Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

Luca Steffen Nerlich

Automatisiertes Testen von Tracking Request in Web-Anwendungen

Fakultät Technik und Informatik
Studiendepartment Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Luca Steffen Nerlich

Automatisiertes Testen von Tracking Request in Web-Anwendungen

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung

im Studiengang Bachelor of Science Wirtschaftsinformatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Bettina Buth
Zweitgutachter: Prof. Dr. Ulrike Steffens

Eingereicht am: 22. August 2016

Luca Steffen Nerlich

Thema der Arbeit
Automatisiertes Testen von Tracking Request in Web-Anwendungen

Stichworte
Analytics, Tracking, Selenium Webdriver, Automatisierung, Testen

Kurzzusammenfassung
Die vorliegende Arbeit präsentiert Grundlagen des halbautomatischen Testens von Tracking Requests in Web-Anwendungen. Zusätzlich zeigt eine Fallstudie die beispielhafte Implementierung von 'Page Objects' und den dazugehörigen Testfällen im Umfeld des Web-Trackings. Aufbauend auf dieser Herangehensweise wird die Ausgliederung und Restrukturierung von Arbeitsschritten zwischen einem technischem Dienstleister und seinen Kunden vorgestellt.

Luca Steffen Nerlich

Title of the paper
Automated testing of tracking requests in web-applications

Keywords
Analytics, Tracking, Selenium Webdriver, Automation, Testing

Abstract
The present paper depicts the basics for half-automated testing of tracking requests in web-applications. A case study additionally presents an exemplary implementation of page objects and the corresponding test cases. Based on this the paper proposes a new allocation of work procedures between the technical service provider and its customer.

Inhaltsverzeichnis

Tabellenverzeichnis

Abbildungsverzeichnis

Sourcecodeverzeichnis

1 Einleitung

Initial wird die aktuelle Situation konzentriert dargelegt und die Problemstellung umrissen. Nach der Definition von Zielen wird abschließend die Arbeit knapp in Kapitel unterteilt und beschrieben.

1.1 Motivation

Die Implementierung und fortlaufende Funktion von Analyse- und Nutzerverfolgungsverfahren auf Webseiten ist wichtiger Bestandteil moderner Webentwicklung. Webseitenbetreiber und Firmen basieren ihre Produkte und deren Weiterentwicklungen auf erkanntem Kundenverhalten und deren Wünsche. Des Weiteren werden Veränderungen im Nutzerverhalten früh erkannt. Somit lassen sich Rückschlüsse auf eventuelle Stimmungsänderungen der Kundschaft schließen. Auf Basis dieser Verhaltensmuster lassen sich weiterführende Massennahmen ableiten. Die Analyse der Kundenstimmung, sowie deren Wünsche, ermöglicht eine nachhaltige Umsatzsteigerung. Geschäftsmodelle die hiervon profitieren sind beispielsweise Suchmaschinenprovider, E-Mail Dienste, oder Hostingprovider. Zusammenfassend sind dies die Anbieter von 'Software as a Service' Produkten (vgl. Schneider u. a., Seite 8f).

Nutzerverfolgungsverfahren(oder Trackingmechanismen) laufen parallel während der Nutzung einer Webseite im Hintergrund. Verschiedenes Verhalten und vordefinierte Aktionen werden analysiert. Daraufhin sendet der Webbrowser Meldungen an den verwaltenden Analytics Server. Beispielsweise kann so die verweilte Zeit, Abbruchzeitpunkte bei einer Videowiedergabe, genutzte Links oder das Verlassen der Webseite

gespeichert und ausgewertet werden. Entsprechende Leistungskennzahlen, insbesondere die Besuchsdauer des Nutzers, sind aussagekräftige Mittel zur Bewertung des Erfolgs der Internetpräsenz (vgl. Xun [2015]).

Anhand dieser gesammelten Informationen ist es dem Webseitenbetreiber möglich, sein Angebot dem Kunden- und Nutzerverhalten entsprechend anzupassen. Bei Fehlern in der Deutung oder Datensammlung kann dies falsche Rückschlüsse auf die Meinung der Kunden oder technische Funktionalität zulassen. Aus diesem Grund ist die stetige Überprüfung des Datensammelvorgangs essentiell. Je mehr sich ein Analyst auf diese Datensätze verlassen kann, desto wertvoller sind sie. Diese Verlässlichkeit soll durch automatisiertes und halbautomatisiertes Testen gestärkt werden.

Der Thesis Titel 'Automatisiertes Testen von Tracking Requests in Web-Anwendungen' wirft die Frage nach der aktuellen Lösungen auf. Die Arbeit zielt auf die Analyse dieser Vorgänge und die Findung möglicher Verbesserungen ab. Durch eine möglichst automatisierte, Regelmäßige Überprüfung von Fehlerfällen, soll der Problematik entgegengewirkt werden.

1.2 Unternehmenskooperation

Im Rahmen des Studiums in dualer Form ist die vorliegende Arbeit in Kooperation mit dem Unternehmen t8y.com GmbH aus Hamburg, Deutschland, und der Hochschule für Angewandte Wissenschaften (HAW) entstanden. Alle nachfolgenden Arbeitsabläufe und beschriebenen Geschäftsprozesse beziehen sich auf das allgemeine Verfahren innerhalb des Unternehmens. Codeausschnitte repräsentieren keine Kundenprojekte und sind lediglich Verdeutlichungen des erarbeiteten Konzepts. Jegliche Beispiele externer Unternehmen sind frei zugänglich und stellen keine vertragsrelevanten Informationen dar.

t8y.com GmbH
Bernhard-Nocht-Straße 113
20359 Hamburg
Telefon: +49 40 2351 390
Telefax: +49 40 2351 3993
E-Mail: info@t8y.com
Internet: https://t8y.com/
Vertretungsberechtigte Geschäftsführer: Detlef Niemann-Bode, Christian Koop
Registergericht: Amtsgericht Hamburg
Registernummer: hrb 117865
(Stand: 21.08.2016)

1.3 Problemstellung

Das Testen von Tracking Requests und entsprechenden Mechanismen ist Teil des allgemeinen Testens von Webseiten und Webanwendungen. Ein umfassendes und vollständiges Testen gewährleistet die erwartete Funktionalität der Anwendung. Die Reduzierung des hierfür benötigten Zeitaufwandes unter Erhalt der Qualität ist naturgemäß Ziel eines jeden Unternehmens. Diesem steht der manuelle Testaufbau, sowie das händische Nachbauen und Implementieren der Anforderungen gegenüber. Da im Falle von t8y.com's aktuellem Arbeitsablauf Trackingverfahren von einem Mitarbeiter manuell überprüft werden müssen, kann es hier zu Fehlern kommen. Beschäftigt sich eine Person länger mit nur einer Tätigkeit, steigt die Gefahr des Tunnelblicks. Zusätzlich reduziert monotone Arbeit die Gründlichkeit und Präzision bei der Ausführung und senkt die Arbeitsmotivation. Automatisierte Vorgänge können bei korrekter Funktionalität Abhilfe schaffen und Prozesse beschleunigen. Der Wartungsaufwand hängt von der Implementierung ab. Im Falle von Selenium Webdriver Tests entspricht dieser der benötigten Zeit zum Anpassen fehlerhafter Elementlokatoren. Abzuwägen ist der initiale Aufbauaufwand des automatisierten Systems im Vergleich zu der möglichen ersparten Zeit. Ein automatischer und regelmäßiger Abgleich der Anfrageparameter mit entsprechender Vorgabe entspräche bereits einem Zeitgewinn, da die zeitaufwändige, manuelle Überprüfung des Netzwerkverkehrs entfällt.

Ein Tag-Management-System (Siehe Abschnitt 2.2.1) ermöglicht das Einschleusen von Code durch externe Editoren und Datenanalysten. Hierdurch kann die doppelte Implementation dergleichen Trackingereignisse entstehen. Außerdem kann die Optik und Funktionsweise der Webseite verändert und angepasst werden. Dies kann zusätzlich Einfluss auf die korrekte Funktionsweise haben. Die hieraus entstehende, zusätzliche Kommunikation zwischen Entwicklern und Analysten erzeugt einen erhöhten Arbeitsaufwand. Die Kernproblematik ist an dieser Schnittstelle angesiedelt. Eine effizientere Aufgabenteilung zwischen beiden Parteien, sowie zeitlich, effektiveres Auffinden entstandener Fehler ist wünschenswert.

1.4 Zielsetzungen

Auf Basis obiger Überlegungen entstehen folgende Ziele und Richtlinien. Im Zuge dieser Arbeit werden alle bearbeitet und mit Inhalt versehen. In Abschnitt 6.1 werden die Ergebnisse zusammengefasst und erneut auf diese Ziele projiziert.

Z1 - Verbesserung der Code Qualität durch konstantes Testen

Z2 - Früherkennung möglicher Fehler und Fehlerquellen

Z3 - Reduzierung des QS Aufwandes

Z4 - Straffung und Vereinheitlichung aktueller Arbeitsschritte

Z5 - Halb-Automatisierung aktuell manuell ausgeführter Tätigkeiten

Ziel der Arbeit ist die Entwicklung und Konzeption eines Systems zur automatisierten Überprüfung gesetzter Trackingparameter, die der Aktivitätsverfolgung von Nutzern dienen, sowie deren Auslösung basierend auf vordefiniertem Verhalten. Zusätzliche Differenzierung zwischen reiner Webseiten Entwicklung und der Überprüfung von Trackingfunktionalitäten erleichtert nachträgliche Arbeiten im Hinblick auf externe Analysten und Tag Management-Systeme. Anhand folgendem Aufbau setzt diese Arbeit obige Ziele um. Auf Basis eines Trackingkonzeptes wird entsprechendes Nutzerverhalten auf einer Webseite analysiert und gespeichert. Dessen Funktionalität wird durch manuell geschriebene (hier mit jUnit[1]) Tests überprüft. Entsprechende Tracking Requests [2] werden aus dem HTTP Archiv (.har / HAR) extrahiert und mit vordefinierten, erwarteten abgeglichen. Durch ein Deployment auf dem Continuous Integration Server Jenkins[3] wird die regelmäßige und automatische Überprüfung gewährleistet. Die Erstellung von Page Objects (Siehe Abschnitt 5.1) für die in Abbildung 2.5 dargestellten SUTs ermöglicht die unkomplizierte und leicht verständliche Imple-

[1] http://junit.org/junit4/
[2] Anfragen an den Analyticsserver (Bsp.: Google Analytics, Adobe Analytics) inklusive gesetzter Parameter
[3] https://jenkins.io/solutions/pipeline/

mentation von Testfällen. Jene Aufgaben können somit auch von weniger technisch versierten Mitarbeitern oder Externen erledigt werden. Entsprechende Kommunikationsstrukturen und Arbeitsabläufe existieren zum Zeitpunkt dieser Arbeit nicht öffentlich.

1.5 Gliederung

Die vorliegende Arbeit ist in sieben Kapitel gegliedert. Nachfolgend ist der Aufbau knapp dargestellt.

K1 **Einleitung** erläutert die Motivation zur Erstellung dieser Arbeit, sowie entsprechende Probleme und einhergehende Ziele.

K2 **Grundlagen** leitet in das Thema des Testens, sowie in die Grundzüge der Nutzerverhaltensverfolgung (Analytics) ein und schafft eine Wissensbasis.

K3 **Literaturüberblick** stellt weitere Forschungen und Arbeiten rund um Testen von Tracking dar.

K4 **Anforderungsanalyse** stellt die gegebenen Probleme und Situationen ausführlich dar. Des Weiteren werden weitere Angebote auf dem Markt betrachtet und weiter ins Thema eingeführt.

K5 **Methodik und Implementierung** erarbeitet einen exemplarischen Ansatz anhand eines selbstgewählten Beispiels. Dies stellt die Probleme und einhergehende Lösungsfindung in den Mittelpunkt. Der Aufbau des Programms und entsprechende Arbeitsabläufe werden präsentiert und erklärt.

K6 **Ergebnisse und Fazit** stellt die erarbeiteten Ziele den gestellten Problemen gegenüber und präsentiert erste Lösungsansätze, sowie mögliche praktischen Nutzen. Die Ergebnisse der Arbeit werden zusammengefasst, gefolgt von einem Ausblick auf mögliche Erweiterungen, sowie einer Kritik.

2 Grundlagen des Testens und der Webanalyse

In diesem Kapitel wird eine gemeinsame Wissensbasis für die anschließende Bearbeitung der Problemstellung geschaffen. Es werden Grundlagen des Testens und verschiedener Web-Tracking Mechanismen gelegt.

2.1 Web Analytics

Das grundlegende Ziel der Web Analyse ist die Entwicklung und Optimierung eines auf den Kunden und Nutzer abgestimmten Internetauftritts. Maßgebend ist hierbei eine für den Besucher klare und einfache Führung, sowie schnell ersichtlich und direkt verständlicher Inhalt. Für den Entwickler einer Webseite ist es ohne das vorherige Einholen von Feedback nicht möglich zu erfahren, inwiefern die Webseite zugreifenden Besuchern gefallen hat, oder ob das vom Kunden Erwartete mit dem präsentierten und erhaltenen Inhalt übereinstimmt. Diese Informationen sind für den Fortbestand mancher Unternehmen jedoch essentiell. Bietet beispielsweise eine Firma online Dienstleistungen an, der mögliche Kunde kann aber nicht erkennen, welche Angebote zur Verfügung stehen, verlieren sie Einnahmen und Aufträge. Dieses sehr simpel Beispiel soll lediglich verdeutlichen, dass der Internetauftritt, wenn effizient und ansprechend, bedeutsam zum Umsatz beiträgt. Die Webseite soll entsprechende Aufmerksamkeit des Kunden erreichen und sich dadurch positiv präsentieren. Bereits kleine Veränderungen können eine große Auswirkung zeigen. Ein Hörgeräte Unter-

nehmen hat Optik ihrer Webseite schlichter gestaltet und konnte dadurch 300% mehr Umsatz generieren. Die Software Firma Valve stellte fest, dass mit bereits geringer Rabattierung von 10% eine kurzfristige Umsatzsteigerung von 35% zu realisieren gewesen ist (vgl. Kissmetrics [2016]). Auch kommt Xun in ihrem Beitrag (Xun [2015]) zum Ergebnis, dass die Besuchsdauer eines Besuchers auf der Webseite maßgeblich zum geplanten Erfolg, wie beispielsweise dem Abschluss einer Bestellung, beiträgt.

Wie in Abbildung 2.1 dargestellt, streben Betreiber danach, Nutzer auf ihre Webseite zu ziehen, um sie daraufhin vom Angebotenen überzeugen zu können. Schließlich soll der Kunden zum abschließenden Ziel geführt werden (vgl. Sikowitz u. a. [2016]). Hier muss sich der Inhaber, beziehungsweise Auftraggeber, aber im Klaren sein, was dieses Ziel sein soll. Dies kann zum Beispiel das Betreiben eines Onlineshops, Aufbau einer Support-, oder Feedbackseite, sowie die Entwicklung einer Webseite zur Darstellung von Nachrichten sein. Das entsprechende Ziel wäre dann jeweils der Abschluss eines Kaufvertrags erfolgreiche Fund erwünschter Informationen, oder das regelmäßige Betrachten und Lesen von Nachrichten durch den Nutzer (vgl. Waisberg und Kaushik [2009], Seite 2).

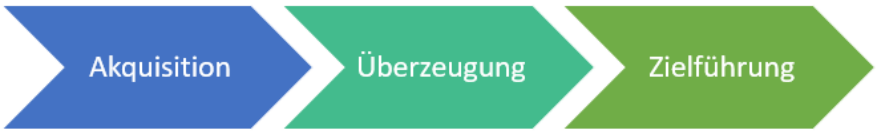

Abbildung 2.1: Kundenführung

2.1.1 Marktverteilung

Die Nutzung von Webseitenanalyse Tools ermöglicht eine komfortable Möglichkeit, das Verhalten von Nutzern der eigenen Webseite zu messen. Grundlegende Funktionen umfassen den Ein- und Ausstiegspunkt von Nutzern, Verweildauer oder beliebte Downloads. Die zwei größten Vertreter am Markt (Top 1000 meist besuchte .com

Webseiten) sind 'Google Analytics[1]' und 'Adobe Analytics[2]'. Im Raum der .de Seiten hält 'Google Analytics' knapp 53% des Markts (Siehe Abbildung 2.2[3]), gefolgt von dem Open Source Tool 'Piwik[4]', sowie 'e-tracker[5]'.

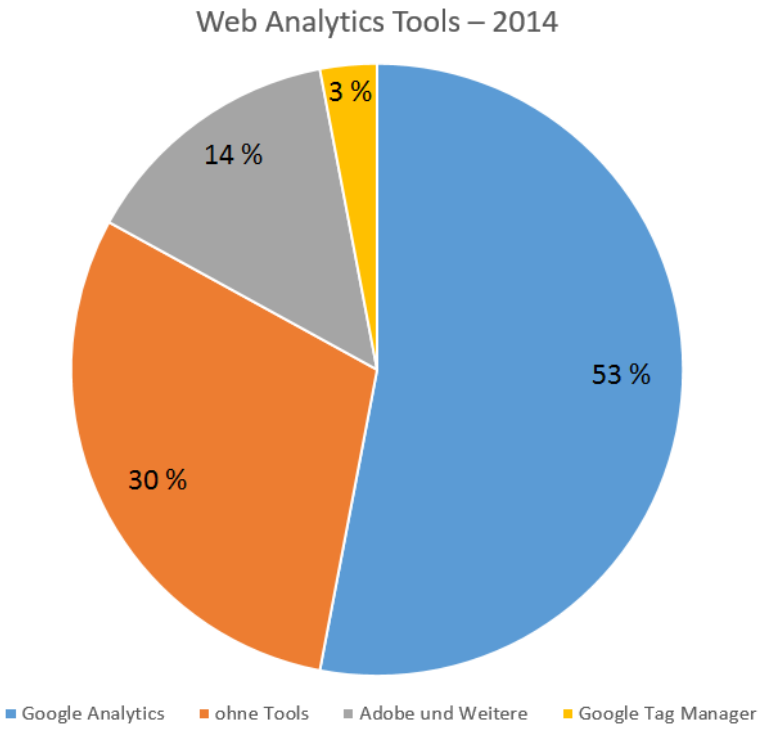

Abbildung 2.2: Web Tracking - Tool Verbreitung 2014

[1] https://www.google.com/intl/de_de/analytics/
[2] http://www.adobe.com/de/marketing-cloud/web-analytics.html
[3] https://web-analytics-tools.com/news/36-idealobserver/432-trends-im-webanalyse-markt-2014-google-dominiert-weiterhin-im-massenmarkt,-top-portale-setzen-auf-deutsche-profil%C3%B6sungen.html
[4] http://www.piwik.org/
[5] https://www.etracker.com/de.html

2.1.2 Auswirkungen und Gefahren für Endnutzer

Wer im Web surft, läuft Gefahr, sich durch gefühlte Anonymisierung in Sicherheit zu wähnen. Diese trügerische, vermutete Anonymität - verursacht durch die Abstraktion der eigenen Gedanken und des Handelns, führt zu Schlussfolgerungen, die nicht der logischen Konsequenz der ursprünglichen Intention entsprechen. Dem gemeinen Endnutzer ist nicht bewusst, dass seit der Entwicklung von Trackingmechanismen ein jeder Schritt im Internet protokolliert, gesammelt und analysiert wird. Dies geschieht fortwährend ohne die gegebene Zustimmung. Unternehmen wie Google erstellen komplexe Profile für jeden einzelnen und lernen dadurch seine Gewohnheiten, Verhalten und Interessen (vgl. Schneider u. a., Seite 5f). Ein einfaches Beispiel stellt die personenbezogene Werbung dar. Nutzer A sieht sich auf Seite B Mobiltelefonverträge an und bekommt daraufhin auf Seite B Bannerwerbung zu weiteren Vertragsangeboten angezeigt.

