Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

Tim Christopher Aldejohann

Automatische Erkennung und Prüfung von Webtracking-Systemen

Fakultät Technik und Informatik
Department Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Tim Christopher Aldejohann

Automatische Erkennung und Prüfung von Webtracking-Systemen

Abschlussarbeit eingereicht im Rahmen der Bachelorprüfung

im Studiengang 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

Abgegeben am 30.08.2016

Tim Christopher Aldejohann

Thema der Arbeit

Automatische Erkennung und Prüfung von Webtracking-Systemen

Stichworte

Webtracking, Webanalyse, Digital Analytics, Testen, Automatisierung, Webcrawler

Kurzzusammenfassung

Das Testen der Nutzerdatensammlung über die Verwendung von Webseiten ist für Unternehmen und deren Nutzerdatenanalysen von hoher Wichtigkeit. Das Testen dieser Datensammler ist insbesondere bei großen Webseiten zeitaufwändig und manuell nur schwer durchführbar. Im Rahmen dieser Bachelorarbeit werden benötigte Funktionen hinsichtlich eines Werkzeuges zur automatisierten Erkennung und Prüfung von Webtracking-Systemen untersucht. Nach der Identifikation verschiedener Szenarien werden diverse Anforderungen daraus abgeleitet. Auf Basis dieser Anforderungen werden danach mögliche Komponenten entworfen. Abschließend wird die Realisierbarkeit des Werkzeuges anhand der Implementierung der Grundfunktionalitäten in Form eines Prototyps nachgewiesen.

Tim Christopher Aldejohann

Title of the paper

Automatic detection and testing of web tracking systems.

Keywords

Web tracking, Web-Analytics, Digital Analytics, Testing, Automatization, WebCrawler

Abstract

The testing of user data collection in terms of website usage is for companies and their user data analytics of high importance. The testing of these data collectors is time-consuming and difficult to perform. As part of this bachelor thesis, required functions regarding a tool for automatic detection and testing of web tracking systems are examined. After the identification of different scenarios, various requirements are derived. Based on these requirements possible components are designed. Finally, the feasibility of the tool implementing it with basic functionality as a prototype is proven.

Inhaltsverzeichnis

1 Einleitung

Das Webtracking bildet die Grundlage der Web-Analytics. Web Analytics liefern Betreibern von Websites eine Vielzahl von Möglichkeiten zur Auswertung von Nutzer-/Nutzungsdaten und eine datenbasierte Optimierung ihrer Onlinepräsenzen. Das folgende Kapitel dient der Einführung in die Abschlussarbeit, es befasst sich mit der Zielsetzung, der Motivation und der Relevanz des Testens von Webtracking für die korrekte Auswertung von Analysedaten. Im Anschluss daran schildert der zweite Teil dieses Kapitels die konkreten Ziele. Diese Arbeit wird im Auftrag der Web Analyse Agentur Peaks & Pies verfasst.

1.1 Relevanz des Webtracking

Web-Analytics-Dienste beschäftigen sich mit der Nutzerdatensammlung im Internet. Diese Sammlung an Daten beschränkt sich nicht nur auf persönliche Daten wie Alter oder Postleitzahl, sondern auch auf das Verhalten der **Besucher** einer Webseite. Mit Hilfe der Analysen dieser Informationen können die **Betreiber**, also die Anwender des Webtracking und Besitzer der Webseite, bessere Einsichten in das Nutzungsverhalten ihre Besucher erhalten. Durch die Nutzung von Web-Analytics Tools wird ersichtlich woher der **Traffic**, der Datenverkehr im Internet, kommt und welche Unterbereiche einer Website den meisten Traffic anziehen.

Beispielszenario:

Ausgangssituation: Der Betreiber eines Online Shops möchte wissen welche, seiner Produkte am meisten betrachtet werden. Dafür implementiert der Betreiber ein Web-Analytics Dienst, wie z.B. Google Analytics, auf seiner Seite.

Der Analytics-Dienst beginnt seine Arbeit und sammelt für jeden Besucher der Seite eine Vielzahl an Informationen. Sind genug Informationen gesammelt worden, kann der Betreiber sich die Informationen ausgeben lassen. Der Betreiber kann nun über die Oberfläche des Dienstes Informationen auslesen und betrachtet die Seitenaufruf-

Statistik seiner Seite in der letzten Woche. Diese Seitenaufruf-Statistik zeigt ihm, welche seiner Unterseiten am öftesten aufgerufen wurde und wie lange die Besucher beispielsweise auf dieser verblieben sind. In dieser Statistik entdeckt er, dass eine Produktseite besonders selten aufgerufen wird. Auf Basis dieser Informationen kann der Betreiber darauf schließen, dass die Besucher diese Seite vielleicht nicht finden können. Daraus beschließt er, die Werbung für dieses Produkt zu steigern und beispielsweise ein Banner auf der Startseite zu schalten, welches dieses Produkt bewirbt. Ist ein solches Banner eingebaut muss eine Erfolgskontrolle durchgeführt werden. Diese Kontrolle kann ebenfalls über den Dienst durchgeführt werden, in dem zu einem späteren Zeitpunkt die Statistik der Seitenaufrufe erneut aufgerufen wird.

Mit Analytics-Diensten kann also in Erfahrung gebracht werden wie sich Besucher auf der Seite verhalten und wie die Seite besser an diese angepasst werden kann. Es können dabei eine Vielzahl von Informationen erfasst werden. Zum Beispiel an welchen Tagen die meisten Nutzer die Website besuchen, zu welchen Uhrzeiten sich Nutzer für den Newsletter registrieren oder wie viele Nutzer Produkt X gekauft haben. Fast jede Nutzerinteraktion mit der Website kann analysiert werden.

