitation.org/6cdXIWvVB

[stab] STATISTA: *Anzahl der Smartphone-Nutzer in Deutschland in den Jahren 2009 bis 2015.* – Zugriff: 28.10.2015, Archiviert mit WebCite®: http://www.webcitation.org/6cdWSrMAX

[Tela] TELERIK: *Release History.* – Zugriff: 28.10.2015, Archiviert mit WebCite®: http://www.webcitation.org/6cdckF2Oo

[Telb] TELERIK: *Test Studio.* – Zugriff: 28.10.2015, Archiviert mit WebCite®: http://www.webcitation.org/6ccP2k7HN

[Tes] TESTLINK: *TestLink Open Source Test Management.* – Zugriff: 28.10.2015, Archiviert mit WebCite®: http://www.webcitation.org/6cgdKVwav

[Vig05a] VIGENSCHOW, Uwe: *Objektorientiertes Testen und Testautomatisierung in der Praxis: Konzepte, Techniken und Verfahren.* 1. Heidelberg : dpunkt.verlag, 2005. – S. 159

[Vig05b] VIGENSCHOW, Uwe: *Objektorientiertes Testen und Testautomatisierung in der Praxis: Konzepte, Techniken und Verfahren.* 1. Heidelberg : dpunkt.verlag, 2005. – S. 160

[Vig05c] VIGENSCHOW, Uwe: *Objektorientiertes Testen und Testautomatisierung in der Praxis: Konzepte, Techniken und Verfahren.* 1. Heidelberg : dpunkt.verlag, 2005. – S. 192 ff.

[Vig05d] VIGENSCHOW, Uwe: *Objektorientiertes Testen und Testautomatisierung in der Praxis: Konzepte, Techniken und Verfahren.* 1. Heidelberg : dpunkt.verlag, 2005. – S. 21 – 25

[Vig05e] VIGENSCHOW, Uwe: *Objektorientiertes Testen und Testautomatisierung in der Praxis: Konzepte, Techniken und Verfahren.* 1. Heidelberg : dpunkt.verlag, 2005. – S. 191

[Vir15] VIRANI, Moiz: *Changes in version 1.4.13 (from 1.4.12).* September 2015. – Zugriff: 28.10.2015, Archiviert mit WebCite®: `http://www.webcitation.org/6cdelbzEn`

[Wal07] WALGENBACH, Gertrud: *Die Vorteilssituation von Innovatoren auf elektronischen Märkten: Strategische Relevanz des frühen Markteintritts am Beispiel des Online-Buchhandels.* 1. Wiesbaden : Deutscher Universitätsverlag, 2007. – S. 211

[Wata] WATIR: *Watir.* – Zugriff: 28.10.2015, Archiviert mit WebCite®: `http://www.webcitation.org/6ccRO81cX`

[Watb] WATIR: *Watir Gems Updated.* – Zugriff: 28.10.2015, Archiviert mit WebCite®: `http://www.webcitation.org/6cdeIECsv`

[Wik] WIKIPEDIA: *Liste von Android-Versionen.* – Zugriff: 28.10.2015, Archiviert mit WebCite®: `http://www.webcitation.org/6cdXE57LG`

# II Abbildungsverzeichnis



# III Tabellenverzeichnis