Durch die massive Ansammlung von Daten und einhergehender Verknüpfung mit Identitäten und sozialen Beziehungen entsteht eine wertvolle Sammlung persönlicher Daten. Unternehmen können gezielt potentielle Kunden erreichen. Des Weiteren ist es möglich, Rückschlüsse auf die Marktdurchdringung und Produktreichweite zu ziehen. Durch zusätzliche Kombinationen gesammelter Datensätze ist die Generierung weiterer Informationen möglich. Beispielsweise lassen Bewegungsprofile zu verschiedenen Tageszeiten Rückschlüsse auf den Arbeitsplatz und Wohnort zu. Hierbei ist die Trennung zwischen personenbezogener und personenübergreifender Datensammlung zu beachten. Auf der einen Seite steht die Aggregation und Profilbildung einer einzelnen Identität. Hier wird ein digitaler Fußabdruck einer Person erstellt. Interessen, Vorlieben, Geschmäcker und Verhalten lassen eine präzise Identifikation zu. Eine Rückverfolgung wird somit über einen Zeitraum und unterschiedliche Seiten möglich (*Cross Domain Tracking*).

Auf der anderen Seite befindet sich die Analyse von Gruppenverhalten und der gesamten Kundenschaft. Für einen Onlineshop ist es dadurch möglich, die Beliebtheit gewisser Produkte oder Produktgruppen zu analysieren. Auch sind Rückschlüsse auf

vorherrschende Tageszeiten und Wochentagen möglich. Beispielsweise könnte man annehmen, dass Kunden häufig am Samstagvormittag einkaufen. Durch die Sammlung von personenübergreifenden Daten ist dieser Fall möglicherweise belegbar.

Auf weitere Gefahren und Einschränkungen in Richtung Freiheit, Privatsphäre oder Demokratie soll an dieser Stelle nicht eingegangen werden, da es den Rahmen der Arbeit überschreitet. Mehr Informationen sind dem Web Tracking Report 2014 zu entnehmen (vgl. Schneider u. a., Seite 58ff).

Prävention

Browser Erweiterungen und verschiedene Tools ermöglichen eine einfache und weitestgehend automatisierte Möglichkeit zur Blockierung von Web Tracking Anfragen. Prominente und kostenlose Vertreter sind Ghostery[6] und uBlock Origin[7]. Ein weiterer Vorteil der Blockierung von Anfragen ist der Zugewinn an Performance und die Reduzierung des Datenverkehrvolumens. Durch das Verhindern von Anfragen und Empfangen von Antworten können Webseiten schneller geladen werden. Die üblichen Browser unterstützen zudem die Blockierung von Cookies Dritter.

2.2 Web Tracking Prozess

Bei der Analyse des Nutzerverhaltens spielt das Web Tracking eine besondere Rolle für die Ermittlung und Protokollierung verwendeter Funktionen und ausgelöster Ereignissee. Im folgenden wird der Prozess des Trackings genauer erläutert und an einem technischen Beispiel vorgestellt.

Webtracking oder Clickstreamanalyse versucht den Verbraucher bei der Nutzung der Webseite zu verfolgen, um Nutzungsdaten zu akkumulieren. Anhand dieser Daten können die Webseitenfunktionen und E-Commerce Produkte präziser angepasst und

[6] https://www.ghostery.com/our-solutions/ghostery-browser-extension/
[7] https://github.com/gorhill/uBlock

ausgebaut werden. Beispielsweise kann so speziell auf den Verbraucher angepasste Werbung ausgespielt werden. Auf Basis von historischem Verhalten, besuchten Webseiten und angeklickten Elementen, könnten präferierte Produkte ermittelt und zielgerichtet beworben werden (vgl. Schneider u. a., Seite 7f).

Beim Aufruf einer Webseite oder beim Interagieren mit Elementen, können diese Aktionen an ein zentrales Tool oder einen zentralen Server weitergeleitet werden. Der Aufruf kann einer HTTP GET Methode mit entsprechenden URL Parametern entsprechen, ist aber nicht auf diese begrenzt. Auch Smartphone-Apps verfügen über eine Vielzahl an Trackingmechanismen. Möglich wäre die Analyse der Verweildauer pro Seite, Scroll-Geschwindigkeit, oder die Login-/Logout Zeitpunkte. Der Fantasie sind hier kaum Grenzen gesetzt und jeder Anbieter einer Webseite oder Onlinelösung implementiert eigene Varianten. Tracking geschieht generell in Echtzeit. Dies bedeutet, dass die Anfragen an den Analyticsserver zum Zeitpunkt des Webseitenaufrufs gestellt werden. Abbildung 2.3 (Siehe Schneider u. a., Seite 21) verdeutlicht den Datenfluss vom Start der Nutzerinteraktion bis zur Auslieferung der Elemente. Durch die Anfrage eines Nutzers werden Elemente geladen und gleichzeitig Daten hochgeladen. Dies geschieht ohne, dass der Nutzer hiervon etwas bemerkt. Ein Beispiel ist die Einbindung eines Trackingpixels. Dieser hat keinerlei Auswirkung auf die Darstellung der eigentlichen Webseite, ermöglicht lediglich die Integration weiterer Trackingmechanismen. Zum Beispiel könnte durch jedes Laden des Trackingpixels ein Seitenaufruf gezählt werden.

Entsprechende Anfragen werden nach Definition übermittelt und vom Server gesammelt, verarbeitet und ausgewertet. Der genaue Ablauf und das Handling der so erhaltenen Daten hängt von der Implementation des Analysetools und der Nutzerintention ab.

Nachfolgend eine Beispiel GET Anfrage mit entsprechenden Parametern:

http://analytics.myserver.com?
pagename=home&**c1=de**&**c8=de**&**u12=meyer**&**d1=01012016**

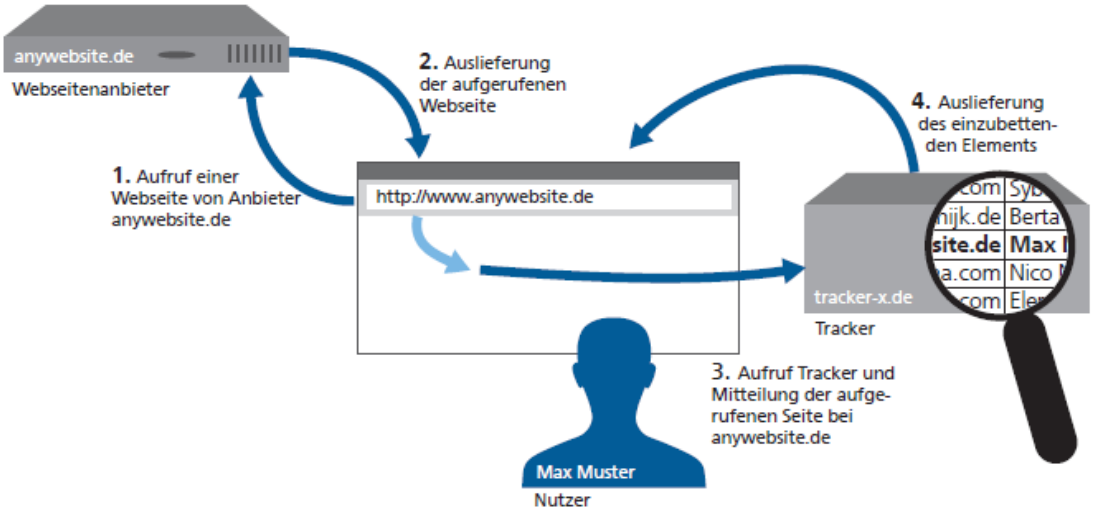

Abbildung 2.3: Web Tracking - Datenfluss

Sourcecode 2.1 zeigt URL Parameter als JSON Objekt. Hieraus könnte sich schließen lassen, dass der Nutzer 'Meyer' am 01.01.2015, aus Deutschland, die Seite 'Home' mit deutschen Spracheinstellungen geöffnet hat.

```
"request": {
"url": "http://analytics.myserver.com?schlüsselX:wertX& [...]",
"queryString": [
{"pagename": "home",
"c1": "de",
"c8": "de",
"u12": "meyer",
"d1": "01012016"}
]
}
```

Sourcecode 2.1: Web Tracking - URL Request Parameter

Die Namen der Parameter Variablen sind hier individuell zu bestimmen und mit dem Analysetool abzugleichen. Aus diesem fiktiven Aufruf könnte man zum Beispiel ableiten, dass die Seite 'home' aufgerufen wurde, von dem Nutzer '*nerlich*', die Sprach- und Landeseinstellungen entsprachen Deutschland ('*de*') und der Aufruf ist aus dem Jahr 2016.

Als Besucher einer Webseite lassen sich diese Trackingrequests in der Debug Konsole des Browsers (Chrome F12) betrachten. Häufig ist der Aufbau und die Benennung jedoch kryptisch und für Menschen nur schwer zu entziffern.

2.2.1 Tag Managementsysteme

Tag Management Systeme ermöglichen das nachträgliche Einschleusen von Code auf einer Webseite, ohne dessen Kernimplementation anfassen zu müssen. Der hinzugefügte Code kann eigenes Javascript, oder kleinere HTML Blöcke sein. Außerdem kann somit weiteres Tracking eingebaut und verwaltet werden. Externen Editoren wird die Möglichkeit gegeben, direkt und unkompliziert Anpassungen vorzunehmen, sowie den Inhalt oder die Darstellung zu erweitern. Dies spart dem Auftraggeber, Nutzer oder Kunden Geld, da entsprechende Aufgaben von eigenen Mitarbeiter übernommen werden können. Gleichzeitig stehen die freien Ressourcen für weitere Aufgaben zur Verfügung.

Google Tag Manager

Google bietet seit Ende 2012 den 'Google Tag Manager'[8] an. Dieser ermöglicht obige Funktionen durch die Verwaltung von Containern und Benutzern. Ein Tag entspricht dem hinzugefügten Code. Durch die Integration von Triggern wird die Auslösung eines Tags gesteuert. Abbildung 2.4 zeigt die Verwaltung der Trigger in einer Liste. Der Ereignistyp definiert den Auslöseelementtyp. In diesem Fall darf der Klick nur auf einen Link erfolgen. Zusätzlich muss der Link eine entsprechende ID aufweisen.

[8] https://www.google.com/intl/de/tagmanager/

Name ▲	Ereignistyp	Filter		Tags	Letzte Bearbeitung
Download File	Nur Links	Click ID	ist gleich llEN	1	vor ein paar Sekunden
		Click ID	ist gleich llDE		

Abbildung 2.4: Google Tag Manager - Trigger Verwaltung

Adobe Tag Manager

Adobe bietet analog zum Google Tag Manager den Service Activation[9] an. Dieser ermöglicht die Integration von benutzerabhängigem Verhalten. Durch die Einbindung von Tags kann die Aktivierung weiteren Inhalts gesteuert werden, somit ist die dynamische Anpassung an einen Nutzer möglich. Des Weiteren erlaubt Activation die Integration von Regeln und Genehmigungen, sowie Applikationen. Hiermit ist eine detaillierte Gruppierung von Benutzerrollen (zB. Besucher, Editor, Admin) und Anpassung von Funktionen möglich.

2.3 Strukturiertes Testen

Der Sinn des Testens ist die kontinuierliche Überprüfung des erwarteten Verhaltens sowie die korrekte Funktionalität der entwickelten Objekte. Elemente und Funktionen sind häufig abhängig von weiteren Faktoren. Das bedeutet, sie erwarten einen vordefinierten Input, oder greifen auf externe Ressourcen zu. Abhängigkeiten dieser Art können sich im Laufe des Entwicklungsprozesses verändern und somit Einfluss auf weitere Elemente und erwartete Verhalten nehmen. Während der Entwicklung und fortlaufenden Nutzung kann nicht davon ausgegangen werden, dass solch ein Fall bei Eintritt bemerkt wird. Um diesem entgegen zu wirken und Folgefehler proaktiv zu vermeiden, beziehungsweise direkt bei Entstehung ausbessern zu können, ist ein kontinuierlicher und umfassender Testaufbau nötig.

[9] http://www.adobe.com/de/marketing-cloud/activation.html

Abbildung 2.5 (vgl. Blackburn und Nauman [2004]) stellt den typischen Aufbau eines Systems (Beinhaltet Nutzerinterface, sowie eine Logik und Datenverarbeitungsschicht) und seines Nutzers dar. Hier als 'System Under Test' repräsentiert es das zu testende System der Testsuite. Im Kontext dieser Arbeit ist dies eine Web-Applikation. Diese bietet eine Interaktionsmöglichkeit für Benutzer und diverse technische Funktionalitäten, wie beispielsweise Nutzerkonten, E-Commerce Lösungen oder Trackingmechanismen. Letztere sind Grundbestandteil der vorliegenden Arbeit.

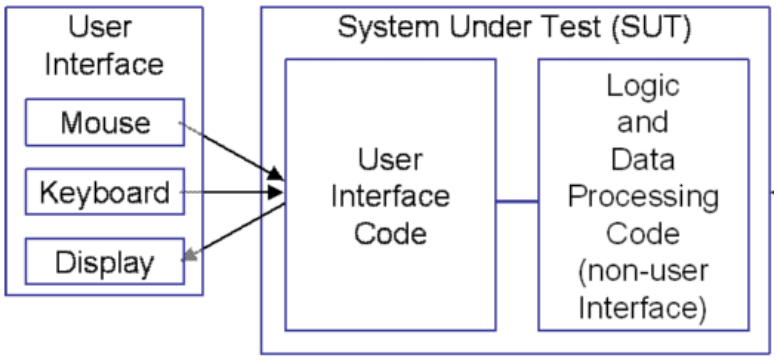

Abbildung 2.5: System Under Test

Der Input ist nicht an die Bedienung eines klassischen Desktop Computers gebunden. In der heutigen Zeit nimmt die Bedeutsamkeit von mobilen Geräten, wie Smartphones oder Tabletcomputern, enorm zu. Aktuell besitzen knapp 30% der Weltbevölkerung ein Smartphone[10]. Dies verdeutlicht den Fokus auf mobiler Webentwicklung und insbesondere dem veränderten Verhalten. Laut Microsoft [2015] beträgt die durchschnittliche Aufmerksamkeitsspanne eines typischen Menschen im Jahre 2013 rund acht Sekunden. Damit einhergehend ist die Optimierung der eigenen Internetpräsenz enorm wichtig. Wird die Webseite oder deren Inhalt nicht zügig als interessant, oder persönlich wichtig eingestuft, steigt die Gefahr, den Nutzer / Kunden zu verlieren (Verlassen die Webseite).

Durch die mögliche, riesige Anzahl von Kombinationsmöglichkeiten des Inputs, sowie unterschiedliche Verwendung von Elementen, ist eine komplette, alles umfassende Testabdeckung extrem aufwändig. Aus diesem Grund ist eine klare Systematik, sowie

[10] http://de.statista.com/statistik/daten/studie/309656/umfrage/prognose-zur-anzahl-der-smartphone-nutzer-weltweit/

ein durchdachter Testprozess, von grundlegender Wichtigkeit. Dieses Ungewisse sollte durch die Gestaltung und den Aufbau von Tests soweit wie möglich abgedeckt werden. Dabei muss bedacht werden, dass Tests lediglich fehlerhafte Software bescheinigen. Ein fehlerfreier Testdurchlauf bedeutet im Umkehrschluss nicht, dass keine weiteren Unstimmigkeiten und Fehler vorhanden sind. Dies ist lediglich für primitive Funktionalitäten wie das Laden der Seite vom Server, oder das Betätigen eines Buttons möglich (vgl. Buth u. a. [2015] Seite 11). Ein Fehler in diesem Sinne entspricht folgender Definition des Duden:

> *etwas, was falsch ist, vom Richtigen abweicht; Unrichtigkeit; irrtümliche Entscheidung, Maßnahme; Fehlgriff*[11]

Hieraus geht hervor, dass eine Spezifikation des erwarteten Ergebnisses für die Erstellung eines Testfalls benötigt wird. Andernfalls ist eine Überprüfung des Korrekten nicht möglich. Beispielsweise kann ein Eingabe-Formular nicht überprüft werden, wenn kein Verhalten beim Abschicken der Daten definiert ist. Testfälle dieser Art, sowie deren Aufbau und Gestaltung, dienen immer dem Zweck der Validierung. Dies ist die Erbringung und Darstellung eines Nachweises, dass die vorliegende Implementierung die Anforderungen korrekt der Spezifikation nach umsetzt (vgl. ISTQB und GTB [2016] Seite 69, Validierung).

Des Weiteren muss zwischen 'Funktionalen' und 'Nicht-Funktionalen' Anforderungen separiert werden. Die Prozesse und Tests in dieser Ausarbeitung befassen sich nur mit funktionalen Tests des Trackings. Nicht-Funktionale Anforderungen, wie die Performance der Webseite, richtige Anordnung von GUI Elementen, oder die korrekte Darstellung dieser müssen mit anderen Methoden auf Korrektheit überprüft werden.

Tests können in zwei Kategorien unterteilt werden, auf der einen Seite stehen White-Box Tests (Glass-Box Tests), auf der Anderen die Black-Box Tests (vgl. ISTQB und GTB [2016] Seite 10 & 71). White-Box Tests werden unter Kenntnis des geschriebenen Programmcodes entwickelt, mit der Zielsetzung diesen möglichst umfassend

[11] http://www.duden.de/rechtschreibung/Fehler

abzudecken[12]. Tests werden so entworfen, dass möglichst jede vorhandene Methode ausgeführt und gegengeprüft wird. Parallel ist es dadurch möglich den getesteten Anteil der Codezeilen[13] festzustellen. Der Aufbau eines White-Box Tests leitet sich aus dem Programmcode selbst ab, nicht aus der zu erwarteten Funktionalität. Black-Box Tests hingegen überprüfen nur Funktionalitäten (Ein- und Ausgabedaten) innerhalb der externen Programmstruktur (Nutzerschnittstelle), die nach Konzept oder Spezifikation des Elements zu erbringen sind (vgl. Ricca und Tonella [2001]).