Durch die Nutzung von Web-Analytics kann die Marketing Abteilung des Betreibers erkennen, ob Werbeaktionen Wirkung zeigen und wann der beste Zeitpunkt ist um eine neue Kampagne zu starten. Eine Vernachlässigung dieser Analysen lässt den Betreiber im Dunkeln darüber, welche Bereiche seiner Website weiterentwickelt bzw. verbessert werden müssen, um die Besucher optimal anzusprechen. (Schallaböck 2014)

Das Webtracking bildet die Grundlage für diese Analysen, denn nur durch die Datenerfassung des Tracking kann anschließend ausgewertet und analysiert werden wie Besucher sich verhalten, welche Seiten populär sind und wo es Hürden für mögliche Kunden gibt. Aufgrund der Analyse der Daten wird entschieden in welche Richtung sich die einzelnen Bereiche der Website weiterentwickeln und wie eine bessere Nutzererfahrung erreicht werden kann.

Damit diese Daten eine valide Grundlage bilden können, müssen sie möglichst lückenlos und einheitlich sein. Fehler im Tracking können hierbei falsche Informationen liefern und die Auswertungen der Nutzerdaten negativ beeinflussen. ObservePoint sagt dazu:

„*More than 80% of sites have improperly deployed data collectors at some point, resulting in broken web pages, loss of site traffic, and lost sales.“*

ObservePoint, 2010 (Peterson 2015)

Diese Aussage bezieht sich nicht nur auf das Tracking, sondern auch auf die allgemeine Datenerfassung auf Webseiten. Das Webtracking muss regelmäßig auf die korrekte Datenerfassung untersucht werden, damit gewährleistet ist, dass die Analysen korrekte Einsichten in das Nutzerverhalten liefern.

Wie Webtracking automatisch erkannt und getestet werden kann, um valide Daten innerhalb der Webanalyse zu erhalten, soll Gegenstand dieser Arbeit sein. Das Testen von Webtracking überprüft, ob Tracking Ereignisse ausgelöst werden und ob die dabei übermittelten Daten korrekt sind. Das Ziel ist es eine funktionierende Erweiterung zu einem bereits bestehenden Webtracking-Testsystem zu entwickeln. Dazu folgt in den nächsten Teilen dieses Kapitels eine genauere Aufschlüsselung der Ziele und der Inhalte von folgenden Kapiteln.

1.2 Zielsetzung

Ziel dieser Arbeit ist es, eine Erweiterung zu einem bereits bestehenden Werkzeug zum Testen von Webtracking zu entwickeln, welche später in dieses System, namens Columbo, integriert werden könnte. Das aktuell genutzte System kann hierbei nur manuell erstellte Abläufe an Nutzerinteraktionen auf einer Webseite nachstellen und dabei prüfen, ob das Tracking vorhanden und korrekt war. Zusätzlich bietet das aktuelle genutzte System eine Funktion zum Durchsuchen einer Domain und deren Unterseiten nach Tracking Aufrufen. Die gefundenen Aufrufe werden hierbei jedoch nur auf Existenz überprüft und nicht weiter analysiert. Ziel ist es also, diese beiden Komponenten so zu verbinden, dass die Webseite automatisch durchsucht und dabei auch getestet wird. Dazu werden mögliche Interaktionen der Anwender und Ereignisszenarien identifiziert. Hierbei spielt das automatische Durchsuchen von Webseiten und allen dazugehörigen Unterseiten und das Testen der gefundenen Trackingaufrufe eine übergeordnete Rolle.

Um die Umsetzbarkeit zu demonstrieren, soll eine prototypische Anwendung entwickelt werden, welche die grundlegenden Funktionalitäten erfüllt. Dazu werden grundlegende Funktionen wie das Durchsuchen einer Webseite nach weiteren Unterseiten und das Testen gefundener Trackingaufrufe realisiert.

1.3 Gliederung

Die Arbeit gliedert sich in sechs Abschnitte, wobei der erste Teil die Einleitung darstellt. Hier wird ein kurzer Überblick über das Themengebiet gegeben und die Zielsetzung der Arbeit erläutert.

Den zweiten Abschnitt der Arbeit bilden die Grundlagen. Hier werden Begriffe und Konzepte erläutert, die für das weitere Verständnis der Arbeit von Bedeutung sind.

Die Analyse Ziels, dessen Rahmenbedingungen und Anforderungen ist Bestandteil des dritten Kapitels. Dazu werden Szenarien erläutert, die verschiedene Interaktionen mit dem System beinhalten. Aus diesen werden funktionale und nicht-funktionale Anforderungen identifiziert und beschrieben. Zum Schluss dieses Kapitels, wird anhand von Beispielprojekten die Durchführbarkeit dieser Arbeit überprüft.

Aufgrund der Ergebnisse aus Teil drei werden im vierten Kapitel verschiedene Design-Möglichkeiten vorgestellt und miteinander verglichen. Anhand der identifizierten Anforderungen wird eine Design-Entscheidung getroffen und anhand von UML-Diagrammen erläutert.

Im fünften Abschnitt der Arbeit wird die Realisierung der Designentscheidung anhand von grundlegenden Funktionalitäten durchgeführt. Anschließend wird die Realisierung evaluiert. Hierfür wird untersucht, welche Anforderungen in der Realisierung umgesetzt werden konnten. Des Weiteren wird die Funktionsweise des entwickelten Prototyps vorgestellt.

Der sechste und letzte Teil der Arbeit beinhaltet die Reflexion. Abschließend wird ein Ausblick gegeben bezüglich der in der Zukunft möglichen Erweiterungen.

2 Grundlagen

Dieser Abschnitt der Arbeit befasst sich mit den Grundlagen. Hier werden genutzte Begriffe und die für die Arbeit benötigten Konzepte erläutert, die für das weitere Verständnis der Arbeit von Bedeutung sind. Allgemeine Tracking Konzepte werden hier erklärt und es wird auf das Testen von Webtracking eingegangen.

2.1 Web-Analyse-Dienste

Web-Analyse-Dienste verfolgen und berichten über den Verkehr auf Webseiten. Diese Dienste untersuchen die Herkunft der Besucher, deren Verweilzeiten, die Nutzung von Suchmaschinen und unterstützen mit der Datensammlung die Erfolgskontrolle von Werbekampagnen vom Webseitenbetreiber (Google 2016a).