2.3.1 Aufbau eines Webtestfalls

Ein Testfall für eine Web-Anwendung (Englisch *Testcase*) besteht aus dem Aufruf vordefinierter URLs und eventueller Eingabe von Daten - beispielsweise in Formularen (z.B. Login, Registrierung). Basierend auf diesen Eingaben überprüft der Testfall die Serverantwort auf Korrektheit. Dies kann zum Beispiel das reine Anzeigen einer angeforderten Seite, Erhalt einer Datei, zeitnahe Rückmeldung oder erwarteter Netzwerkverkehr sein. Die Überprüfung von Letzterem ist das Ziel dieser Arbeit. Das Wirken bestimmter Ereignisse soll den Versand entsprechender Serveranfragen auslösen - hier Nutzeranalyse- und Verfolgungsmechanismen. Durch den Aufbau diverser Testfälle wird so die komplette Seitenstruktur abgebildet und überprüft. Die Abbildung 2.6 (Siehe Ricca und Tonella [2001] (vereinfacht), Seite 4) präsentiert einen Beispielhaften Aufbau einer kleinen Web-Anwendungen. Einzelne Seiten verlinken zu weiteren Unterseiten, sowie zurück zur Startseite. Der folgende Abschnitt erläutert die Herangehensweise für solch einen Testaufbau mit mehreren Unterseiten und komplexeren Funktionen.

Die initial aufgerufene Webseite **p1** ermöglicht das Laden zwei verschiedener Frames **f1** und **f2**. Diese wiederum verlinken (e_n) jeweils zu **p2**, oder **p3**. In dieser Abbildung enden beide Pfade auf der Seite **p5**. Führt man automatisierte Pfadsuche durch, ist es von essentieller Wichtigkeit zu beachten, dass man keine Schleifen kreiert. Dies

[12] Code Coverage / Test Coverage z.B: http://martinfowler.com/bliki/TestCoverage.html
[13] auch LoC - Lines of Code

wäre der Fall, wenn beispielsweise **p4** zurück auf **p1** verlinkt. Sucht der entsprechende Algorithmus dann erneut nach Verlinkungen, könnte **p2** über **f1** geladen werden. Testfälle die sich auf nur eine Funktion beschränken gestalten sich deutlich simpler, denn in diesem Fall muss keine Rücksicht auf Weiterleitungen oder Kreisverweise genommen werden.

Lediglich das Warten auf eventuell dynamisch ladenden Inhalt ist von hoher Relevanz. Wird dies nicht beachtet, ist es möglich, dass das Testframework schneller agiert als die Webseite. In diesem Fall wird ein Element erwartet, das gerade noch nachgeladen wird.

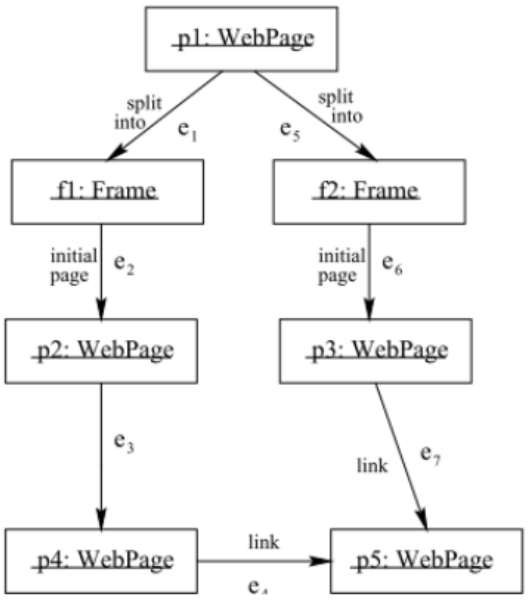

Abbildung 2.6: Pfade einer Webseite - Strukturaufbau

2.3.2 Testlevel

Im Bereich des Softwaretestens existieren folgende Gruppierungen von Testszenarios (vgl. Skumar [2016], Ammann und Offutt [2008] Seite 5):

L1 - **Unittests** sind vom Entwickler geschriebene Testfälle, die einen spezifischen Bereich abdecken. Hierzu zählen Methoden, Funktionen, Interfaces und Klassen. Durch diese Segementierung wird die Funktionalität eines atomaren Bereichs sichergestellt.

L2 - **Komponententests (*Modultests*)** kombinieren mehrere Einheiten zu einer größeren. Hier wird das korrekte Zusammenspiel mehrerer Einheiten / Units überprüft.

L3 - **Integrationstests** kombinieren mehrere Module und testen ihre Funktionalität innerhalb der finalen Umgebung des Programms. Hier muss zwischen verschiedenen Verfahren unterschieden werden. Zum Beispiel können per 'Big Bang' alle Module auf einmal integriert und überprüft werden, es kann aber auch inkrementell 'top down', oder 'bottom up' geschehen.

L4 - **System Integrationstests** testen die Funktionalität benachbarter Systeme innerhalb der gleichen Umgebung. Durch die Integration eines neuen Systems dürfen andere Systeme nicht in der Funktionalität eingeschränkt oder behindert werden.

L5 - **Systemtests** stellen das korrekte Verhalten von Applikationen innerhalb des eigenen Systems sicher.

L6 - **Akzeptanztests** überprüfen den aktuellen Stand mit den Vorgaben des Konzepts oder der Spezifikation.

In Abbildung 2.7 (Siehe Ammann und Offutt [2008] Seite 6) sind die Abschnitte der Softwareentwicklung in Relation zu den Testlevel gesetzt. Es wird der typische, chronologische Verlauf der Softwareentwicklung abgebildet und dargestellt, welches Testlevel zu welchem Abschnitt der Entwicklung durchgeführt wird. Innerhalb von t8y.com wird aktuell auf Basis von fertigen Einheiten getestet. Die vom Entwickler implementierte Funktion durchläuft eigene Unittests. Im Anschluss überprüft die Qualitätssicherungsabteilung auf Abweichungen zur Spezifikation. Jeder Build wird automatisch in einem Continuous Integration Server (zB *Jenkins*) bei Neuerung auf Implementationsfehler

geprüft. Bei einen Buildcrash wird die entsprechende Person direkt benachrichtigt. Für Trackingfunktionen existiert so ein Ablauf noch nicht. Da sich die korrekte Funktionsweise jedoch auch auf Einwirkung Dritter ändern kann, reicht ein Akzeptanztest am Ende der Entwicklungsphase nicht aus. Hier muss in Zusammenarbeit mit außenstehenden Entwickler, Autoren und Analysten eine Lösung gefunden werden. Ein Einfacher Ansatz in Abschnitt 4.1.4 beschrieben, stellt die Unternehmensübergreifende Erstellung von Testfällen dar. Somit wird direkt an der Fehlerentstehungsquelle angesetzt.

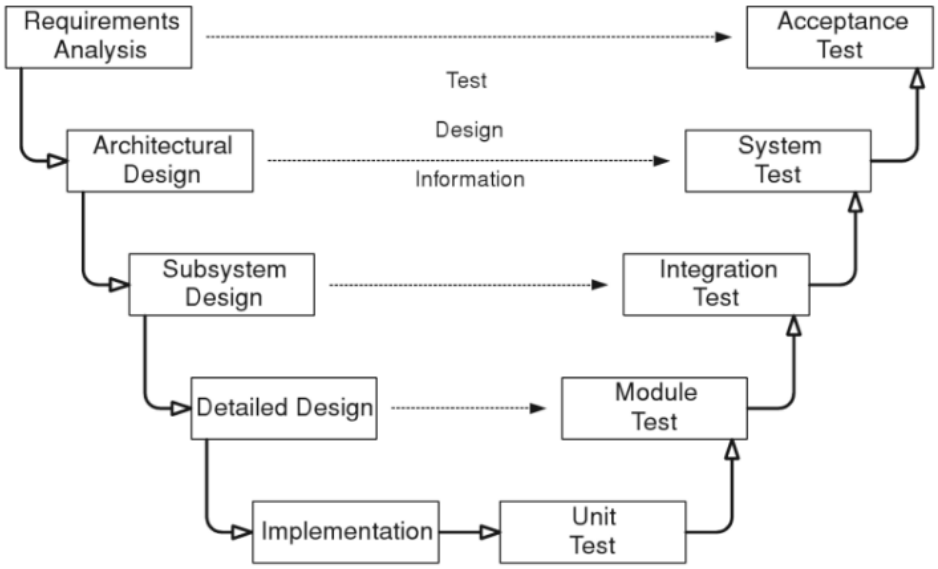

Abbildung 2.7: Softwareentwicklung und Testlevel

Wartungskategorien eines Testfalls

Leotta stellt in (Leotta u. a. [2013c], Seite 274f) zwei Bereiche der Wartung eines Testfalls vor. Logische Veränderungen stehen strukturellen gegenüber, wobei erstere Änderungen an der Funktionsweise des Elements vornehmen und daher zu komplexeren Anpassungen des Testfalls führen. Häufig hängen diese mit sogenannten '**Change Requests**' zusammen. Dies sind nachträgliche Änderungswünsche die so im Vorfeld nicht im Lastenheft festgehalten wurden. Diesen logischen oder funktionalen Ab-

wandlungen stehen die strukturellen gegenüber. Veränderungen an Element IDs oder Änderungen am DOM Baum veranlassen kleinere Anpassungen an Lokatoren. Ein ausgebesserter String der ID oder ein neuer XPath reichen meist aus um den Testfall zu reparieren.

2.3.3 Manuelles Testen

Manuelles Testen von Tracking Requests beläuft aktuell bei t8y.com, in Abschnitt 2.3 kurz erklärten, Black-Box Tests. Der Tester überprüft oberflächlich das Nutzerinterface der Anwendung (Siehe Abschnitt 2.3.2 [L6]) und vergleicht die gebotenen Funktionen mit den Erwarteten. Per Entwicklerkonsole[14] ist es möglich den Netzwerkverkehr oder GUI Elemente genauer zu untersuchen, da so direkt auf die Live-Webseite Einfluss genommen werden kann - per Javascript oder durch direkte Anpassung von Werten. Manuelle Tests sind, bei häufiger Durchführung, zeitaufwändig und bei komplexeren Abläufen schwer zu reproduzieren, denn durch die Komponente der menschlichen Interaktion ist eine weitere Fehlerquelle gegeben. Diese entsteht durch ein hohes Maß an erforderlicher Konzentration - benötigt für stetig korrekte Durchführung immer gleicher Abläufe.

2.3.4 Halbautomatische Testausführung

Das halbautomatische Testen lehnt sich an den Aufbau der Abbildung 2.5 an. Die Eingaben über das UI werden von einem Framework[15] automatisiert (Siehe Kapitel 5). Ein Webdriver übernimmt die Aufgaben des menschlichen Nutzers und tätigt Eingaben. Diese basieren auf im Voraus kreierten Testfällen und programmierten Abläufen. Wie man den Zielstellungen aus Abschnitt 1.4 entnehmen kann, ist das Ziel dieser Arbeit eine halbautomatische Überprüfung des nach Kundenanforderungen implementierten Trackings zu erstellen.

[14] Chrome und Firefox => F12 Taste
[15] z.B Selenium http://www.seleniumhq.org/

2.3.5 Vollautomatische Testausführung

Beim vollautomatischen Testen wird erreicht, dass das testende Framework die Webelemente selbstständig erkennt und auf Basis vordefinierter Parameter prüft. Zum Beispiel sind Login Daten (Benutzername und Passwort) hinterlegt. Findet das Framework ein Loginformular, werden automatisch diese Daten eingetragen und auf Korrektheit überprüft.

Das Verfahren der automatischen Erkennung ist sehr komplex und wird stetig weiterentwickelt und erforscht.

2.4 Verwendete Tools

T1 - Selenium Webdriver[16]

T2 - IntelliJ Idea[17]

T3 - Jenkins[18]

T4 - BrowserMob[19]

T5 - Browserstack[20]

T6 - Chrome HAR Viewer[21]

Als Java Entwicklungsumgebung wurde JetBrains IntelliJ Idea in der Version 15.0 verwendet. Das Modul Selenium Webdriver steuert auf Basis eines Selenium Client Treibers Browser fern. Somit können Nutzereingaben automatisiert nachgebildet und

[16] http://www.seleniumhq.org/projects/webdriver/
[17] https://www.jetbrains.com/idea/
[18] https://jenkins.io/index.html
[19] https://github.com/lightbody/browsermob-proxy
[20] https://www.browserstack.com/
[21] https://ericduran.github.io/chromeHAR

durchgeführt werden.
Die Integration des Browsermob Proxys ermöglicht das Erfassen des HTTP Archivs HAR (Siehe Abschnitt 2.5 und 5.3.1). Dessen Analyse und Abgleich mit vordefinierten Testwerten, wird anhand einer eigenen Implementation durchgeführt. Um die Testdurchläufe nativ auf verschiedenen Systemen und Webbrowsern zu testen, kann zum Beispiel der Service Browserstack genutzt werden (Siehe Abschnitt 6.3). IntelliJ Idea ermöglicht die einfache Integration von jUnit Testfälle und eine direkte Anbindung an ein Git-Repository. Somit werden zielgerichtete Arbeitsabläufe und eine redundante Datensicherung gefördert.

Der 'Chrome HAR Viewer' ermöglicht eine visuell ansprechende Analyse vorhandener HTTP Archive und erleichtert dadurch den Abgleich gegen vorhandene Spezifikationen im Stadium der manuellen Überprüfung.

2.5 http Archiv - HAR

Das HTTP Archiv (.har Dateiendung) schreibt alle HTTP Transaktionen des Browsers in einer Log-Datei mit. Aufbauend auf dem JSON Dateiformat werden die einzelnen Transaktionen in entsprechenden Objekten persistiert. Dieser Export kann nachträglich zur Analyse des Netzwerkverkehrs genutzt werden (vgl. Odvarko u. a. [2012]).

Sourcecode 2.2 entspricht dem Aufbau des kompletten Objekts (vgl. Odvarko [2009], *verkürzt*). Innerhalb von '**log**' befinden sich die wichtigsten Informationen und Werte. '**Version**' repräsentiert die Versionsnummer des '**har**' Formates selber. '**creator**' enthält Angaben zum Herkunftsort der Datei. Innerhalb von '**browser**' befinden sich weitere Informationen zum ausgeführten Browser und dessen Version. '**pages**' verkörpert ein Array der exportierten Seiten. Diese werden dargestellt durch eine id, ihren Seitentitel. Des Weiteren werden Informationen zum Aufrufzeitpunkt festgehalten. Wenn vorhanden, können zusätzliche Daten zu Ladenzeiten von Webseiten inkludiert werden. Innerhalb des Eintrags '**entries**' befinden sich alle mitgeschnittenen HTTP-Requests. Abgebildet sind hier die jeweils ersten Datensätze für '**request**' (*Anfrage*)

und '**response**' (*Antwort*). Es existieren noch zusätzliche Informationen über etwaige Header, Cookies oder Rumpfgröße. Diese sind an dieser Stelle aber nicht weiter wichtig.

Auf Basis dieser gesammelten Informationen lassen sich präzise Rückschlüsse über den Ereignisverlauf ziehen. Alle relevanten Daten können ausgelesen, verglichen und ausgewertet werden. Anhand der Häufigkeit auftretender Requests könnte man beispielsweise die Beliebtheit eines Produkts oder Downloads bewerten.

```
{
"log": {
        "version" : "1.2",
        "creator" : {
                "name": "BrowserMob Proxy",
                "version": "2.1.0-beta-4-littleproxy"
        },
        "browser" : {
                "name": "Firefox",
                "version": "45.0",
        },
        "pages": [{
                "startedDateTime": "2009-04-16T12:07:25",
                "id": "page\_0",
                "title": "Test Page",
                "pageTimings": {}
                }
        ],
        "entries": [{
           "pageref": "visitHomepage(com..tests.homepage)",
           "startedDateTime": "2016-06-22T12:55:11",
           "request": {
                   "method": "GET",
                   "url": "http://homepage.com",
                   "httpVersion": "HTTP/1.1"
           },
            "response": {
                    "status": 200,
                    "statusText": "OK",
                    "httpVersion": "HTTP/1.1"
            }
         }],
        "comment": ""
        }
}
```

Sourcecode 2.2: HAR - Grundlegende Struktur

3 Literaturanalyse

In diesem Kapitel werden ältere und aktuelle Forschungsarbeiten im Rahmen der vorliegenden Thematik vorgestellt und in Kontext gesetzt. Betrachtet werden Ausarbeitungen der Themen 'Testing', 'Testfall Generierung' und 'Analytics'.

Nach bestem Wissen existiert aktuell kein Programm, das automatisiert Trackingimplementierungen erkennt und entsprechende Testfälle generiert. Es existieren diverse Paper zur automatischen Generierung von Testfällen. Die Integration von Tracking innerhalb dieser wird dabei noch nicht beachtet. Nach aktuellem Wissensstand ist dies ein Prozess, der im Nachhinein manuell eingebaut werden muss. Parallel existieren Firmen (Siehe Abschnitt 4.4), die diese Aufgaben erledigen. Es ist nicht bekannt, ob jegliche Prozesse manuell umgesetzt werden, oder ob diese Unternehmen ein automatisches oder halb-automatisches Verfahren entwickelt haben.

Das Thema der automatisierten Überprüfung von Tracking Request, sowie all-in-one Webseitenaudits, ist zum Entstehungszeitpunkt dieser Arbeit noch nicht wissenschaftlich erörtert. Einen ersten Ansatz für eine umfassendere Testsuite präsentiert Jan Exner auf seinem Blog[1]. Exner bietet ein Framework zur Erstellung von Testfällen auf Basis häufig anzutreffender Funktionen. Jeder Testfall ist im JSON Format aufgebaut. Über diverse Parameter können die zu testende URL und Elemente definiert werden. Die Suite nutzt den Kommandozeilenbrowser PhantomJS[2] zur schnellen und effizienten Ausführung, denn eine grafische Wiedergabe des Testdurchlaufs ist nicht nötig. Der Ansatz ist ähnlich dem dieser Arbeit. Die Kreation und Ausführung von Testfällen

[1] https://webanalyticsfordevelopers.com/2016/06/07/wanna-test-heres-how
[2] http://phantomjs.org

ist so simple wie möglich gestaltet, damit auch nicht Programmierer diese Aufgabe übernehmen können.

Des Weiteren ist zu erörtern, inwiefern diese Geradlinigkeit und Einfachheit der Testfallerstellung erreicht werden kann. Leotta stellt zwei Ansätze in Leotta u. a. [2013c] gegenüber. Auf der einen Seite das Capture-Replay Verfahren, jenes entspricht einem Testroboter der im Vorfeld aufgezeichnete Abfolgen automatisch abspielt. Dies wird mit den klassisch, programmierten Testfällen hinsichtlich Effizienz und Zeitaufwand verglichen. Capture Replay Verfahren brechen bereits bei kleineren Anpassungen an der Webseite. Programmierte Tests sind in der Wartung im Mittel 33% leichtgewichtiger, brauchen aber zwischen 30% und 112% länger in der initialen Erstellung. Final wird festgehalten, dass je länger die Testfälle benötigt werden und je mehr Revisionen die Entwicklung durchwandert, programmierte Tests deutlich effizienter werden. Dies fällt insbesondere bei Web-Applikationen ins Gewicht, da diese sich in einem schnellen Wandel befinden.

In Leotta u. a. [2013a] untersucht Leotta den für die Wartung von Testfällen nötigen Zeitaufwand auf Basis verschiedener Elementzeigerverfahren (Siehe Abschnitt 5.2). Das Ergebnis sagt aus, dass die Elementzugriffsvariante per ID, wenn vorhanden, am wenigsten Zeitaufwand in nachträglicher Wartung erfordert. Andernfalls bieten sich XPath Pfade an. Des Weiteren wird erneut festgehalten, dass die Verwendung und Integration des 'Page Object Patterns' eine längerfristig hilfreiche Abstraktionsebene schafft, denn es entkoppelt Webseite und Testfälle. Testfälle werden somit stabiler.

In Leotta u. a. [2015] hat Leotta auch die automatische Generierung von Testfällen auf Basis einer Bilderkennung untersucht. Eventuell lässt sich dieser Ansatz um die nachträgliche Integration von Trackingabfragen und Überprüfung jener erweitern. Im Grunde entspräche dies den im Fallbeispiel 5.3 umgesetzten Methoden. Der Testaufbau auf Screenshots unterstützt jedoch nicht das Testen von Tracking, da entsprechende Mechaniken unsichtbar für den Webseitennutzer ablaufen. Fraglich ist, ob dies einen zeitlichen Vorteil ermöglicht. Andernfalls reicht die manuelle Erstellung von Testfällen aus. Basierend auf wenigen Fällen, die die Grundfunktionen umspannen, kann der restliche Anteil abgeleitet werden.

Xun arbeitet im Paper Xun [2015] die Wichtigkeit von Analytics Daten im Zusammenhang mit Kundenbindung heraus. Anhand diverser Leistungskennzahlen (Englisch: *Key Performance Indikatoren*) wird dieser Faktor für Webseitenbetreiber dargestellt. Eine längere Besuchszeit führt direkt zu stärkerer Finanzperformance des angeboten Produkts. Die höchste Steigerung wird durch das Verlinken auf Seiten Dritter und der Integration sowie Distribution von Empfehlungslinks (Englisch: *Affiliatelink*, *Referrallink*) generiert. Diese Arbeit stellt nur ein Beispiel für die erhöhte Relevanz korrekter Nutzeranalysedatensätze dar. In (Hassler [2012], Seite 25-30) unterstützt Hassler dieses Ergebnis um weitere Analogien. Zum Beispiel könnte ein Zeitungsverlag den Rückgang von Lesern und dessen Ursachen erst nach Wochen bemerken, da ein direkter Rückmeldungs- und Meinungskanal für die Massen fehlt, denn im Internet ist das Verhalten eines Besuchers und Nutzers genau analysierbar. Selbst minimale Veränderungen können zeitnah bemerkt werden und bilden aussagekräftige Standpunkte.

Die steigende ökonomische Relevanz von Internet Applikationen fördert die Nachfrage nach qualitativ hohen und inhaltlich korrekten Datensätzen. Aus diesem Grund ist eine hohe Anzahl von Testfällen vorteilhaft für die Validierung des Produktes und Vereinfachung von Regressionstests. Ricca präsentiert diesen Ansatz in Ricca und Tonella [2001] und entwickelt ein UML-Modell dieser Applikationen auf einer hohen Abstraktionsebene. Außerdem wird ein Ansatz der Testfallgenerierung auf Basis von Linkpfaden vorgestellt. Auch hier muss der Entwickler im Anschluss Werte auf Basis der Spezifikation händisch in die Testfälle eintragen.

4 Anforderungsanalyse

Durch eine Analyse der Ist-Situation werden Problematiken herausgearbeitet und anschließend im Rahmen eines Soll-Konzepts mit Lösungsansätzen erweitert. Des Weiteren werden die Anforderungen erläutert, und eine kurze Marktanalyse stellt die aktuellen, proprietären Angebote dar.

4.1 Systemanalyse

Die Systemanalyse präsentiert die aktuelle Ist-Situation inklusive Schwachstellen. Nachfolgend stellt eine Potential- und Risikoanalyse mögliche Veränderungen in den Fokus. Im Anschluss wird ein Soll-Konzept dargelegt. Dies nimmt sich der Probleme an, integriert erste Lösungen und gibt eine Richtlinie vor.

4.1.1 Ist-Analyse

Das Unternehmen t8y.com GmbH (Siehe Abschnitt 1.2) entwickelt und pflegt Webseiten nach Kundenanforderung. Die Implementierung tiefgreifender Nutzeranalysemechanismen auf Basis aktueller Analysesoftware (zum Beispiel: Google Analytics[1], Adobe Analytics[2]) ist Teil des Kerngeschäftes. Wie (Heinrich [2007], Seite 25ff) beschreibt, ist die Informationsgewinnung das primäre Ziel einer Systemanalyse. Auf Basis dieses

[1] https://www.google.com/intl/en_en/analytics
[2] http://www.adobe.com/de/marketing-cloud/web-analytics.html

gewonnenen Grundlagenwissens können fundierte Entscheidungen getroffen werden. Gesammelt werden diese Informationen überwiegend über vorliegende Dokumente oder durch allgemeine Bekanntheit innerhalb des Kollegiums. Bei der Planung des neuen Systems und dessen Einführung müssen einige Ansatzpunkte beachtet und bedacht werden. Zu diesen gehören unter anderem, aber nicht ausschließlich, die entstehende, finanzielle Entwicklung und entsprechende Auswirkungen, sowie mögliche neue, oder überflüssige Arbeitsplätze. Des Weiteren muss immer der damit einhergehende Kompetenzverlust oder Kompetenzgewinn in Betracht gezogen werden.

Abbildung 4.1: Akueller Arbeitsablauf

Um Fehler zu minimieren wird konstant getestet. Dies wird durch die Entwickler selbst realisiert, sowie von einem eigenen QS Team parallel begleitet. Die Programmierer kümmern sich weitestgehend um Komponenten und Systemtests, während das dedizierte QS Team kontinuierliche Akzeptanztests und Regressionstests durchführt. Trackinganforderungen, auf Basis des Trackingkonzepts des Kunden, werden zum Stand vor dieser Arbeit manuell während der Implementierungsphase und exemplarisch zur Laufzeit überprüft. Mögliche Fehlfunktionen, selbst verschuldet oder durch externen Code (Siehe Abschnitt 2.2.1) ausgelöst, werden häufig erst verspätet bemerkt. In Abbildung 4.1

wird dieser Ablauf verdeutlicht. Eine Menge von Webseitenentwicklern implementiert die grundlegenden Funktionen und das Design nach Auftragswunsch. Das Testing wird nach obigem Prinzip durchgeführt. Durch die zusätzliche Komponente des Tag Managers muss die Implementierung der Trackingverfahren geklärt werden. Eine doppelte Einbindung ruft eine weitere Fehlerquelle hervor, da sie sich gegenseitig behindern können. Es gibt keine Entität die sich speziell um Trackinganforderungen kümmert. Dieser Aspekt wird erst auf Nachfrage bearbeitet.

Die Kommunikation mit entsprechenden Kunden findet überwiegend per Zwischenstelle Projektmanager statt. Dieser übernimmt die Aufgabenkoordination, Zeitplanung und verwaltet Rückmeldungen sowie Kritik. Entsprechende Fehler werden dokumentiert und an Entwickler weitergegeben. Auf Basis einer Fehleranalyse wird die Lösung entworfen und umgesetzt. Im Falle von Fehlern im Tracking wird das Auftreten nachvollzogen und manuell erneut überprüft. In den meisten Fällen führten lediglich kleinere Änderungen im Aufbau des *Document Object Model*, der Struktur von HTML Elementen einer Webseite, zur inkorrekten Funktionsweise des Tracking. Wird zum Beispiel die Element-Id verändert, ist der direkte Zugriff nicht mehr möglich. Weitere Informationen befinden sich im Abschnitt 5.2. Der Kunde arbeitet aktuell nicht direkt am Produkt mit, er stellt die Spezifikation und Ideen.

4.1.2 Potential- und Risiko-Analyse

Die Potential-Analyse stellt aktuelle Schwachstellen und Verbesserungsmöglichkeiten dar. Diese sind Teil eines oder mehrerer der folgenden Prozesse (vgl. Heinrich [2007], Seite 46):

a - Aufbauorganisation

b - Ablauforganisation

c - Informationsgewinnung oder -verteilung

d - Hardware-, Software- oder Netzwerksysteme

e - Kommunikation und Beteiligung

Eine Aufbauorganisation beschreibt die Struktur eines Unternehmens. Inbegriffen sind Geschäftsziele, Aufgabendefinitionen und Organisationseinheiten. An dieser Stelle soll hier aber nicht weiter drauf eingegangen werden. Lediglich die Aufteilung in Teams und damit einhergehende Aufgabenverteilung ist teil-relevant, denn aktuell existiert keine Verantwortungsposition für Tätigkeiten im Tracking Bereich. An dieser Stelle würde die Verlagerung der Kommunikationsleitung zu einem QS-Mitarbeiter den Ablauf gradliniger und strukturierter gestalten. Auftretende Probleme können direkt an die Person mit Wissen weitergeleitet werden. Der Umweg über eine zentrale, verwaltende Stelle in Form eines klassischen Projektmanagers führt hier zu Mehrarbeit.

Die Ablauforganisation beschreibt Aufgabenketten, Prozessreihenfolgen und Vermittlung zwischen Mitarbeitern und Kunden. Dieser Punkt hängt mit dem eben angeschnittenen Aufbau zusammen und ist einer der Kernbereiche dieser Arbeit. Nachfolgend wird ein angepasster Testprozess dargestellt und erläutert. Abweichend von der aktuellen Durchführung wird dadurch der Kunde deutlich enger in den Bearbeitungsprozess der Testerstellung und Testsuiteausführung eingebunden. Diese Arbeit beschränkt sich auf die Analyse und Erweiterung von Prozessen innerhalb der Testphasen.

Wie in Abschnitt 1.4 Zielsetzungen formuliert, streben wir nach einem Testkonzept für das Tracking, dass von Abbildung 4.2 (Siehe Sommerville [2010] Kapitel 2.) abweicht. Der Ablauf des Testens von Tracking Implementierungen soll abstrahiert vom Wasserfallkonzept nicht mehr linear am Ende des Zyklus stattfinden, sondern durch vereinfachte Strukturen regelmäßig und früher durchgeführt werden. Das im Verlauf dieser Arbeit entwickelte Konzept unterstützt diesen Plan und vereinfacht den Ablauf.

Das Wasserfallmodell beschreibt einen linearen Ablauf in der Softwareentwicklung. Die einzelnen Phasen laufen abschließend direkt ineinander über. Initial werden die Anforderungen in einem Lasten- und Pflichtenheft festgelegt. Auf Basis dieser Einzelheiten wird das System geplant und strukturiert. Darauf folgend findet die Implementation dieses Produktes statt. Einem funktionalen Unit-Testing einzelner Einheiten folgt die Überprüfung des Zusammenspiels mehrere Komponenten bis das System schließlich

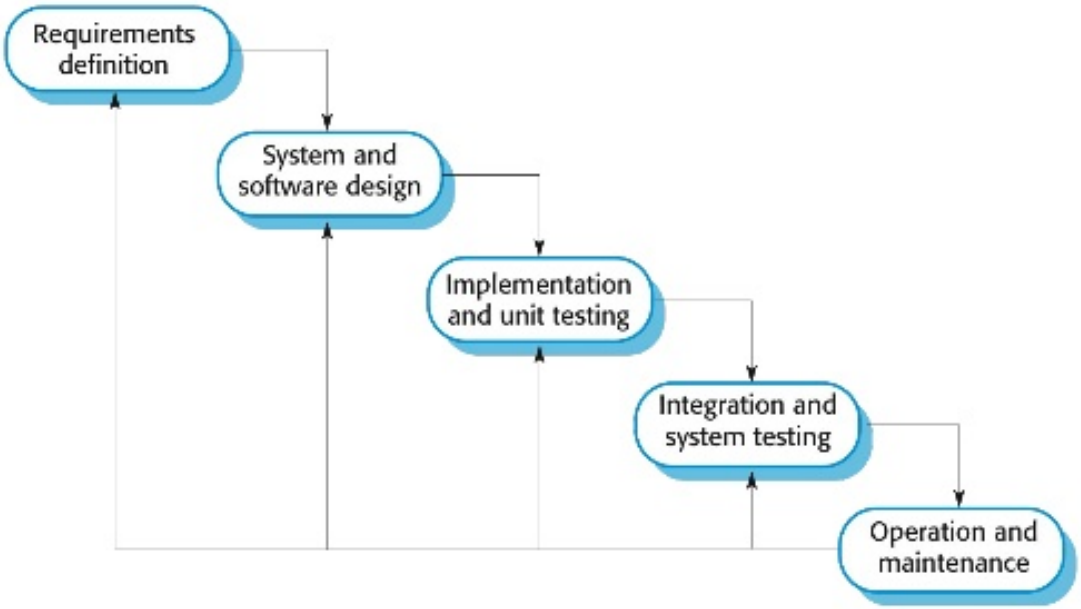

Abbildung 4.2: Softwareentwicklungszyklus

in den *Live* Zustand (zB. die Veröffentlichung einer Webseite im Internet) übergeht und dem eigentlich Zweck eröffnet wird. Das Wasserfallmodell eignet sich aus diesen Gründen für Systeme, die sich sehr genau und präzise im Vorfeld gestalten lassen.

Wenn das Tracking nach diesem Schema implementiert und getestet wird, entstehen zwangsläufig Fehler, da die Trackingimplementation abhängig von weiteren Komponenten ist. Bereits kleinere Änderungen an der Seitenstruktur oder Funktionalität können Fehler hervorrufen. Zusätzlich lässt sich aus Erfahrung sagen, dass der Kunde häufig im Verlauf der Entwicklung Änderungen vornimmt oder detaillierte Daten sammeln möchte.

Auf Grund von obigen Erläuterungen, ist die Ausgliederung der Entwicklung und Durchführung von Testfällen für Trackingmechanismen die logische Konsequenz. Da der Kunde die Spezifikation erstellt, würde sich die zeitgleiche Definition eines Testfalls positiv auf die Längerfristig korrekte Funktionalität auswirken, denn bei einem Fehlerszenario würde das Testergebnis eindeutige Rückschlüsse bieten.

4.1.3 Risiko

Die Restrukturierung und Verlagerung von Aufgabenteilbereichen auf Kundenseite stellt einen Verlust von Wissen und Kontrolle dar. Zusätzlich wird hierdurch das Einkommen reduziert, da der Kunde weniger Leistung in Anspruch nimmt. Gleichzeitig wird jedoch das Arbeitsvolumen reduziert und die Entwickler können sich auf die eigentliche Entwicklung der Kernfunktionalitäten konzentrieren. Des Weiteren ist der Kunde ab jetzt zu einem gewissen Teil mitverantwortlich. Nicht jeder Kunde möchte dies. Eventuell könnte bereits der Ideenvorschlag auf starken Widerspruch und Abneigung stoßen. Längerfristige Folgen lassen sich hier nicht vorhersehen. Im schlimmsten Fall wechselt der Kunde den Anbieter.

Einigt man sich auf eine geteilte Aufgabenstruktur und wendet das vorgeschlagene Konzept aus Abschnitt 4.1.4 an, besteht die Möglichkeit des stark erhöhten Kommunikationsaufwandes bei Fehlschlägen auf Kundenseite. Eventuell fehlen die technischen Voraussetzungen oder Qualifikationen der Mitarbeiter. In diesem Fall müssten wiederum Entwickler für Aufgaben einspringen, von denen sie eigentlich entlastet werden sollten.

Automatisierung

Der Entwickler muss für jedes zu testende Trackingereignis einen separaten Unit Test implementieren. Auf Basis des *Page Object Pattern* (Siehe Kapitel 5.1) ist es möglich, den Code so zu gestalten, dass möglichst viel wieder verwendet werden kann. Beispielsweise entspricht die Aufnahme und Analyse des Anklickens eines Links auf Webseite A weitestgehend dem Vorgehen für Webseite B. Somit kann durch Verwendung des Page Object Pattern die Testfallgestaltung unkompliziert gehalten werden.

Die entstandene Testsuite kann im Verlauf stetig erweitert und angepasst werden. Durch ein Deployment auf einen Continuous Integration Server (zB Jenkins) wird eine regelmäßige Durchführung ermöglicht. Die hierdurch entstehende Automation ersetzt den manuellen Überprüfungsprozess einzelner Parameter und dessen Ereig-

nisauslösung. Die Wartung bei Page Object Tests basiert im wesentlichen auf der Korrektur fehlerhafter Elementids oder XPath Verweisen. Im Paper von Leotta (Leotta u. a. [2013b]) werden beiden Möglichkeiten an Hand einer Testsuite verglichen. Setzen Entwickler konsequent IDs für HTML Objekte, ist die Lokalisierung auf Basis dieser vorzuziehen.

4.1.4 Soll-Konzept

Aufbauend auf der Ist-Analyse stellt das Soll-Konzept erste Lösungsansätze und fortführende Ideen dar. Nachfolgend wird ein veränderter Arbeitsablauf und eine effizientere Kommunikationsstruktur präsentiert. Abbildung 4.3 bildet den Ablauf aus der Ist-Analyse in verbesserter Form.

Der Prozess der Testfallerstellung beginnt beim Input der Spezifikation (Siehe Abschnitt 4.5), geht weiterführend über den Aufbau des expliziten Testfalls und endet bei der Auswertung präsentierter Ergebnisse. Die Trackingspezifikation sollte in einer vordefinierten, maschinenlesbaren Struktur existieren. Dadurch kann sie direkt analysiert und verarbeitet werden. Dies beinhaltet das automatische Erkennen entsprechender Anfrageparameter (Siehe Sourcecode 2.1). Der Entwickler bildet die Webseitenstruktur und dessen Funktionen als Page Objects ab. Hierauf aufbauend ist es weiteren Parteien möglich, Testfälle zu erstellen. Diese Aufgabe kann auf Grund des Page-Object-Patterns(Siehe Abschnitt 5.1) auch von Personen mit geringeren Programmierkenntnissen durchgeführt werden.

Die entstandenen Testfälle können, solange am Testobjekt nichts verändert wurde, automatisiert und regelmäßig durchgeführt werden. Dies ermöglicht die Früherkennung von Fehlern und verbessert dadurch implizit die Codequalität. Des Weiteren wird der Aufwand des manuellen Testens vom QS Team deutlich reduziert. Die hiermit geschaffenen Ressourcen können anderweitig eingesetzt werden. Durch konsequentes und präzises Reporting ist eine genaue Analyse der Testergebnisse möglich.

Die gewonnenen Analysedaten sind somit verlässlich und ermöglichen dem Analyst präzise, statistische Auswertungen. Außerdem können eindeutig Aussagen zur Funktionalität der implementierten Trackingmechanismen getroffen werden.

Abbildung 4.3: Arbeits- und Kommunikationsablauf

Zusätzlich ist zu beachten, dass die regelmäßig durchgeführten Testfälle nicht die Live Statistiken im Analyticssystem beeinflussen. Durch Maßnahmen wie die Blockierung bestimmter IPs, kann hier entsprechend gefiltert werden.