Abbildung 1 (W3Tech 2016b) zeigt die Verbreitung von Analytics Diensten. Google Analytics ist hierbei der am weitesten verbreitete Service und bietet drei Nutzungsmodelle an. Eine Gratis-Variante, eine kostenpflichtige Premiumvariante namens Analytics 360 und Google Analytics für Mobile Apps.

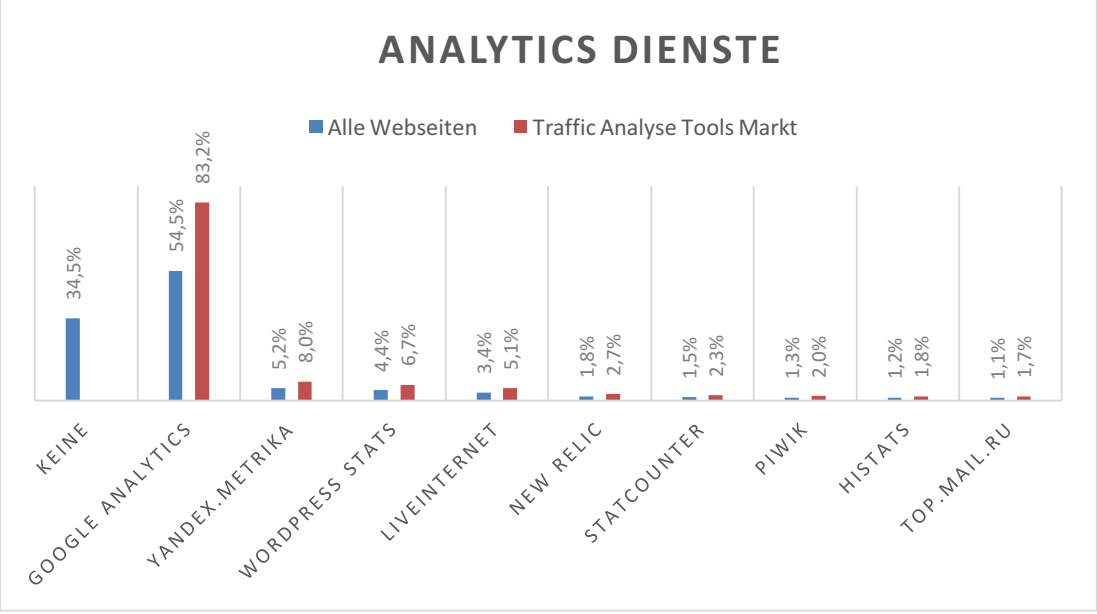


Abbildung 1: Verbreitung Analytics Dienste. (W3Tech 2016b) Stand: 16. Juli 2016

Analytics Werkzeuge bieten hierbei eine Vielzahl an Möglichkeiten an um Berichte anzuzeigen. Es können Dashboards für gelegentliche Nutzer mit den für diese Nutzergruppe relevantesten Informationen angezeigt werden, oder aber auch Analysen in Tiefe zu diesen Informationen durchgeführt werden.

Wie Analytics Dienste auf der technischen Ebene eingebaut werden und funktionieren, wird im folgenden Teil anhand von Google Analytics erläutert.

Google Analytics Funktionalitäten werden über Seitentags implementiert. Diese Tags werden bei Google als Google Analytics Tracking Code bezeichnet. Dieser Code besteht aus einem JavaScript Schnipsel, welches vom Betreiber einer Webseite auf jeder seiner Unterseiten platziert werden muss. Dieser JavaScript Code wird im Browser des Clients ausgeführt, sobald dieser die Seite besucht. Einzige Voraussetzung hierfür ist, dass der Anwender JavaScript aktiviert hat und das Tracking somit nicht blockiert. Der ausgeführte Code sammelt nun Besucherdaten und fordert per Request einen Tracking Pixel an. Der geladene Tracking Pixel enthält keinerlei Bildinformationen und ist meistens transparent. Das Wichtige hierbei ist die Anforderung des Pixels. Bei diesem Request werden Parameter mitgesendet, welche anschließend weiter zu einem Google Datenserver gesendet werden. (Google 2016b) Der ausgeführte Tracking Code lädt anschließend eine größere JavaScript Datei vom Google Server und setzt dort Variablen mit der Kontonummer des Benutzers. Diese JavaScript Datei ist bekannt unter den Namen ga.js oder analytics.js. Diese Datei wird normalerweise beim ersten Seitenaufruf, also dem Beginn eines Besuches auf einer Webseite, geladen und danach im Cache zwischengespeichert. (Google 2016b)

Zusätzlich zum Übertragen von Informationen an den Google Datenserver setzt der Tracking Code Cookies. Diese Cookies werden bei jedem Besucher gesetzt und speichern anonymisierte Informationen, wie z.B. eine ClientId um diesen Besucher später wiedererkennen zu können.

2.2 Tag Manager

Ein **Tag Manager** ist eine Code-Verwaltungsplattform, welche auf einer Webseite ausgespielten Code-Stücke verwaltet. Der Großteil der Webseiten nutzt hingegen keinen Tag Manager. Die meisten Webseiten, die einen Tag Manager nutzen, nutzen den Google Tag Manager, da dieser kostenfrei nutzbar und direkt mit Google Analytics verwendbar ist. Google Tag Manager lässt sich abgekürzt als **GTM** ausdrücken. Die Abbildung 2 (W3Tech 2016a) zeigt eine Übersicht über die Verbreitung von Tag Management Systemen.

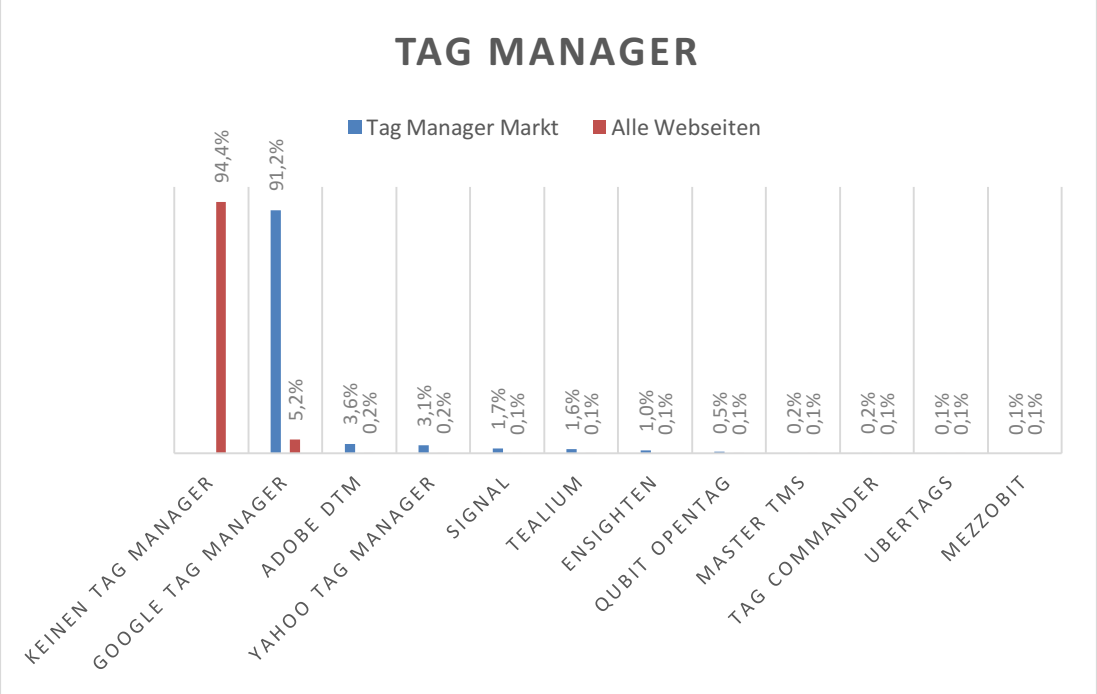


Abbildung 2: Verbreitung Tag Manager. (W3Tech 2016a) Stand: 16. Juli 2016

Die Hauptfunktionalität des Tag Managers wird durch **Tags** umgesetzt. Jeder Tag enthält einen automatisch generierten oder benutzerdefiniert geschriebenen Code, um eine Tracking Funktionalität umzusetzen. Der Tag Manager kann aber auch für weitere Zwecke verwendet werden, um beispielsweise andere Werkzeuge oder Werbungen auf allen Seiten auszuspielen.

Des Weiteren verwaltet der Tag Manager das Ausspielen von diesen Tags über sogenannte Trigger. Jeder Tag im GTM besitzt einen oder mehrere Trigger. Diese Trigger prüfen, ob auf der Webseite eine gewisse Aktion stattgefunden hat oder ob ein bestimmter Status auf der Internetseite erreicht wurde. Trigger können einfache Klicks, Eingaben in Felder oder auch komplexere Codes sein. Diese Trigger beinhalten meistens Bedingungen. Ist diese Bedingung erfüllt, wird der Tag abgesendet und die benötigten Informationen getrackt. (Yeskow 2015)

Der Tag Manager erleichtert es, auf größeren Seiten flexibel Code auszuspielen, da auf jeder Seite nur einmal der Code des Tag Managers platziert werden muss. Ist der Code dort einmal platziert, können Änderungen über eine Weboberfläche durchgeführt werden, sodass der Quelltext der einzelnen Seiten nicht mehr direkt

angepasst werden muss. Alle Änderungen, die im Tag Manager vorgenommen werden, wirken sich anschließend auch global auf alle Seiten aus, die den Tag Manager Code enthalten. Über diese Funktionalität des GTM können auch andere, nicht für das Tracking relevante, Code-Schnipsel in die Seite eingebunden werden. Dies können beispielsweise auch Marketing Tools oder andere Analytics Werkzeuge sein. (Google n.d.)

Eine Beispielabfolge mit einem Tag Manager wäre: Der Tag Manager erkennt, dass die Seite aufgerufen wird. Der Tag wurde so konfiguriert, dass er beim Seitenaufruf auslöst und feuert anschließend einen Google Analytics Tag ab. Durch dieses Abfeuern des Tags wird dieser auf der Seite aktiviert. Anschließend sammelt er die benötigten Informationen und sendet diese weiter an den Google Analytics Server.

Die Abbildung 3 (To 2014) zeigt diesen Prozess umgesetzt über einen **Tag Container**, der alle Tags für eine Webseite beinhaltet. Dieser Tag Container befindet sich in diesem Beispiel in dem Google Tag Manager. Dort wird sichtbar, dass der Tag Manager alle weiteren verfügbaren Trigger prüft und nur die feuert, die auch alle Bedingungen erfüllen.

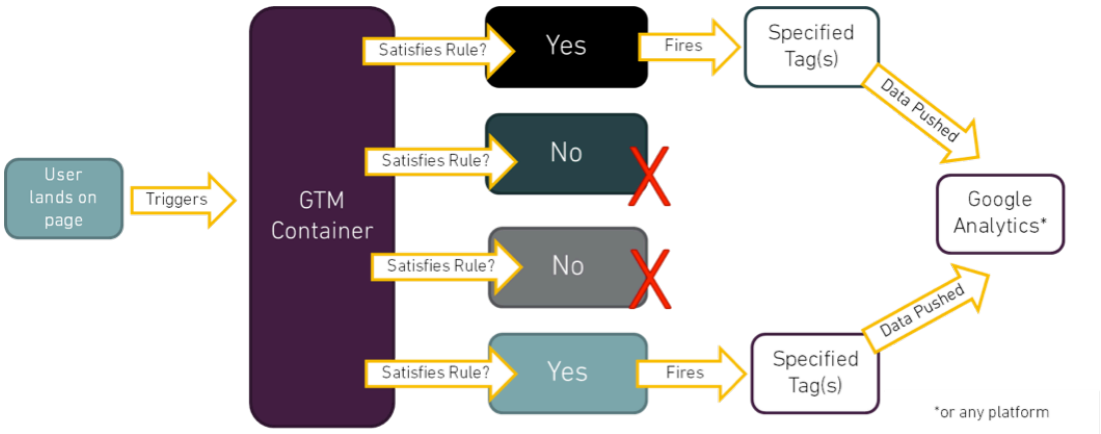


Abbildung 3: Funktionsweise eines Tag Managers (To 2014)