Arbeits- und Aufgabenteilung

Wir bereits angedeutet, ist die Aufteilung von Arbeitsschritten und Verantwortlichkeiten ein erheblicher Faktor in der benötigten Zeit zur Implementierung von Tests, sowie der Korrektur fehlerhafter Funktionen. Übernimmt der Entwickler den Aufbau der Webseite und legt gleichzeitig das Test-Grundgerüst in Form von Page Objects an, während die Implementation und Durchführung von Testfällen auf Seiten des Kunden und Analysten stattfindet, herrscht eine strukturierte Aufgabenteilung mit klareren Verantwortungsbereichen. Zusätzlich steht den Entwicklern mehr Zeit zu Entwicklung

und Verbesserung der Kernfunktionalitäten zur Verfügung, denn die Überprüfung und ein Teil der Tracking Implementation entsteht extern. Ideal ist es, wenn eine Partei die komplette Integration von Trackingmechanismen übernimmt. Geschieht dies rein über einen Tag Manager kann dieser Aspekt inklusive Testing komplett ausgelagert werden. Aktuelle Möglichkeiten sind hier die Dienstleistungen von Firmen wie Columbo. (Siehe Abschnitt 4.4.2).

Abbildung 4.4 stellt die Aufgabenteilung in Form eines Use Case Diagrammes dar.

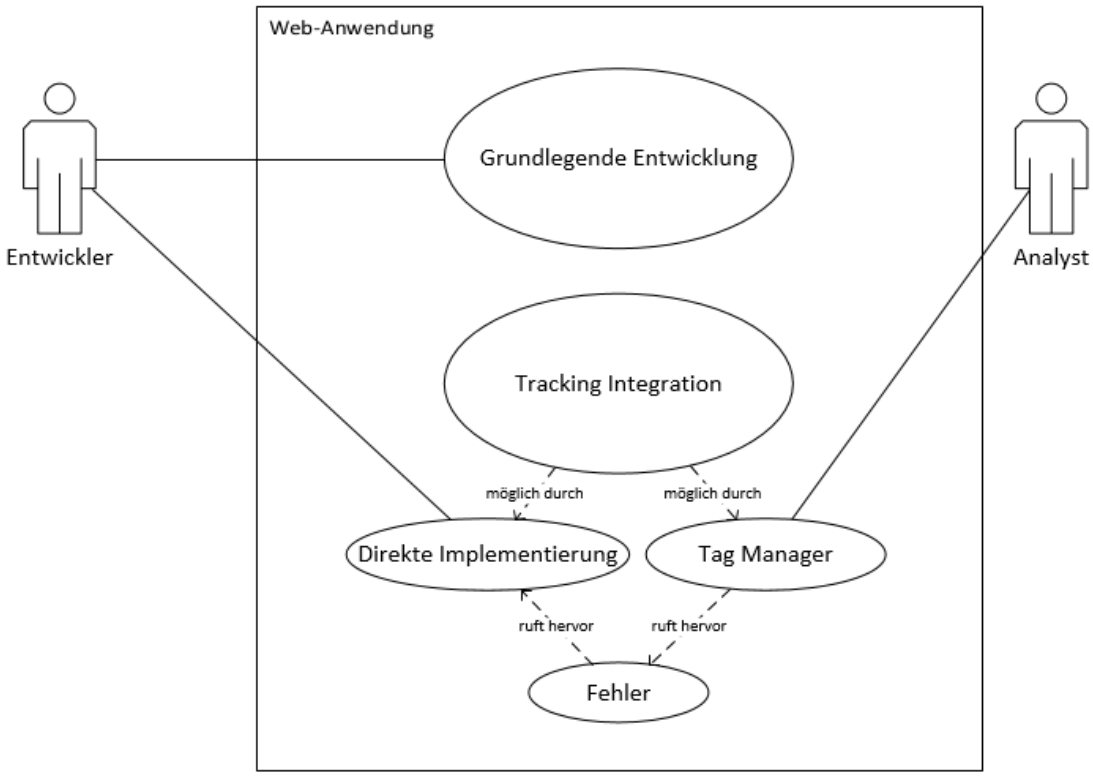

Abbildung 4.4: Use Case Aufgabenteilung

Der Akteur Entwickler ist für die Entwicklung der Webseite zuständig. Gleichzeitig übernimmt er anfallenden, direkte Implementierung von Trackingintegrationen im Code. Hierunter fällt die Einbindung der Basistags oder, wenn gefordert, vom Tag Manager unabhängige Lösungen. Der Akteur Analyst ist für die Benutzung des Tag

Managers zuständig. Durch die Verwendung der externen Komponente Tag Manager werden Fehler hervorgerufen die in den Aufgabenbereich des Entwicklers fallen. Aus diesem Grund ist ein ausgebautes Testsystem notwendig.

Abbildung 4.5 bildet die Aufteilung der Testfallgenerierung ab.

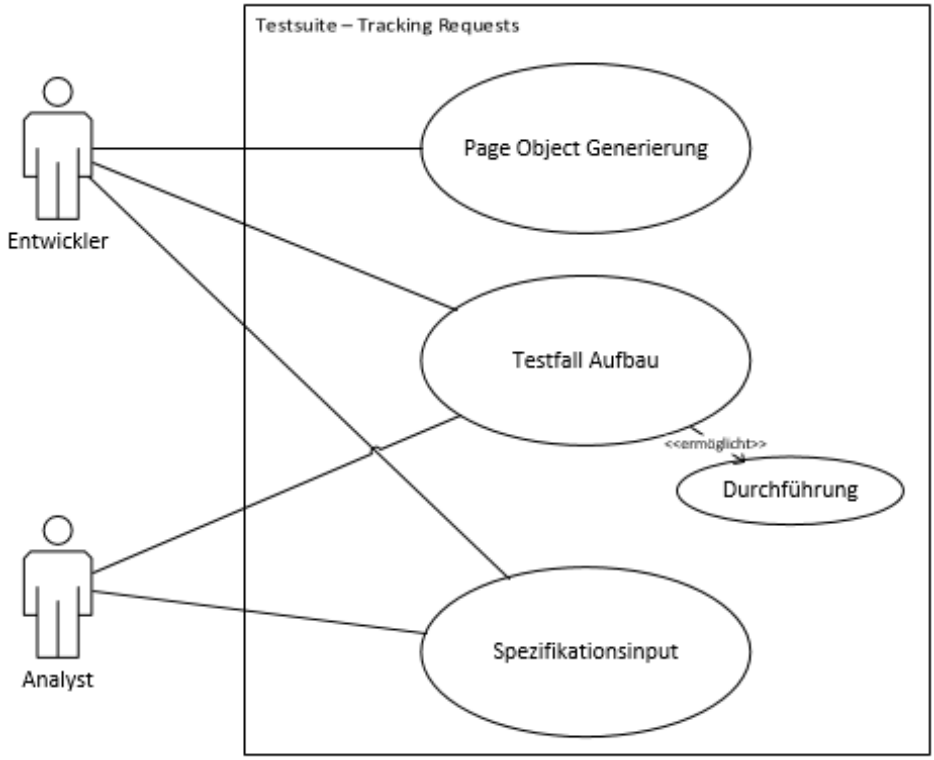

Abbildung 4.5: Use Case Testaufbau

Der Akteur Entwickler übernimmt die Erstellung der Testfallbasis inklusive Page Objects. Der Akteur Analyst und sein Arbeitgeber ist Ersteller der Trackingspezifikation. Auf Basis der vom Entwickler geschaffenen Dateninputmöglichkeit kümmert sich der Analyst um die Erstellung von Testfällen die auf diesen Datensätzen aufbauen. Abbildung 4.4 legt die Aufgabenteilung inklusive Tag Manager dar. Ausgehend von dieser Grundlage entwickelt der Analyst Testfälle entsprechend seiner Spezifikation und umgesetzter Tags im Tag Manager. Weitere anfallende Testfälle werden vom Entwickler übernommen.

4.2 Funktionale Anforderungen

Da diese Arbeit in großen Teilen ein Konzept und verbesserte Arbeitsabläufe beschreibt, gibt es nur wenig funktionale Anforderungen zu definieren. Das System soll alle integrierten Testfälle automatisiert, parallel oder sequentiell ausführen. Im Anschluss an einen Durchlauf müssen die Ergebnisse gespeichert und für eine Auswertung bereitgestellt werden. Zusätzlich muss das System diesen Ablauf automatisch anstoßen und ohne menschliche Einwirkung wiederholt durchlaufen. Somit ist die stetige Überprüfung implementierter Funktionen gegeben.

4.3 Nichtfunktionale Anforderungen

Im Gegensatz zu den mess- und greifbaren funktionalen Anforderungen beschreiben die nichtfunktionalen Anforderungen zu erbringenden Werte und Qualitäten des Systems.

Benutzbarkeit

Die Interaktion mit dem gesamten System, sowie Implementation einzelner Testfälle soll so einfach und primitiv wie möglich sein. Ziel ist, dass auch ein nicht dedizierter Entwickler mit kurzer Einarbeitungsphase in der Lage ist, eigene Testfälle nach Vorgabe zu implementieren. Auf Basis der Page Objects und vergangener Beispiele, können diese Tests in den meisten Fällen analog aufgebaut werden. Bei komplizierteren Trackingfunktionen, beispielsweise der Dauer einer Videowiedergabe, oder aktiv auf der Seite verbrachten Zeit, kann hier ein Entwickler unterstützen. Die Integration der Testsuite in einen Service wie Jenkins, erlaubt eine automatische Ausführung, sowie die Überwachung per Weboberfläche. Erste Fehler oder Konsoleausgaben lassen sich bereits komfortabel erfassen. Zusätzlich ist die Benachrichtigung bei Ereignissen per E-Mail möglich.

Wartbarkeit

Fertiggestelle Testfälle für Webfunktionen basieren stark auf der Implementation eben dieser. Selenium Webdriver ist auf den, zum Zeitpunkt der Erstellung aktuellen, Seitenaufbau angewiesen. Wie in Abschnitt 5.2 beschrieben, gibt es mehrere Methoden um auf Elemente zuzugreifen. Diese können leicht vom Webentwickler verändert, verschoben oder umbenannt werden. In diesem Fall ist ein Anpassen des Testfalls nötig. Page Objects sind durch den Gestaltungsstil einfach zu verstehen, aus diesem Grund ist die Anpassung kleinerer Änderungen nur von minimalem Aufwand.

Korrektheit

Das primäre Ziel einer Testsuite ist die Überprüfung gewählter Funktionen. Diese Testfälle müssen die gewünschten Ereignisse mit Input versehen und den Output gegen eine Spezifikation vergleichen. Stimmen beide überein, ist funktioniert das Testobjekt korrekt. Zu beachten ist, dass der Testfall alle Input Möglichkeiten in Betracht zieht. Wenn dies nicht der Fall ist, kann nicht von Funktionskorrektheit ausgegangen werden. Beispielsweise muss ein Datumsfeld mit Werten umgehen können, die nicht dem erwarteten Datumsformat entsprechen. Ein entsprechender Testfall würde dieses Feld mit verschiedenen Inputs befüllen und die Antworten verarbeiten, sowie entsprechende Ereignisse und Log-Ausgaben speichern.

Flexibilität

Eine zentrale Anforderung dieses Konzepts und des Testsystems ist die dynamische Aufgabenteilung und erweiterte Absprache mit externen Parteien. Die Testfallerstellung und Wartung soll zwischen dem Webseitenentwickler und Analysten aufgeteilt werden. Durch die Verwendung von Teststandards wie dem Page Object Pattern, jUnit und Selenium Webdriver ist eine große Community gegeben. Es existieren viele Hilfestellungen, 'Frage und Antwort' Foren, sowie Support durch andere Entwickler.

4.4 Marktanalyse

Es gibt diverse Dienstleister und Produkte am Markt, die sich der Problemstellung des Testens von Nutzungsanalyseverfahren angenommen haben. Um sich einen Überblick der aktuellen Lage zu verschaffen und Lösungen, sowie Herangehensweisen Anderer zu vergleichen, ist es wichtig, sich im Klaren über die Problemstellung und der gewünschten Ergebnisse zu sein. Ein manueller Testaufbau besitzt einen gewissen systemübergreifenden Basisarbeitsaufwand.

In den folgenden Unterkapiteln werden drei Dienstleister und deren Produkte genauer betrachtet und auf mögliche Lösungswege hin untersucht.

4.4.1 ObservePoint

Alle folgenden Informationen wurden einer Produktdemo, durchgeführt am 9.3.2016 von Jacob Sohn (Director - EMEA bei ObservePoint), sowie der Firmen Webseite[3] entnommen.

ObservePoint bietet einen automatischen Webseiten Prüfungslauf / Audit an. Dieser untersucht die angegebenen Seiten auf eventuelle Fehler und prüft die Antwortzeiten auf Anfragen. Des Weiteren kann man gezielt nach gesetzten Tags, Variablen und fehlenden Werten suchen. Außerdem werden fehlende Daten und zugewiesene Werte protokolliert.

Zusätzlich können Editoren eigene Tests manuell anlegen. Element Lokatoren (Siehe Abschnitt 5.2) müssen hier händisch identifiziert und gepflegt werden. Bei anstehenden Wartungen oder Fehlerverbesserung muss jeder entsprechende Test gezielt angepasst werden. Eine Funktion des aus Entwicklungsumgebungen (IDE) bekannten Code Refactoring existiert nicht. Ebenso gibt es keine testübergreifenden Variablen, die eine Wartung ermöglichen. Zusätzlich ist das Hinzufügen von kurzen jQuery[4] Anweisungen,

[3] http://www.observepoint.com/
[4] https://jquery.com/

sowie die Umleitung des Netzwerkverkehrs über einen eigenen Proxyserver möglich. Abschließend ermöglicht ObservePoint die ausführliche Generierung von Berichten in diversen Exportformaten.

Zusammenfassend bietet ObservePoint viele Komfortfunktionen und ein praktisches Berichtswesen. Die Kernfunktionalität der Testerstellung unterscheidet sich aber nicht wesentlich von der eigenen Implementation durch jUnit[5] Tests. ObservePoint bietet einen 'Proof Of Concept' Audit um das Angebotene zu testen. In der Nutzung ist das Softwareangebot für bis zu 3.000 Pagetests gratis. Benötigt man umfangreichere Möglichkeiten, kann man für USD10.000 im Jahr eine Partnerlizenz erwerben. ObservePoints weiteres Preismodell wird je nach Kunde individuell gestaltet.

4.4.2 Columbo.io

Columbo[6] ermöglicht das Testen von Webanalyse Daten. Die Kernfunktionalität umfasst die Erstellung und Verwaltung separierter Tracking Tests. Außerdem bietet Columbo einen Scanner der mit minimaler Konfiguration Webseiten überprüft und bei fehlendem Tracking oder falschen Informationen entsprechend informiert. Szenarien hingegen stellen eine komplexere Abfolge von Befehlen dar. Der Crawler von Columbo führt im Vorfeld angelegte Schritte sequentiell aus. Mögliche Beispiele sind ein Seitenaufruf, Erstellen eines Screenshots oder die Ausführung eines im Vorfeld erstellten Testfalls. Durch Einbindung dieser Testfälle ist die präzise Auswertung gesetzter Trackingparameter möglich.

Columbo bietet zusätzlich die Möglichkeit des kompletten Testmanagements an. Der Kunde teilt Columbo mit, welche Seiten getestete werden müssen und wie die Spezifikation hierzu aussieht. Columbo übernimmt dann die Testfallerstellung und regelmäßige Überprüfung, sowie entsprechende Wartung. Die Preisgestaltung wird hier individuell mit dem Kunden erarbeitet.

[5] http://junit.org/junit4/
[6] https://columbo.io/

Columbos Preismodell basiert auf der Anzahl von Scannereinsätzen im Monat. Eine Staffelung von 25.000 - 250.000 Scans für 99€ - 499€ im Monat ermöglicht auch die Verwendung im kleineren Rahmen. 500 Scans oder weniger sind hingegen gratis[7].

4.4.3 Tag Inspector

Der Tag Inspector[8] ist ein weiterer Anbieter von Analyselösungen im Tracking / Analytics Sektor. Angeboten wird ein Crawler der, unbemerkt von bestehenden Analyse-Tools, die gegebene Webseite durchläuft und nach vordefinierten Ereignissen, Inhalten und Funktionen sucht. Hauptaugenmerk liegt auf vorhandenen, gesetzten Tags und deren Eigenschaften, sowie entsprechend nachgeladenen Tags anderer Anbieter. Tag Inspector beinhaltet keine manuellen Testfälle zum Auslösen vorhandener Tags. Der Monitor Service analysiert lediglich die gesendeten Daten und vergleicht diese gegen vordefinierte Anforderungen. Außerdem registriert Tag Inspector die Zeitdifferenz zwischen Ereignisauslösung und Ladezeit des Tags. Eine fortlaufende Statistik stellt alle Ergebnisse und Ausfälle konsequent dar.

Das Preismodell von Tag Inspector ist analog zu Columbo's aufgebaut. Die Anzahl der Scandurchläufe beträgt jedoch nur bis zu 5.000. Eine genaue Anforderungsanalyse ist in jedem Fall nötig.

4.5 Dateninput

Um ein Testergebnis überprüfen zu können, muss eine Spezifikation vorliegen. Ein Test ist erfolgreich, wenn das Testergebnis exakt der Vorlage entspricht. Dies kann ein binärer Wahrheits-, numerischer- oder Zeichenwert sein. Die Aufgabe des Testentwicklers ist die Implementation und der Abgleich dieser zwei Einheiten. Um dies leisten zu können, muss die Spezifikation dem Test zur Laufzeit bekannt sein. Die

[7] https://columbo.io/preise
[8] https://taginspector.com

Umsetzung dessen kann manuell, also dem händischen Übertragen von Daten, oder automatisch sein.

Tabelle 4.1 zeigt einen Beispielsatz von Schlüssel / Wert Paaren. Die Anzahl und der Inhalt kann beliebig variieren und hängt von der Implementation, dem genutzten Analyticstool, sowie den zu trackenden Datensätzen ab. Im Beispiel hier befindet sich ein Aufrufstyp, ein Name und vier Datenpaare. Der Typ hier entspricht '**p**' für PageView (*Seitenaufruf*) und '**e**' für exit (*Verlassen der Seite*). Der Name ist der Titel der Seite mit dem auslösenden Ereignis. Die zusätzlichen Variablen '**v1**', '**v2**', '**p1**' und '**p8**' tragen beliebige Inhalte.

Tabelle 4.1: Beispielinput von Schlüssel / Wert Paaren

typ	**name**	**v1**	**v2**	**p1**	**p8**
p	homepage	de	de	2016	august
e	impressum	de	de	2016	september

Für einen automatisierten Ablauf ist die Vorgabe eines Datenschemas nötig. Eine .csv Datei könnte beispielsweise den Inhalt und Struktur der Tabelle 4.1 enthalten. Wichtig sind hier das Trennzeichen und eine Spaltendefinition. Gibt man diese Anforderung an den Analysten oder Ersteller des Trackingkonzepts weiter, wäre die Implementierung eines solchen Prozesses denkbar und mit Hinblick auf den zeitlichen Aspekt vorteilhaft, sowie effizient. Die klare Zuweisung von importierten Datensätzen und dazugehörigem Testfall ist hier von fundamentaler Wichtigkeit. Zum Beispiel wäre es möglich, den Dateinamen mit Testfallnamen abzugleichen und bei Übereinstimmung eine Zuweisung durchzuführen. Beim Datenimport wäre der Abgleich mit allen in der Testsuite registrierten Testfällen möglich. Eine Dopplung wird durch explizite Benennung von Klassen und Methoden vermieden.

Für das Speichern von Schlüssel / Wert Paaren eines GET Requests (*?schlüssel=wert*) ist eine HashMap als Datenstruktur ideal. Der Abgleich über den einzigartigen Schlüssel eines jeden Wertes, kann so direkt und in $\mathcal{O}(1)$ stattfinden. Andere Datensätze oder Ergebnisse, wie beispielsweise das erfolgreiche Anklicken eines Downloadbuttons,

oder Abspielen eines Videos muss vom Testentwickler separat und manuell abgeglichen werden.

4.6 Reporting und Logging

Ein Output von Informationen wird im gegebenen Anwendungsfall nur als Ergebnisdokumentation benötigt. Erfolge und Fehlschläge müssen aussagekräftig und korrekt dokumentiert werden. Auf Basis dieser Präsentation ist es dem Analysten möglich, eine fundierte Auswertung zu erstellen. Durch die Ausgabe von Fehlern, können nicht korrekt funktionierende Bereiche erkannt und daraufhin repariert werden.

Das folgende Zitat zeigt zwei beispielhafte Log-Einträge für die Durchführung der '**visitKundenportal( )**' Testmethode:

> *DEBUG: OK : @visitKundenportal : 2016-08-10*
> *DEBUG: FAIL : @visitKundenportal : 2016-08-10*

Gegeben ist das aktuelle Log-Level, das Testergebnis, der Name des Testfalls und das Datum der Testausführung. Diese Ausgabe kann beliebig erweitert werden. Denkbar sind genauere Angaben zur überprüften Funktionalität, Zeilenangaben von erfolgreichen oder fehlerhaften Testabfragen. Des Weiteren ist eine Zeitangabe bis zur genauen Sekunde hilfreich bei oft ausgeführten Durchläufen der Testsuite.

Abbildung 4.6 zeigt den Aufbau einer Testsuite mit enthaltener Testklasse '**TestKundenportal**' und ausführender Testmethode '**visitKundenportal**' bei erfolgreichem Durchlauf. Auf der rechten Seite wird die benötigte Zeit pro Testmethode angegeben. Ein jUnit Testfall enthält verschiedene Behauptungen (Englisch: *Assertions*) die am Ende der Testmethode zutreffen müssen, damit der Testfall als bestanden gilt. Ein primitives Beispiel wäre die Aussage das '$x+3=4 \Rightarrow x=1$'. Da diese Aussage wahr ist, trifft die ursprüngliche Behauptung zu. Abbildung 4.7 zeigt die Darstellung eines fehlgeschlagenen jUnit Tests in der IDE 'IntelliJ Idea 2016.2'. Obiges Zitat spiegelt das Ergebnis durch den Ausdruck '**FAIL**' wieder.

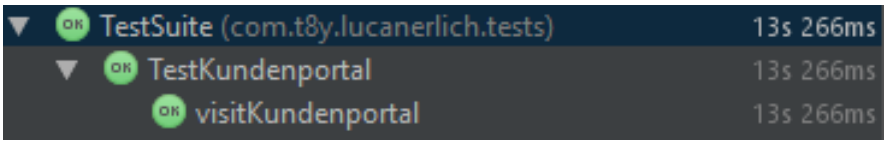

Abbildung 4.6: Erfolgreicher jUnit Testdurchlauf

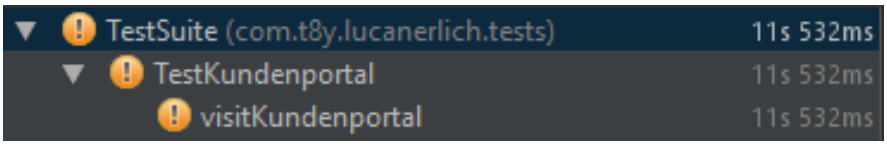

Abbildung 4.7: Fehlgeschlagener jUnit Testdurchlauf

Werden die Ergebnisse über einen längeren Zeitraum gespeichert und aufbereitet, ergibt sich ein Verhältnis der erfolgreichen und fehlgeschlagenen Testdurchläufe. Hieraus lassen sich weitere Informationen zur Robustheit und Verlässlichkeit der gewonnen Datensätze schließen. Wird ein Ergebnis von beispielhaften ∼50% korrekten Testdurchläufen im Livebetrieb der Webseite erreicht, kann ein Analyst des Unternehmens nicht davon ausgehen, dass diese gesammelten Informationen dem tatsächlichen Kundenverhalten entsprechen.

Erfolgreiche und fehlgeschlagene Testdurchläufe müssen festgehalten und erkenntlich wiedergegeben werden. So ist gewährleistet, dass die einzelnen Testdurchläufe ihre Funktion erzielen. Der Analyst kann aus aggregierten Daten vieler Testdurchläufe Rückschlüsse auf die korrekte Funktionalität implementierter Trackingmechanismen ziehen. Durch verlässliche Datensätze sind präzisere, nachfolgende Berechnungen möglich. Bei unvollständigen, oder fehlerhaften Mengen ist nicht gegeben, dass die Abbildung der Realität entspricht und somit eventuell Tatsachen verzerrt. Aus diesem Grund ist die Auswertung der Testergebnisse wichtig und essentieller Bestandteil einer Testsuite.

Ein aussagekräftiges Logging- und Reportingsystem gibt die Ergebnisse in aufbereiteter Form wieder. Die Integration einer Datenbank würde die Akkumulation von Testergebnissen ermöglichen. Somit wäre ein Rückschluss auf historische Daten und die Berechnung einer Erfolgsquote möglich. Des Weiteren wird dadurch die Korrektheit

im Verlauf der Entwicklung festgehalten werden. An Hand der Erfolgsquote könnte ein Analyst präzisere Entscheidungen bezüglich Kundenverhalten oder möglichen Änderungen treffen. Als Datenstruktur würde ein primitiver Tabellenaufbau mit Informationen über den Namen des Testfalls, Ausführungszeitpunkt und Erfolgsstatus reichen.

5 Methodik und Implementierung

Auf Basis des Soll-Konzepts wird ein erster Lösungsansatz beispielhaft implementiert und an Hand eines Fallbeispiels erläutert. Des Weiteren wird die technische Überprüfung der Tracking Requests modelliert und erklärt.

5.1 Page Object Pattern

Bei der Entwicklung von Testfällen für die Elemente einer Webseite ist es wichtig zu beachten, dass die Tests bei kleineren Veränderungen nicht direkt fehlschlagen. Der entsprechende Testfall hat zum Ziel, die Funktionalität zu überprüfen. Ändert sich eine Kleinigkeit am Frontend (zB. Ein Element wird um x Pixel verschoben), wird ein robuster Testfall weiterhin zum korrekten Ergebnis gelangen (vgl. Blackburn und Nauman [2004] Seite 4). Um diese Anforderung zu erreichen, stellt das Page Object Pattern eine Lösung bereit. Durch die Zusammenfassung und Untergliederung von Eigenschaften der Webseite in klassischen Methoden, entsteht eine Abstraktionsebene. Wie Fowler (vgl. Fowler [2013]) beschreibt, versieht ein Page Object eine HTML Seite mit einer entsprechenden Programmierschnittstelle. Dem Entwickler ist es dadurch möglich, Elemente direkt anzusprechen, ohne den HTML Code zu verändern. Werden die Lokalisierungsmöglichkeiten der Elemente nicht verändert, bleibt die Testfunktionalität auch bei Anpassungen der Oberfläche bestehen. Dies entspricht der Definition von Leotta u. a. [2015]:

> *A Page Object Pattern is a class that represents the web page elements as a series of objects and encapsulates the features of the web page in methods.*

Frei ins Deutsche übersetzt:

> *Ein Page Object ist eine Klasse, die Webseiten Elemente als Gruppe von Objekten repräsentiert und dessen Funktionen in Methoden kapselt.*

Kumar (vgl. K [2013]) ergänzt weitere Merkmale, wie die Reduzierung von Code Duplikation oder die Möglichkeit das Benutzerinterface der Webseite im Code nachzubauen. Beispiel 5.1 zeigt eine Methode des Page Objects Login. Diese stellt den Benutzernamen und das Passwort für einen Loginvorgang bereit.

```
1 public void with(String username, String password) {
2         type(username, usernameLocator);
3         type(password, passwordLocator);
4         //Locatorbeispiel: By loginFormLocator = By.id("login");
5         submit(loginFormLocator);
6 }
```

Sourcecode 5.1: Page Object - Login Methode Beispiel Aufbau

In Abschnitt 5.2 werden Lokatoren, sowie deren Unterschiede, genauer betrachtet. Methoden werden üblicherweise so geschrieben, dass sie der natürlichen Nutzung einer Webseite entsprechen. Aus diesem Grund heißt obige Methode 'with'. Wie man dem folgenden Beispiel entnehmen kann, entsteht hierdurch ein einfach zu verstehender Testaufbau und Ablauf. Der Nutzer kann die Funktionalität in natürlicher Sprache auslesen. Methoden dieser Art spiegeln häufige Implementationen von Webseiten wieder. Durch den Aufbau von Page Objects ist es möglich dieses Verhalten auf unterschiedlichen Webseiten wiederzuverwenden.

Als Abstraktionsebene wird ein Base Page Object angelegt. Alle grundlegenden Funktionen werden hier gesammelt und implementiert. Häufig benötigte Methoden, vereint an einer zentralen Stelle, ermöglichen den schnelleren Aufbau von weiterführenden Page Objects, minimieren Code Duplikationen und vereinfachen zukünftige Wartungen bei Veränderungen des Frameworks oder Austausch des WebDrivers[1] (vgl. Haeffner [2013], Seite 36).

[1] Schnittstelle die eine Steuerung des Browsers durch API Aufrufe ermöglicht. Beispiel: http://www.seleniumhq.org/projects/webdriver/

Die Basisklasse kann beispielsweise Methoden zum Öffnen von URLs (Siehe Sourcecode 5.2), Finden von Web Elementen (Siehe Sourcecode 5.7), Klicken oder die Eingabe von Text bereitstellen. Weiterführende Page Objects, die von dieser Basis Klasse erben oder ableiten, verfügen dadurch über einen Grundstock an elementaren Funktionen. Der Fokus liegt in der Abbildung von UI Funktionalitäten - Zugriffslogik und anderweitiger Code bleiben im Hintergrund (vgl. Fowler [2013]).

```
1 public void visit(String url) {
2         if (url.contains("http")) {
3         webDriver.get(url);
4         } else {
5         //String baseUrl = "http://";
6         webDriver.get(baseUrl + url);
7         }
8 }
```

Sourcecode 5.2: Page Object - Aufruf einer URL

Der Methodenaufruf '**isDisplayed(By locator)**' (Siehe Sourcecode 5.3) ermöglicht die Kontrolle der Präsenz des übergebenen Elements.

```
1 public Boolean isDisplayed(By locator) {
2         try {
3                 return find(locator).isDisplayed();
4         } catch (NoSuchElementException exception) {
5                 return false;
6         }
7 }
```

Sourcecode 5.3: Page Object - Präsenzkontrolle eines Elements

Methodenaufrufe dieser Art repräsentieren die möglichen Interaktionsmöglichkeiten des Webseiten Nutzers mit der Webseite. Beispielsweise können Formulare ausgefüllt, Dokumente und Dateien heruntergeladen, Buttons geklickt, Videos abgespielt oder Links gefolgt werden.
Durch diese Kapselung werden die Tests sauber von dem Benutzerinterface (UI) getrennt. Die Logik existiert an einer zentralen Stelle und kann dadurch effizient gewartet werden.

5.1.1 Testausführung

jUnit ermöglicht die Zusammenfassung multipler Testklassen innerhalb einer TestSuite. Durch diesen Vorgang ist es möglich, auf manueller Weise viele Testmethoden aus unterschiedlichen Testklassen in einem Ablauf auszuführen. Die TestSuite Klasse (Siehe Sourcecode 5.4) ist leer. Grundlegend sind nur die beiden Anmerkungen (Englisch: *Annotation*) '**@RunWith**' und '**@Suite.SuiteClasses**' wichtig. Wobei erstere den standard Test-Runner überschreibt und mit dem der annotierten Klassen ersetzt. In diesem Fall wird der Runner '**Suite**' verwendet. Ein Runner verwaltet die Ausführung der Testmethoden der jeweiligen Testklassen. '**Suite**' gruppiert mehrere Testklassen zu einer größeren Einheit.

```
@RunWith(Suite.class)
@Suite.SuiteClasses({
        TestHomepage.class,
        TestShop.class
        [...]
        }
)

//Klasse mit leerem Körper
public class TestSuite {
}
```

Sourcecode 5.4: Testausführung - jUnit TestSuite Runner

Wie der Sourcecode 5.5 zeigt, gestaltet sich der jUnit Methodenaufbau geradlinig und leserlich. Eine Abfrage des HTTP Codes '200' überprüft die erwartete Antwort 'OK' bei korrekten Anmeldedaten.

```
//public class Login extends Base [...]}

@Test
public void succeeded() {
        login.with("usernameX", "passwordY");
        assertTrue(login.httpStatusOfPageIs200());
}
```

Sourcecode 5.5: Testausführung - Test der PO Loginmethode

5.2 Web Element Lokalisierung

Für die Automatisierung eines Webseitentestfalls, ist die exakte Lokalisierung des benötigten Elements essentiell. Der Selenium Webdriver benötigt eine einzigartige und eindeutige Referenz. Diese Referenzen werden im Frontend festgelegt und müssen hier ausgelesen werden. Die beiden Hauptmethoden der Lokalisierung sind die Objektreferenz per ID und der Aufruf über den XPath. Wie Leotta (vgl. Leotta u. a. [2013b]) darlegt, entsteht durch die Wartung der Testfälle der größte Zeitbedarf. Die Lokalisierung der Elemente für den Webdriver stellt hier einen relevanten Faktor dar, denn generell ändert der Entwickler im späteren Verlauf der Umsetzung eher die Kategorisierung einzelner Elemente anstatt einer Reorganisation des Seitenaufbaus, denn ein einfaches Hinzufügen einer zusätzlichen * -Box* kann den XPath bereits verändern. Zu beachten ist, dass die Methode des Zugriffs per ID im Verlauf der Projektwartung ungefähr drei Mal schneller ist (Siehe Leotta u. a. [2013b] Abbildung 3, Tabelle 5.1). Dieser Wert ist programmabhängig, kann jedoch als Richtlinie angesehen werden. Die Tabelle 5.1 von Leotta zeigt den benötigten Mehraufwand bei der Lokalisierung per XPath. Außerdem ist die Anzahl der zu verändernden Codezeilen angegeben.

Tabelle 5.1: Benötigte Zeit für Wartungsarbeiten am Code

	Time		**Code**	
Test Suites	**Minutes**	**Difference**	**LOCs**	**Difference**
ID	43	0 %	9	0 %
ID + LinkText	36	- 16 %	8	- 11 %
XPath	183	+ 326 %	96	+ 967 %
XPath+LinkText	135	+ 214 %	86	+ 856 %

'**LinkText**' ist der für den Nutzer sichtbare Textteil eines Hyperlinks. Im folgenden Beispiel wäre dies der '**Öffne Google**!' Anteil.

Öffne Google!

5.2.1 ID Lokalisierung

Die Referenzierung eines Elementes per ID greift auf den, optional gesetzten, ID-Wert eines HTML Elements zu. Sourcecode 5.6 zeigt einen die Implementierung eines Buttons mit gesetzter ID in HTML.

```
<button id="button1" type="button">Downloads</button>
```

Sourcecode 5.6: ID Lokalisierung - HTML Button

Die zugewiesene ID entspricht "**button1**". Im Kontext eines Testfalls würde die Verwendung wie in Sourcecode 5.8 durchgeführt werden. Sourcecode 5.7 beinhaltet die Methode '**find(By Locator)**' zur Rückgabe des gesuchten Webelements auf Basis des übergebenen Parameter.

```
public WebElement find(By locator) {
        return webDriver.findElement(locator);
}
```

Sourcecode 5.7: ID Lokalisierung - Finden eines Webelements

```
//Selenium Locator
By buttonLocator = By.id("button1");

//Methodendefinierung
public void click(By buttonLocator) {
        find(buttonLocator).click();
}
```

Sourcecode 5.8: ID Lokalisierung - Testcase Button

Verwendet ein Entwickler konsequent ID's und nutzt diese nicht doppelt, gestaltet sich der Zugriff gradlinig und simpel. Da dieses Verfahren jedoch nicht garantiert ist, muss der Tester sicherstellen, dass die ID's noch den gewünschten Elementen entsprechen. Ist diesem so, gestaltet sich die benötigte Wartung verhältnismäßig kurz,

da lediglich der ID Parameter angepasst werden muss, um erneut einen korrekten Testfall zu erhalten.

5.2.2 XPath Lokalisierung

Existiert keine ID für das jeweilige HTML Element, kann im Zuge der Elementlokalisierung auf den XPath zurückgegriffen werden. An dieser Stelle soll die Syntax von XPath nicht weiter erläutert werden. Für dieses Beispiel reicht die Erklärung, dass folgende XPath Angabe die HTML Struktur von oben nach unten traversiert. Jedes Element ist dem vorherigen untergeordnet. Durch eine Nummerierung wird das x-te Elemente derselben Ebene gewählt.

/html/body/div[3]/a

Im obigen Beispiel wird das **<a>** Element der dritten ** ** Box innerhalb des HTML Rumpfes referenziert. Im Falle einer manuellen Ermittlung lässt sich leicht über die Entwicklerkonsole innerhalb verschiedener Browser ermitteln. In Chrome muss hierfür mit '**F12**' die Entwicklerkonsole geöffnet und mit '**Strg-Shift-C**' das entsprechende Element markiert werden. Per Maus-Rechtsklick -> '**Copy**' -> '**Copy-XPath**' lässt sich der XPath in der Zwischenablage speichern.

5.3 Fallbeispiel

Im folgenden Abschnitt wird der Aufbau eines Testfalls inklusive Überprüfung des entsprechenden Trackingrequests exemplarisch dargestellt. Als Beispiel Webseite mit Request wurde `http://volkswagen-carnet.com/de/de/start.html` ausgewählt (zuletzt besucht am 31.05.2016). Durch einen Klick auf den Button 'Kundenportal' innerhalb der Navigationsleiste wird ein Trackingrequest ausgelöst. Die erforderten Werte/Schlüssel Paare der im Request enthaltenen Parameter, liegen zum Zeitpunkt der Testfallerstellung als Kundenspezifikation vor.