2.3 Webtracking

Beim Webtracking geht es darum, dass Besucher von Webseiten durch sogenannte Tracker verfolgt werden, wenn sie im World Wide Web unterwegs sind. Diese Tracker sammeln hierbei Daten über das Nutzerverhalten und die Nutzung der Webseiten. Tracker können dabei sehen, welche Webseiten von Besuchern aufgerufen werden, wie lange diese auf einer Seite verweilen und sogar ihr Verhalten aufzeichnen. Den Nutzerdatensammlern, auch **Tracker** genannt (Markus Schneider, Matthias Enzmann 2014a), sind hierbei kaum Grenzen gesetzt, da diese durch benutzerdefinierte Skripte erweitert werden können. (Yeskow 2015)

Webtracking funktioniert auch als **Cross-Domain-Tracking** über mehrere Webseiten unterschiedlichster Anbieter hinweg. Diese Datensammlung spielt sich im Hintergrund ab, ohne das der Nutzer davon weiß. Tracker tun dies, um Verbraucherdaten zu bekommen. Diese Daten sind ein wichtiger Rohstoff für viele Unternehmen. (Markus Schneider, Matthias Enzmann 2014a)

Wenn man weiß, welche Seiten ein Nutzer abruft, dann weiß man auch, für was er sich interessiert, was er gerne konsumiert, welche politische Einstellung er hat, wie alt er ungefähr ist, ob er allein lebt oder Familie hat, welchen Lebensstil er pflegt und vieles mehr.

Frauenhofer Tracking Report (Markus Schneider, Matthias Enzmann 2014a)

All diese Daten werden beim Webtracking über die Tracker gesammelt und anschließend an Analytics-Werkzeuge wie Google Analytics übermittelt. Diese Informationen werden beispielsweise im Marketing für zielgerichtete und personalisierte Werbung benötigt.

2.4 Funktionsweise des Webtracking

Beim Webtracking geht es hauptsächlich um das Zusammenspiel von Besucher und Betreiber. Der Betreiber möchte Informationen von den Besuchern sammeln, um diese besser verstehen zu können und um deren Erlebnis auf der Webseite zu optimieren. Durch die Sammlung von Nutzerdaten kann dieser seine Webseite beispielsweise so optimieren, dass er in seinem Onlineshop langfristig durch angepasste Werbung mehr Verkäufe erzielt und somit seinen Gewinn erhöht. Auf der anderen Seite ist währenddessen der Besucher die Person, die wirklich mit der Webseite interagiert. Der Besucher bekommt von der Datensammlung des Betreibers wenig bis überhaupt nichts mit und besucht diese Webseite wie jede andere. Der Besucher hinterlässt jedoch einen Abdruck auf der Webseite, indem

dieser sein Verhalten und persönliche Informationen indirekt an die Tracker weitergibt.
Ruft ein Besucher die Webseiten eines Betreibers ab, dann empfängt der Besucher zusätzlich zur ausgelieferten Webseite Code-Abschnitte, welche in dieser eingebunden sind. Dieser Code ist für das Webtracking in die Seite implementiert worden. Dies kann manuell oder aber auch durch Tag-Management-Systeme wie Google Tag Manager automatisiert geschehen. (Yeskow 2015)

Der Code dient nun als Schnittstelle zwischen den Interaktionen des Besuchers und dem Tracker. Der Browser des Nutzers führt die enthaltenen Codezeilen aus und baut anschließend eine Verbindung zu einem Tracker auf. Diese Verbindung mit einem Tracker wird anschließend dazu genutzt um für jede Aktion und Bewegung des Nutzers einen Pixel über eine HTTP-Anfrage zu laden. Dieses Bild ist leer und hat nur eine Größe von 1x1 Pixel. Die wichtigen Informationen befinden sich hierbei nicht in dem Bild, sondern in den Parametern, die bei der Anfrage mit an den Tracker gesendet werden. (Google 2016b) Dieser Ablauf ist schematisch in Abbildung 4 (Markus Schneider, Matthias Enzmann 2014a) dargestellt.

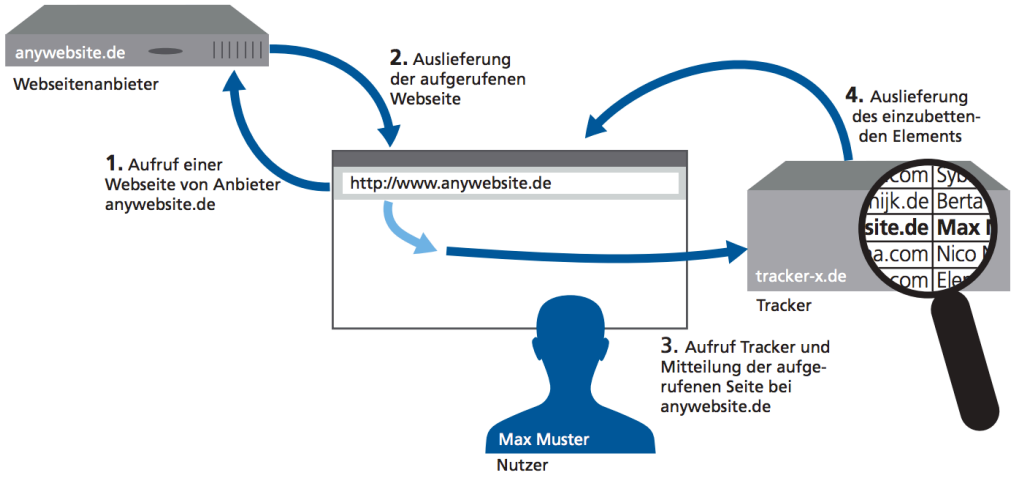


Abbildung 4: Schematische Darstellung des Webtracking (Markus Schneider, Matthias Enzmann 2014a)

Eine HTTP-Anfrage enthält mehrere URL-Parameter, welche die relevanten Informationen an den Tracker liefern (Johnson 2008). Eine Anfrage könnte beispielsweise folgendermaßen aussehen: Max Mustermann ist bei www.anbieterxy.com als Nutzer registriert, wo er nach Informationen zur Behandlung einer Krankheit sucht.