Die Herangehensweise beinhaltet den Aufbau der Basis sowie der Nachbildung der Zielseite als Page Objects. Durch Verwendung des Selenium Webdrivers wird der Trackingrequest ausgelöst. Lokal ist der Proxy 'Browsermob' zwischengeschaltet und zeichnet den kompletten HTTP Netzwerkverkehr mit. In diesem HTTP Archiv ist zwangsläufig der gesuchte Trackingrequest enthalten. Ziel ist es, diesen JSON Eintrag im Laufe des Testdurchlaufs zu erkennen und mit der gegebenen Spezifikation zu vergleichen. Schlussendlich soll das Ergebnis dem Tester aussagekräftig präsentiert werden.

Der Aufbau des initialen Page Objects Konstrukts orientiert sich an der detailliert beschriebenen Themeneinführung von D. Haeffner (vgl. Haeffner [2013]). Eine Basisklasse (hier: 'base') beinhaltet alle Grundfunktionalitäten eines Page Objects. Hiermit wird beispielsweise das Klicken von Elementen, das Finden eben dieser, Dateninput, sowie entsprechende Warte-Funktionalität ermöglicht. Durch eine Erweiterung und Anpassung dieser Basisklasse wurde die Integration des BrowserMob Proxy ermöglicht. Abbildung 5.1 zeigt ein Klassendiagramm der Basis mit entsprechender Erweiterung passend zur obigen URL.

Die Klassen *CarNetBase* und *Kundenportal* sind schlicht programmiert und repräsentieren lediglich den optischen Aufbau der Webseite. Sourcecode 5.9 zeigt die Kernmethoden. Durch das Erben der 'Base' Klasse existiert der Zugriff auf alle im Vorfeld implementierten Kernfunktionen. Bei wachsender Größe der Testsuite mit steigender Anzahl von Testfällen, ermöglicht das Page Object Pattern ein klar strukturiertes Projekt und eine gute Orientierungsmöglichkeit zu jeder Zeit. Alle Methoden lassen sich beim Lesen deutlichen Funktionalitäten zuweisen und ermöglichen so einen einfach verständlichen Testfallaufbau.

Der Elementenzugriff in Sourcecode 5.10 wurde per XPath Lokalisierung (Siehe Zeile 9) durchgeführt, da auf der Webseite keine ID für den Button vergeben wurde.

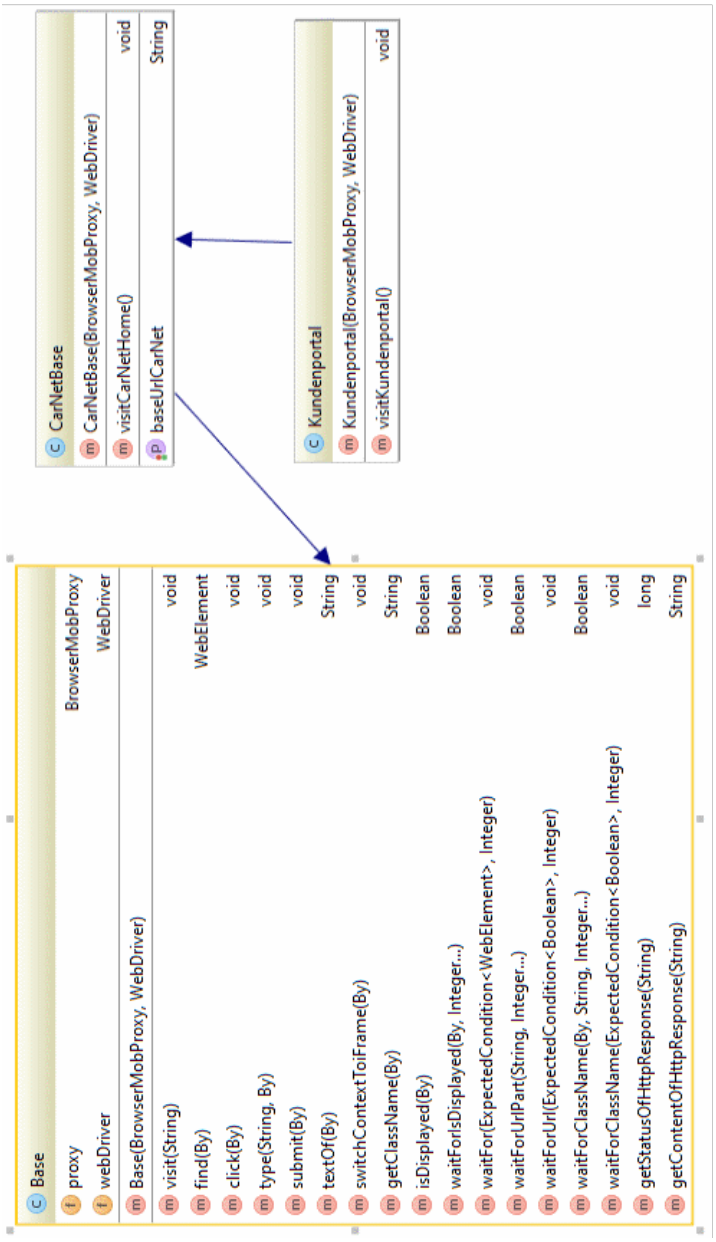

Abbildung 5.1: CarNet Kundenportal - UML Basis Page Objects

```
1  //Class BaseCarNet
2  String baseUrlCarNet = "http://volkswagen-carnet.com/de/de/start.html";
3
4  //Konstruktor
5  public CarNetBase(BrowserMobProxy proxy, WebDriver driver) {
6          super(proxy, driver);
7  }
8
9  public void visitCarNetHome(){
10         visit(baseUrlCarNet);
11 }
```

Sourcecode 5.9: PageObject - BaseCarNet

```
1  //Class Kundenportal
2  //Konstruktor
3  public Kundenportal(BrowserMobProxy proxy, WebDriver driver) {
4          super(proxy, driver);
5  }
6
7  public void visitKundenportal(){
8          visitCarNetHome();
9          click(By.xpath("/html/body/header/div[2]/ul[2]/li[4]/a"));
10 }
```

Sourcecode 5.10: PageObject - Kundenportal

5.3.1 HAR Analyse

Jede Anfrage an einen Server erzeugt Netzwerkverkehr. Dieser wird von dem Proxy BrowserMob (Siehe Abschnitt 2.4) mitgeschnitten und als LogDatei im .har Format gespeichert. Um zur Laufzeit des Testfalls (Richtige Anfrage, an der richtigen Stelle ausgelöst) den gestellten Trackingrequest zu überprüfen, müssen die gesendeten URL Parameter mit einer im voraus definierten Spezifikation verglichen werden. Dazu wird im HTTP Archiv die entsprechende Anfrage gesucht und extrahiert. Anschließend werden die URL Parameter getrennt und einzeln in einer HashMap gespeichert. Die gesammelten Schlüssel-Wert Paare werden im Anschluss mit, vom Tester vorgegebenen, Paaren verglichen.

Codebeispiel 5.11 zeigt die Initialisierung und Basiskonfiguration des BrowserMob Proxy in Java. Weiterführende Erklärungen finden sich als Kommentar im Code.

```
1  protected BrowserMobProxy proxy = new BrowserMobProxyServer();
2  private String httpArchive; //Name des Http Archivs
3  private final int PORT = 0;
4
5  //Übergibt Anforderungen an Selenium
6  DesiredCapabilities capabilities = new DesiredCapabilities();
7
8  //Welche Daten sollen geloggt werden
9  HashSet<CaptureType> enabledInfo = new HashSet<CaptureType>();
10 enabledInfo.add(CaptureType.RESPONSE_HEADERS);
11 [...]
12 proxy.enableHarCaptureTypes(enabledInfo);
13
14 //Welche Daten sollen nicht geloggt werden
15 HashSet<CaptureType> disabledInfo = new HashSet<CaptureType>();
16 disabledInfo.add(CaptureType.RESPONSE_COOKIES);
17 [...]
18 proxy.disableHarCaptureTypes(disabledInfo);
19
20 //Starte den Proxy und übergebe den Namen des HTTP Archivs
21 proxy.start(PORT);
22 proxy.newHar(httpArchive);
23
24 //Füge den Proxy Selenium hinzu, lege Anforderungen fest
25 Proxy seleniumProxy = ClientUtil.createSeleniumProxy(proxy);
26 capabilities.setCapability(CapabilityType.PROXY, seleniumProxy);
```

Sourcecode 5.11: HAR Analyse - Basiskonfiguration BrowserMob Proxy

Im Codebeispiel 5.12 wird das HAR Archiv gegriffen und jeder Eintrag untersucht. Zeile **2** speichert alle separaten Netzwerkverkehrseinträge in einer Liste. Zeile **4** ist ein regulärer Ausdruck. Dieser trennt alle Schlüssel-Wert Paare der URL in jeweils drei Gruppen. Dies ist nötig, um die einzelnen Paare getrennt in einer Map zu speichern. Die erste Teilgruppe ist der Teilstring, die zweite ist der Schlüssel und als dritte Gruppe wird der Wert abgelegt. Die For-Schleife in Zeile **12** bis **25** führt den Abgleich der Schlüssel-Wert Paare mit der Spezifikation durch. Hierbei wird jeder Eintrag im HTTP Archiv abgeglichen. Ist mindestens eine korrekte Anfrage enthalten, kann der Tester davon ausgehen, dass die Anfrage korrekt ausgelöst worden ist.

Zeile **16** splittet die URL in Adresse und Parameter Teilstrings. Die For-Schleife in **18** bis Zeile **21** weist die Schlüssel-Wert Paare der Vergleichs-Map zu.

Zeile **23** vergleicht die gesammelten Paare mit der Spezifikation. Wird ein richtiger Eintrag gefunden, unterbricht Zeile **24** den Durchlauf.

```
1  Har httpArchive = proxy.getHar();
2  List<HarEntry> entries = httpArchive.getLog().getEntries();
3
4  Pattern p = Pattern.compile("([^=]+)\\=([^&#]+)");
5  boolean trackingReqFound = false;
6
7  Map<String, String> expectedValues = new HashMap<String, String>();
8  expectedValues.put("key", "value");
9  expectedValues.put("key2", "value2");
10 [...]
11
12  for (HarEntry entry : entries) {
13    String url = entry.getRequest().getUrl();
14         if (url.contains("metric")) {
15         Map<String, String> parameter = new HashMap();
16         String[] parts = url.split("\\?")[1].split("&");
17
18         for (String part : parts) {
19      Matcher m = p.matcher(part);
20      if (m.matches()) parameter.put(m.group(1), m.group(2));
21         }
22
23         trackingReqFound = compareMap(parameter, expectedValues);
24         if (trackingReqFound) break;
25  }
26 }
```

Sourcecode 5.12: HAR Analyse - Key Value Pairs

Die Vergleichsmethode (Code 5.13) stellt jeden Parameter Wert der Inputmap dem entsprechenden Wert aus der Spezifikationsmap gegenüber. Wird ein nicht übereinstimmendes Schlüssel-Wert Paar gefunden (**4**), kann die Anfrage nicht mehr korrekt sein. Somit wird der Vergleich abgebrochen und die Rückgabe '**False**' in Zeile **5** signalisiert einen fehlgeschlagenen Vergleich.

5.3.2 Klassenaufbau

Testklassen werden analog zu den Page Object Pattern Klassen implementiert. Dies bedeutet, dass es für jedes Page Object eine direkt zu überführende Testklasse geben

```
1 //Base.class
2 boolean compareMap(Map input, Map spec) {
3  for (Map.Entry entry : spec.entrySet()) {
4   if (!input.get(entry.getKey()).equals(entry.getValue())) {
5    return false;
6   }
7  }
8  return true;
9 }
```

Sourcecode 5.13: HAR Analyse - Compare String Map

sollte. Durch diese Differenzierung ist die Erstellung von schlichten Code gewährleistet. Des Weiteren wird zukünftige Wartung unkomplizierter. Abschnitt 5.2 beschreibt die gängigen Mechanismen der Elementlokalisierung. Dies ist eine der häufigsten Stellen mit nachträglichem Anpassungsbedarf, da sich der Lokalisierungspfad oder ID Wert leicht verändern kann. Die Abbildung 5.2 stellt eine vereinfachten Aufbau eines 'Page Object Pattern' Klassenkonstrukts in Kombination mit einer Testklasse dar. Die Klasse 'Base' stellt jeweils für das Page Object, sowie für die Testklassen grundlegende Funktionalität und Variablen zur Verfügung. Zum Beispiel wird der Webdriver initialisiert und der Proxy konfiguriert. Die Page Object Basis biete weiterhin Methoden zur fundamentalen Navigation einer Webseite. Zum Beispiel ist es so allen erbenden Klassen möglich Elemente anzuklicken, Test zu übergeben, oder auf dynamischen Inhalt zu warten. Die Test Basis stellt sicher, dass vor jeden Testfalldurchlauf eine immer gleichbleibende Umgebung geschaffen ist.

5.3.3 Kommunikationsstruktur

Die Abbildung 5.3 stellt den Projektverlauf der Beziehung '**Externer Kunde -> Werbeagentur / Designer -> Technischer Dienstleister / Entwickler**' dar. Nach ursprünglicher Auftragserteilung eines Dritten beginnt die Werbeagentur oder ein Designer mit der Kozeptionierung und Entwicklung des Layouts. Nach erfolgreicher Abnahme durch den Kunden wird dieses Konzept wiederum als Unterauftrag an den technischen Dienstleister oder Entwickler weitergegeben. Dieser plant und setzt die

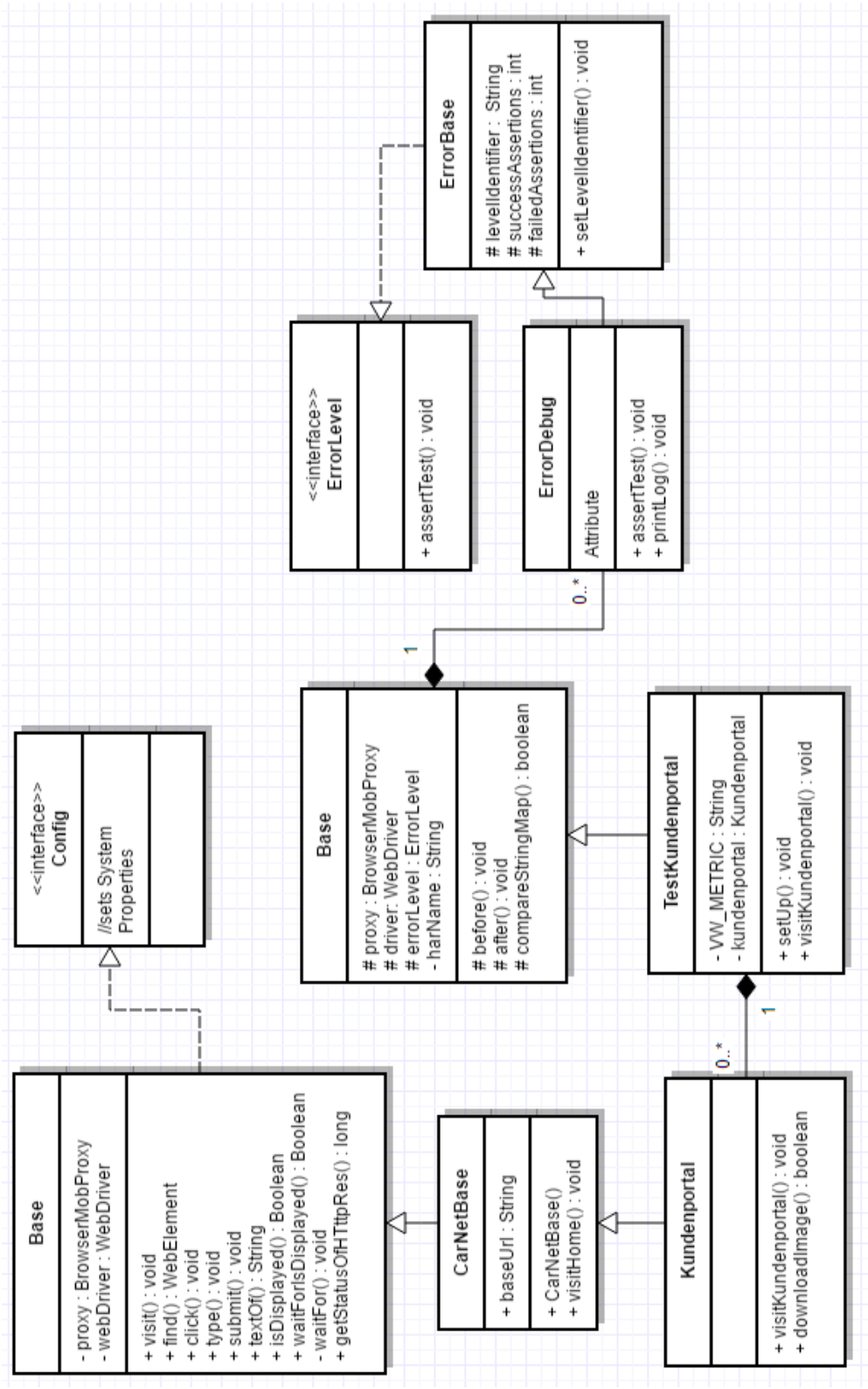

Abbildung 5.2: Vereinfachtes Klassendiagramm

Entwicklung auf Basis des Konzeptes und weiterer technischer Anforderungen um. Dabei werden für jegliche Seiten und Funktionen der Webseite 'Page Objects' und Methoden angelegt. Dem Designer oder Analysten werden entsprechende Inhalte weitergeleitet und erklärt.

Auf Basis dieser können Testfälle entwickelt und durchgeführt werden. Dies geschieht beim Kunden des technischen Dienstleisters, aber auch rudimentär beim Entwickler. Genauere Abläufe müssen je Projekt und Komplexitätsgrad individuell abgesprochen und verhandelt werden. Durch die Abgabe der Testfallerstellung zum Entstehungspunkt der Trackingspezifikation, können wesentliche präzisere Testfälle aufgebaut und auf anfallende Fehler reagiert werden.

Sind die regulären Inhalte fertiggestellt und das Trackingkonzept wurde in Absprache mit dem Kunden komplett umgesetzt, geht das Projekt in eine finale Abgabe. Entstehen hieraus keine weiteren 'Change Requests' wird die Abnahme eingeleitet. Wurde kein fortlaufender Support vertraglich geregelt, trennen sich die Wege der Parteien vorerst.

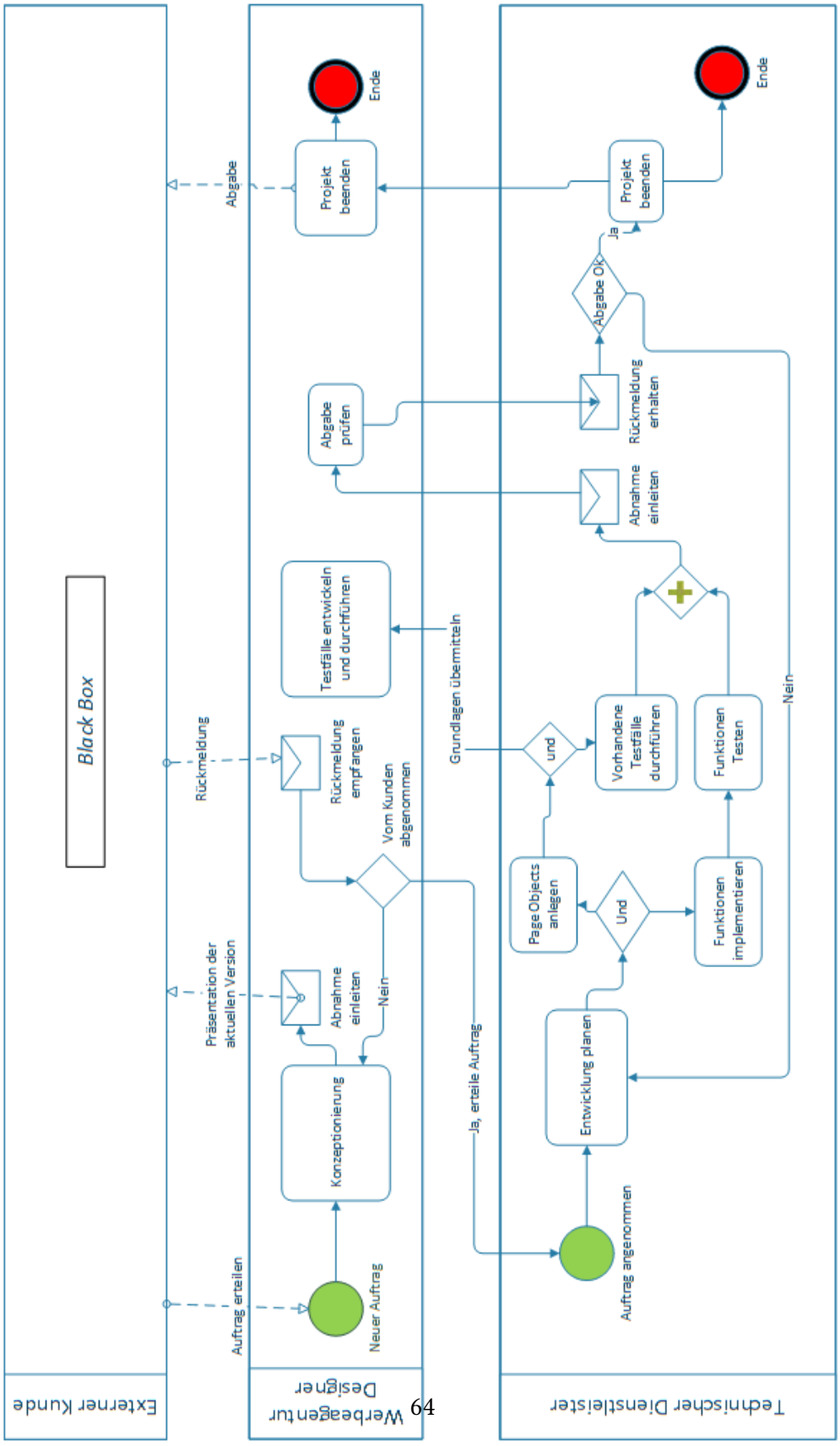

Abbildung 5.3: BPMN Diagramm des Projektverlaufs

6 Ergebnisse und Fazit

Im abschließenden Kapitel werden die einleitenden Ziele erneut aufgegriffen und mit den erarbeiteten Lösungsansätzen unterlegt. Gefolgt von einer Zusammenfassung wird ein kurzer Ausblick über mögliche Erweiterungen gegeben. Eine finale Kritik fügt zusätzliche Gedanken hinzu.

6.1 Beantwortung der Zielsetzung

Im folgenden werden die Ziele aus Abschnitt 1.4 erneut aufgenommen und im Kontext der Arbeit erläutert sowie mit Inhalt versehen. Außerdem stellt dies eine Zusammenfassung der einzelnen Ergebnisse dar.

Z1 - Verbesserung der Code Qualität durch konstantes Testen:

Die Ausgliederung des Testprozesses aus dem linearen Ablauf der Webseitenentwicklung ermöglicht einen konstanten Kontrollzyklus überprüfter Funktionen. Die Entstehung von Fehlern im Code, ist nicht zu vermeiden. Die Ermittlung von Fehlern und deren Korrektur ist sogar essentieller Bestandteil der Entwicklung und Verbesserung eines Produktes. Man kann nicht die Annahme treffen, dass das Produkt aus dem Stand heraus perfekt ist.

Z2 - Früherkennung möglicher Fehler und Fehlerquellen:

Einhergehend mit Z1 ist die Früherkennung von Fehlern respektive deren Auslöser. Durch einen aus dem Testalltag ausgekoppelten Prozess, ist eine frühere Erkennung und Bearbeitung von Fehlern möglich. Die Testsuite inklusive Testfälle kann regelmäßig ausgeführt werden. Dies ist beispielsweise möglich durch Jenkins. In einem aussagekräftigen Report steht das Ergebnis jedes Testfalls. Durch die Aggregation über einen Zeitraum lassen sich außerdem Rückschlüsse über besonders häufig fehlschlagende Testfälle gewinnen. Diesen sollte besondere Aufmerksamkeit gegeben werden. Eventuell ist eine Veränderung der Webseitenimplementierung oder eine Überprüfung des Testfalls selber nötig. Bei Fortbestand hoher Fehlermengen muss die Trackimplementation erneut besprochen und präziser geplant werden.

Z3 - Reduzierung des QS Aufwandes:

Auf Grund von effizienteren Kommunikationswegen und neu aufgeteilten Aufgabenstrukturen werden im Endeffekt weniger Probleme auftreten. Die ursprünglichen Aufgaben des QS Teams, das manuelle Testen von Tracking Implementationen bei Auftreten eines Fehlers, werden aufgeteilt und an Personen dichter am Ursprung abgegeben.

Z4 - Straffung und Vereinheitlichung aktueller Arbeitsschritte:

Zum Stand vor dieser Arbeit wurden auftretende Probleme beim Kunden über mehrere Stationen zum Entwickler getragen. Dieser musste per Hand das Problem nachstellen und die Ursache suchen. Durch erstmalige Generierung von Testfällen direkt beim Kunden, Analyst oder Tag-Manager-Verwender hat der eigentliche Webseitenentwickler Zeit für weitere Aufgaben und komplexere Implementationen. Wenn Kunde und Inhaber der Trackingspezifikation direkt die Testfälle erstellen, werden die Fehler bei übergebender Kommunikation gestrichen. Führt er Änderungen an den Anforderungen durch, können diese sofort auch im entsprechenden Testfall inkludiert werden. Der Entwickler verändert nur noch die eigentliche Implementation.

Z5 - Halb-Automatisierung manueller Tätigkeiten:

Zum Stand der Arbeit muss das Tracking per Hand ausgelöst und überprüft werden. Dieser Prozess ist zeitaufwändig, mühsam und fehlerträchtig. Die Einführung von Testfällen und entsprechender Browserautomatisierung durch Selenium ersetzt einen Großteil dieser Arbeit. Eine zusätzliche Integration in Jenkins als ausführenden Server, ermöglicht eine regelmäßige, autonome Ausführung der Testsuite. Somit wäre eine konstante Überprüfung sichergestellt. Eventuelle Fehler würden zeitnah entdeckt werden. An dieser Stelle entsteht die Halb-Automatisierung. Zuvor entwickelte Testfälle werden wiederholt, ohne menschlichen Eingriff, ausgeführt.

6.2 Fazit und Zusammenfassung

In dieser Arbeit wurden aktuelle Schwierigkeiten und mögliche Verbesserung des Testprozesses von Tracking Requests im Unternehmen t8y.com herausgearbeitet. Durch die häufige und oft doppelte Kommunikation zwischen dem Kunden und Entwicklern bezüglich Trackinganforderungen, entsteht eine Fehlerquelle die vermieden werden kann. Im Projektverlauf ist die Implementation des Trackings ein Punkt der häufig neu ausgerichtet oder erweitert wird. Dies geschieht auf Grund von stetigen Veränderungen des Inhalts und der Funktionalität einer Webseite. Arbeitet eine Partei an der Umsetzung von Anforderungen während die Andere etwaige Planungen übernimmt, würden Fehler strukturierter bearbeitet werden können. Die Ist-Situation beschreibt aber den Fall von multiplen Aktoren im Bereich der Trackingimplementation. Insbesondere durch Tag-Manager-Programme wird Code von externen Bearbeitern integriert. Durch diesen Vorgang entstehen unbemerkte Fehler und Datenqualitätsprobleme auf Grund von nicht korrekt funktionierender Trackingmechanismen.

Die Auslagerung der Testfallerstellung direkt zum Entstehungspunkt der Spezifikation und den häufigsten Anwendern der Tag-Manager, würde diese Problematik minimieren. Durch den Aufbau von 'Page Objects' ist es auch Personen mit grundlegenden Programmierkenntnissen möglich Testfälle zu erstellen. Im Falle von t8y.com wäre dies der initiale Kunde, Auftraggeber und Ideenverfasser. Ergeben sich neue Trackinganforderungen kann dieser sie dem Entwickler mitteilen und im gleichen Zug einen Testfall

aufbauen. Durch diesen Prozess ist eine stetige Kontrolle oft veränderter Funktionen gesichert. Problemfälle werden in kurzen Zeitrahmen nach Auftreten bemerkt und können somit zeitnah repariert werden.

6.3 Mögliche Erweiterungen

Dem in dieser Arbeit beschriebenen, halbautomatischen Ansatz zur Überprüfung von Tracking Requests mangelt es an funktionaler Automatisierung der Testfallerstellung. Das Tool APOGen[1] ermöglicht die automatische Generierung von Page Objects auf Basis eines Webcrawlers (vgl. Stocco u. a. [2015]). In folgender Mail vom 27.06.2016 teilte Andrea Stocco die jüngsten, vom ihm getesteten Versionen mit.

> Hi Luca! At the time of the experiments, I tested APOGEN with these correct settings: - Firefox 31.0 - Crawljax 3.7 - Selenium 2.45 (but FirefoxDriver 2.44)
>
> Yeah, I'm planning to move APOGEN to GitHub and Maven, but I've no time at now.
>
> Good luck for your thesis, let me know if you need anything else.
>
> Best,

Im Verlauf der Arbeit ist es nicht möglich gewesen, APOGen mit exakt dieser Konfiguration erfolgreich zu starten. Auch neuere Versionen der integrierten Tools verhalfen keine Abhilfe. Eine Portierung des Tools auf die neuste FireFox WebDriver Version, sowie eine Projektbasis auf Git und Maven als Abhängigkeitsverwaltung wäre denkbar und praktisch, jedoch aktuell zu zeitaufwändig.

[1] https://app.assembla.com/spaces/sepl-apo/subversion/source/HEAD/trunk

Screenshot bei Test-Fehlschlag

Selenium ermöglicht die automatische Erstellung eines Screenshots bei Testfehlschlag. Ein Einbau dieser Funktionalität würde das finden der Ursprungsortes erleichtern. Beim Aufbau aussagekräftiger Page Objects und klar benannter Testmethoden ist dies jedoch zweitrangig, da der Klassen und Methodenname generell Aufschluss über den Testort gibt.

6.4 Kritik

Der Thesis Titel lässt Rückschlüsse auf eine komplette Automatisierung zu, dies ist jedoch im Umfang dieser Bachelorarbeit nicht möglich. Für eine volle Automatisierung ist deutlich mehr Forschungsarbeit, sowie die Entwicklung neuer Algorithmen zum Parsen von Webseiten inklusive Trackinganalyse, Erkennen von Funktionsstrukturen und Erstellen von Testfällen nötig.

Durch die veränderte Arbeitsaufteilung und erweiterten Kommunikationswegen zum Kunden, beziehungsweise zu Analysten, wird vormals selbst erledigte Arbeit, zu einem gewissen Teil aus der Hand gegeben. Somit entsteht ein Verlust von Kontrolle und Wissen. Gleichzeitig sinkt das mögliche Aufgabenpensum und die Umsatzsumme wird somit reduziert.

Literaturverzeichnis

[Ammann und Offutt 2008] Ammann, Paul ; Offutt, Jeff: *Introduction to software testing*. Cambridge University Press, 2008. – 321 S

[Blackburn und Nauman 2004] Blackburn, Mark ; Nauman, Aaro: *Strategies for Web and GUI Testing*. 2004. – URL http://www.knowledgebytes.net/downloads/Strategies{_}Web{_}and{_}GUI{_}testing{_}spc-2004014-D-P.pdf

[Buth u. a. 2015] Buth, Bettina ; Lehrmann, Thomas ; Zhen Ru, Dai: HAW SEA2 - Testen Allgemein. v11 (2015), S. 41

[Fowler 2013] Fowler, Martin: *PageObject*. 2013. – URL http://martinfowler.com/bliki/PageObject.html. – Zugriffsdatum: 2016-03-26

[Haeffner 2013] Haeffner, Dave: *The Selenium Guidebook*. Haeffner, Dave, 2013. – URL https://seleniumguidebook.com/

[Hassler 2012] Hassler, Marco: *Web Analytics Metriken auswerten, Besucherverhalten verstehen, Website optimieren*. MITP-Verlags GmbH & Co. KG, 2012. – 602 S. – ISBN 978-3-8266-9122-5

[Heinrich 2007] Heinrich, Gert: *Allgemeine Systemanalyse*. 1. München : Oldenbourg Wissenschaftsverlag GmbH, 2007. – 139 S. – ISBN 978-3-486-58-365-6

[ISTQB und GTB 2016] ISTQB ; GTB: *ISTQB®/GTB Standardglossar der Testbegriffe*. 2016. – URL http://www.german-testing-board.info/service/information/glossar.html. – Zugriffsdatum: 2016-07-27

[K 2013] K, Manoj K.: *Page Object Pattern*. 2013. – URL http://www.assertselenium.com/automation-design-practices/page-object-pattern/. – Zugriffsdatum: 2016-03-26

[Kissmetrics 2016] Kissmetrics: *How These 7 Companies Increased Revenue by an Average of 425%*. 2016. – URL https://blog.kissmetrics.com/increased-revenue-by-425. – Zugriffsdatum: 2016-07-27

[Leotta u. a. 2013a] Leotta, Maurizio ; Clerissi, Diego ; Ricca, Filippo ; Spadaro, Cristiano: Comparing the maintainability of selenium WebDriver test suites employing different locators: a case study. In: *... Contributions to testing ...* (2013), Nr. Jamaica, S. 53–58. – URL http://dl.acm.org/citation.cfm?id=2489284. ISBN 9781450321617

[Leotta u. a. 2013b] Leotta, Maurizio ; Clerissi, Diego ; Ricca, Filippo ; Spadaro, Cristiano: Repairing selenium test cases: An industrial case study about web page element localization. In: *Proceedings - IEEE 6th International Conference on Software Testing, Verification and Validation, ICST 2013*, IEEE, 2013, S. 487–488. – URL http://ieeexplore.ieee.org/xpls/abs{_}all.jsp?arnumber=6569767. – ISBN 978-0-7695-4968-2

[Leotta u. a. 2013c] Leotta, Maurizio ; Clerissi, Diego ; Ricca, Filippo ; Tonella, Paolo: Capture-replay vs. programmable web testing: An empirical assessment during test case evolution. In: *Proceedings - Working Conference on Reverse Engineering, WCRE*, URL http://www.computer.org/csdl/proceedings/wcre/2013/9999/00/06671302-abs.html, 2013, S. 272–281. – ISBN 9781479929313

[Leotta u. a. 2015] Leotta, Maurizio ; Stocco, Andrea ; Ricca, Filippo ; Tonella, Paolo: Automated Generation of Visual Web Tests from DOM-based Web Tests. In: *Proceedings of the 30th Annual ACM Symposium on Applied Computing*. New York,

NY, USA : ACM, 2015 (SAC '15), S. 775–782. – URL http://doi.acm.org/10.1145/2695664.2695847. – ISBN 978-1-4503-3196-8

[Microsoft 2015] Microsoft: *Attention spans*. 2015. – URL https://advertising.microsoft.com/en/WWDocs/User/display/cl/researchreport/31966/en/microsoft-attention-spans-research-report.pdf

[Odvarko 2009] Odvarko, Jan: *HAR 1.2 Spec*. 2009. – URL http://www.softwareishard.com/blog/har-12-spec/. – Zugriffsdatum: 2016-03-30

[Odvarko u. a. 2012] Odvarko, Jan ; Arvind, Jain ; Davies, Andy: *HTTP Archive (HAR) format*. 2012. – URL https://w3c.github.io/web-performance/specs/HAR/Overview.html. – Zugriffsdatum: 2016-03-30

[Ricca und Tonella 2001] Ricca, Filippo ; Tonella, Paolo: Analysis and Testing of Web Applications. In: *Proceedings of the 23rd International Conference on Software Engineering*. Washington, DC, USA : IEEE Computer Society, 2001 (ICSE '01), S. 25–34. – URL http://dl.acm.org/citation.cfm?id=381473.381476. – ISBN 0-7695-1050-7

[Schneider u. a.] Schneider, Markus ; Enzmann, Matthias ; Stopczynski, Martin: *Web Tracking Report 2014*. Darmstadt, Germany : SIT Technical Reports. – 118 S

[Sikowitz u. a. 2016] Sikowitz, Sarah ; Moorehead, Michelle ; Trafton, Ryan ; Berman, Anna ; Hartig, Kara: *The Customer Life Cycle: A Blueprint For Customer-Obsessed Enterprises*. 2016. – URL https://www.forrester.com/report/Embed+The+Customer+Life+Cycle+Across+Marketing/-/E-RES89261. – Zugriffsdatum: 2016-08-13

[Skumar 2016] Skumar: *What are Software Testing Levels?* 2016. – URL http://istqbexamcertification.com/

what-are-software-testing-levels/. – Zugriffsdatum: 2016-04-18

[Sommerville 2010] Sommerville, Ian: *Software Engineering*. 9th. USA : Addison-Wesley Publishing Company, 2010. – ISBN 0137035152, 9780137035151

[Stocco u. a. 2015] Stocco, Andrea ; Leotta, Maurizio ; Ricca, Filippo ; Tonella, Paolo: Why Creating Web Page Objects Manually if It Can Be Done Automatically? In: *Proceedings of the 10th International Workshop on Automation of Software Test*. Piscataway, NJ, USA : IEEE Press, 2015 (AST '15), S. 70–74. – URL http://dl.acm.org/citation.cfm?id=2819261.2819283

[Waisberg und Kaushik 2009] Waisberg, Daniel ; Kaushik, Avinash: Web Analytics 2.0: empowering customer centricity. In: *The original Search Engine Marketing Journal* 2 (2009), Nr. 1, S. 5–11

[Xun 2015] Xun, Jiyao: Return on web site visit duration: Applying web analytics data. In: *Journal of Direct, Data and Digital Marketing Practice* 17 (2015), Nr. 1, S. 54–70. – URL http://dx.doi.org/10.1057/dddmp.2015.33. – ISSN 1746-0174

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 22. August 2016 Luca Steffen Nerlich