Dazu befindet er sich auf einer Seite mit der URL:
www.anbieterxy.com/krankheit.html?vorname=Max&name=Muster

Werden von dieser Seite aus Verbindungen zu einem Tracker aufgebaut, dann enthält die Anfrage typischerweise einen GET Befehl an www.trackerxy.com mit dem HTTP-Verweis. Dieser Verweis überbringt dem Tracker nun die gesammelten Informationen. Der Verweis sieht im Beispiel wie folgt aus:
www.anbieterxy.com/krankheit.html?vorname=Max?name=Muster

Dieser Verweis enthält alle benötigten Informationen als URL-Parameter, um diese anschließend verarbeiten zu können. Der Empfänger, in diesem Fall der Tracker, kann nun aus dem Seitenaufruf schließen, dass sich der registrierte Benutzer Max Muster für die Behandlung einer Krankheit interessiert,

2.5 Testen von Webtracking

Dieser Abschnitt befasst sich damit, was bei Webtracking getestet werden muss und welche Werkzeuge und Möglichkeiten es hierfür gibt.

2.5.1 Blackbox Testen

Beim Testen von Tracking Parametern ist der ausgeführte Tracking Code meistens unbekannt. Insbesondere wenn der Tracking Code über ein Tag Management System ausgespielt wird. Deshalb handelt es sich bei diesem Testvorgang um ein sogenanntes Blackbox Testen. Der Tester weiß also nichts über das Innere des Systems, hierbei wird also nur die abstrakte Funktion getestet. Dem Tester stehen in diesem Fall nur die Spezifikationen des Tracking zur Verfügung. (Jacken 2010)

In Bezug auf das Webtracking bedeutet dies, dass der Tester einen Input in Form einer Interaktion mit einer Webseite kennt und daraus ein vorher in der Spezifikation definierter Output resultieren muss. Dieser Output stellt hier das Absenden von gesammelten Informationen an den Webanalyse-Dienst dar.

2.5.2 Testobjekt

Das Testobjekt ist beim Webtracking der Request nach einem Tracking Pixel. Der Request des Pixels ist der Informationsträger, der alle Daten beinhaltet. Es muss hier getestet werden, ob dieser Pixel zum einen überhaupt existiert und ob er beim Request die richtigen Parameter mitsendet. Die Struktur der Parameter dieses Request ist bei jedem Anbieter unterschiedlich und die einzelnen Parameter können

auch benutzerdefiniert erstellt werden. Im folgenden Beispiel wird die Webseite der Google Developers aufgerufen und die Daten eines Google Analytics Requests für den Tracking Pixel über die Entwicklerkonsole abgefangen.

Der gesendete Request nach einem Tracking Pixel in Abbildung 5, aufgezeichnet mit Hilfe der Entwickler Konsole im Google Chrome Browser, beinhaltet alle Informationen, die an den Server von Google in URL-kodierter Form gesendet werden. Darunter werden diese Parameter vom Debug-Skript mit ihren richtigen Bezeichnern versehen und aufgelistet.

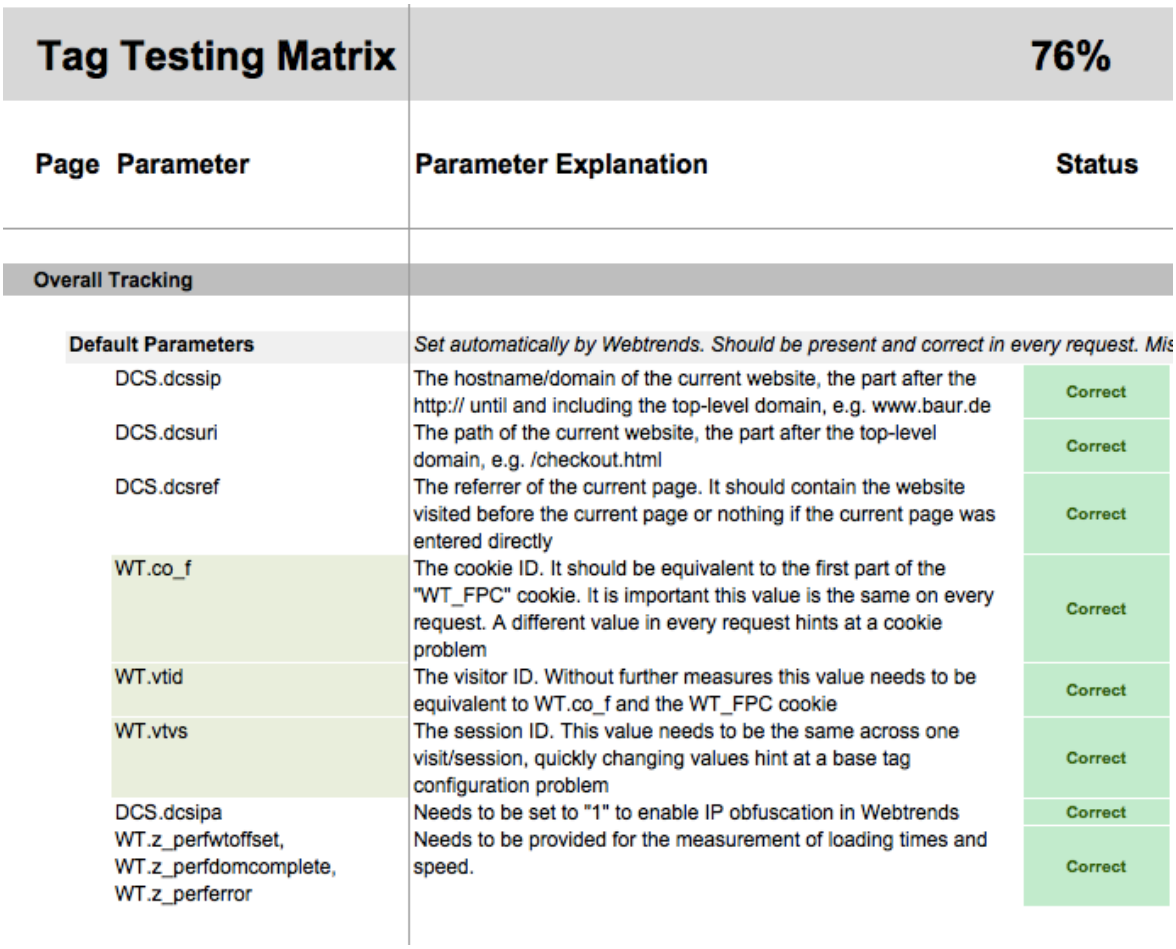
```
Sent beacon:
v=1&_v=j44d&a=786322177&t=pageview&_s=1&dl=https%3A%2F%2Fdevelopers.google.com
%2F&ul=de&de=UTF-8&dt=Google%20Developers&sd=24-
bit&sr=1440x900&vp=762x801&je=0&fl=22.0%20r0&_u=QDCAAAIhI~&jid=&cid=940199653.1437
653993&tid=UA-49880327-1&cd5=de&cd3=0&cd1=Signed%20out&z=579193296


_j1              (&jid)
adSenseId        (&a)   786322177
apiVersion       (&v)   1
clientId         (&cid) 940199653.1437653993
dimension1       (&cd1) Signed out
dimension3       (&cd3) 0
dimension5       (&cd5) de
encoding         (&de)  UTF-8
flashVersion     (&fl)  22.0 r0
hitType          (&t)   pageview
javaEnabled      (&je)  0
language         (&ul)  de
location         (&dl)  https://developers.google.com/
screenColors     (&sd)  24-bit
screenResolution (&sr)  1440x900
title            (&dt)  Google Developers
trackingId       (&tid) UA-49880327-1
viewportSize     (&vp)  762x801
```

Abbildung 5: Tracking Parameter eines Seitenaufrufs

2.5.3 Manuelles Testen

Beim manuellen Testen werden Tracking Parameter händisch auf ihre Korrektheit untersucht. Der Ablauf ist hierbei meistens strikt vorgegeben und dieser muss genauso vom Tester abgebildet werden, damit die Parameter korrekt getestet werden können. Der Tester nutzt hierfür eine Checkliste an Ereignissen und dazugehörigen Parametern, die in dem zu testenden Request vorhanden sein müssen. Der vorbestimmte Ablauf an Aktionen wird anschließend durchgeführt und der daraus resultierende Tracking-Request muss mit den Soll-Parametern der Checkliste abgeglichen werden. Diese Checklisten werden bei Peaks & Pies **Tag Testing Matrix** genannt. Listen wie diese werden jedoch auch von anderen Firmen

genutzt, wie z.B. AT Internet. Eine Version dieser Tag Testing Matrix ist in Abbildung 6 dargestellt (Peaks&Pies 2016b). Diese Tabellen müssen händisch von einem Arbeiter abgearbeitet und ausgefüllt werden und benötigen viel Arbeitszeit. Im Gegensatz dazu bietet Columbo Szenarien zum automatische Überprüfen einzelner Interaktionsabfolgen.



Tag Testing Matrix		76%
Page Parameter	**Parameter Explanation**	**Status**
Overall Tracking		
Default Parameters	*Set automatically by Webtrends. Should be present and correct in every request. Mis*	
DCS.dcssip	The hostname/domain of the current website, the part after the http:// until and including the top-level domain, e.g. www.baur.de	Correct
DCS.dcsuri	The path of the current website, the part after the top-level domain, e.g. /checkout.html	Correct
DCS.dcsref	The referrer of the current page. It should contain the website visited before the current page or nothing if the current page was entered directly	Correct
WT.co_f	The cookie ID. It should be equivalent to the first part of the "WT_FPC" cookie. It is important this value is the same on every request. A different value in every request hints at a cookie problem	Correct
WT.vtid	The visitor ID. Without further measures this value needs to be equivalent to WT.co_f and the WT_FPC cookie	Correct
WT.vtvs	The session ID. This value needs to be the same across one visit/session, quickly changing values hint at a base tag configuration problem	Correct
DCS.dcsipa	Needs to be set to "1" to enable IP obfuscation in Webtrends	Correct
WT.z_perfwtoffset, WT.z_perfdomcomplete, WT.z_perferror	Needs to be provided for the measurement of loading times and speed.	Correct

Abbildung 6: Tag Testing Matrix (TTM) (Peaks&Pies 2016b)

2.5.4 Halb-Automatisiertes Testen

Beim halb-automatisierten Testen werden zur Untersuchung des auf einer Seite installierten Tracking externe Werkzeuge verwendet. Werkzeuge wie Columbo von Peaks & Pies oder Tag Inspector (InfoTrust 2016) zeigen die verwendeten Tracking-Anbieter und die prozentuelle Verbreitung dieser auf allen Unterseiten einer Website an. Zusätzlich können bei Columbo Abläufe von Nutzerinteraktionen mit der Seite als Szenarien nachgestellt werden, um daraus resultierende Tracking-Requests zu

testen. Diese Szenarien können Beispielsweise den kompletten Checkout-Prozess nachstellen. Die einzelnen Schritte, Seitenaufrufe, Klicks und Eingaben müssen manuell erstellt werden bevor sie automatisiert und regelmäßig ausgeführt werden können. Zum Testen werden hierbei zwischen den Schritten einzelne Tests zum Überprüfen der Parameter eingebaut. Diese Art der Überprüfung wird hauptsächlich bei großen Webseiten genutzt, wo die manuelle Überprüfung zu aufwändig wäre und Lücken schnell übersehen werden könnten. Die regelmäßige und automatische Ausführung dieser Szenarien bringt ein aktuelles und archivierbares Reporting über den Status des verbauten Tracking.

2.5.5 Voll-Automatisiertes Testen

Das vollautomatische Testen geht, im Vergleich zum halb-automatisierten Testen, noch einen Schritt weiter und versucht Tracking-Aufrufe zu erkennen und direkt zu testen. Sind zum Beispiel Buttons auf einer Seite eingebaut und mit Tracking hinterlegt, erkennt das Framework dies automatisch und überprüft, ob die dazugehörigen Tracking Parameter stimmen und die gesendeten Daten damit valide sind. Diese Art des Testens wird im nächsten Teil dieser Ausarbeitung weiter vertieft.

2.6 Werkzeuge

Dieser Unterpunkt beschäftigt sich mit den gewählten Werkzeugen und Frameworks und die Beweggründe diese für die Entwicklung des Prototyps zu verwenden. Zusätzlich dazu werden hier weitere in Verbindung auftretende Begriffe erläutert.

2.6.1 Selenium

Selenium ist eine Testumgebung für Webanwendungen, welche von der Firma ThoughtWorks entwickelt und als freie Software unter der Apache-2.0-Lizenz veröffentlicht wurde. Selenium bietet eine Vielzahl an Funktionalitäten und Bibliotheken zur Automatisierung von Interaktionen mit Webbrowsern. Selenium kann mithilfe des WebDrivers über Java Code benutzt werden und somit in größere Programme integriert werden. (Selenium 2016)

2.6.2 PhantomJS

PhantomJS ist ein skriptbasierter, kopfloser Browser zur Automatisierung von Interaktionen mit Webseiten. PhantomJS wird bei diesem Prototyp in Verbindung mit Selenium verwendet. Diese Verbindung von PhantomJS und Selenium wird über einen extra PhantomJS Treiber für Selenium hergestellt. PhantomJS kann genutzt werden um Webseiten zu öffnen, Bildschirmfotos zu machen, Nutzerinteraktionen auszuführen und um in den Seitenkontext injizierten JavaScript Code auszuführen. (Hidayat 2016)

Mit PhantomJS können alle benötigten Interaktionen mit Webseiten der Komponenten realisiert werden und mithilfe der kopflosen Funktionalität im Hintergrund ausgeführt werden. Eine besonders nützliche Verwendung für einen kopflosen Browser ist ein **Crawler**. Mit Hilfe eines Crawlers kann sich ein Programm beispielsweise im Hintergrund durch alle Unterseiten einer Webseite arbeiten und diese dabei indizieren.

2.6.3 BrowserMobProxy

BrowserMobProxy basiert auf Technologie aus Selenium und ermöglicht es Netzwerkverkehr mitzuschneiden, zu manipulieren, zu simulieren oder HTTP-Requests neuzuschreiben. (Lightbody 2016) Für den Prototyp wird der BrowserMobProxy genutzt um Netzwerkverkehr aufzuzeichnen und im HAR Format zu speichern.

2.6.4 HAR – HTTP Archive

Das HTTP Archive Format (HAR) bietet sich für die Speicherung von Netzwerkdaten an, da dort alle resultierenden HTTP-Transporte gespeichert werden können. Das HAR ist ein JSON-formatiertes Archiv für das Mitschreiben von Aktionen eines Webbrowsers. Die Spezifikation dieses Formats wird zurzeit noch durch eine Arbeitsgruppe des World Wide Web Consortium (W3C) bearbeitet und weiterentwickelt. (W3C 2012) Mithilfe dieses Formats können alle Browserdaten gespeichert und anschließend weiterverarbeitet werden können.

3 Analyse

Ziel dieser Analyse ist es, das bereits bestehende Werkzeug Columbo zu betrachten und daraus Anforderungen für die automatische Erkennung und Prüfung von Webtracking zu identifizieren und zu untersuchen. Hierfür wird zunächst das bestehende Werkzeug Columbo vorgestellt, bevor einige Beispielszenarien vorgestellt werden. Aus diese Szenarien werden anschließend die sich aus ihnen ergebenden Anforderungen ausformuliert.

3.1 Vorhandenes System zum Testen von Webtracking

Columbo ist ein Werkzeug, das von Peaks & Pies GmbH entwickelt und betreut wird. Columbo ruft beim Prüfen die Websites auf wie jeder andere Browser auch. Dadurch werden bei jeder Interaktion mit einer Webseite weiterhin alle Tracking-Tools verschiedenster Anbieter ausgelöst. (Peaks&Pies 2016a)

Columbo hat mehrere Möglichkeiten Tracking Funktionen zu testen. Zum einen bietet es den **Scanner**. Dieser Scanner beginnt auf einer Startseite und arbeitet sich anschließend durch eine vorher definierte Anzahl an Unterseiten. Auf jeder gefundenen Seite prüft er, welche Tracking-Werkzeuge dort eingebunden sind und stellt diese Informationen in einer Übersicht dar.

Die zweite Funktionalität wird über sogenannte **Szenarien** abgebildet. Hier wird manuell eine Schrittfolge an Interaktionen mit einer Webseite abgebildet und dabei werden Tests auf die dabei gefundenen Tracking Requests ausgespielt, um deren Parameter auf Korrektheit zu prüfen.

Ein solcher Ablauf an Nutzerinteraktionen wird in Abbildung 7 (Peaks&Pies 2016a) dargestellt. Dort wird als Erstes ein Seitenaufruf durchgeführt und nach diesem Schritt direkt ein Test ausgeführt. Dieser Test soll überprüfen, ob ein Seitenaufruf Tracking Request gesendet wurde und ob dieser die korrekten Parameter beinhaltet. Anschließend werden alle bis dahin angekommenen Trackingaufrufe gelöscht, da diese nach dem Test nicht mehr weiter für den Ablauf des Szenarios relevant sind. Als weitere Nutzerinteraktion nach dem Seitenaufruf kann in Columbo auch ein Klick auf ein Element auf der Seite simuliert werden. Dieser Klick wird in diesem Fall auf das Element mit der ID show_contact_form ausgeführt. Anschließend wird auch hier wieder ein Test dazu ausgeführt.

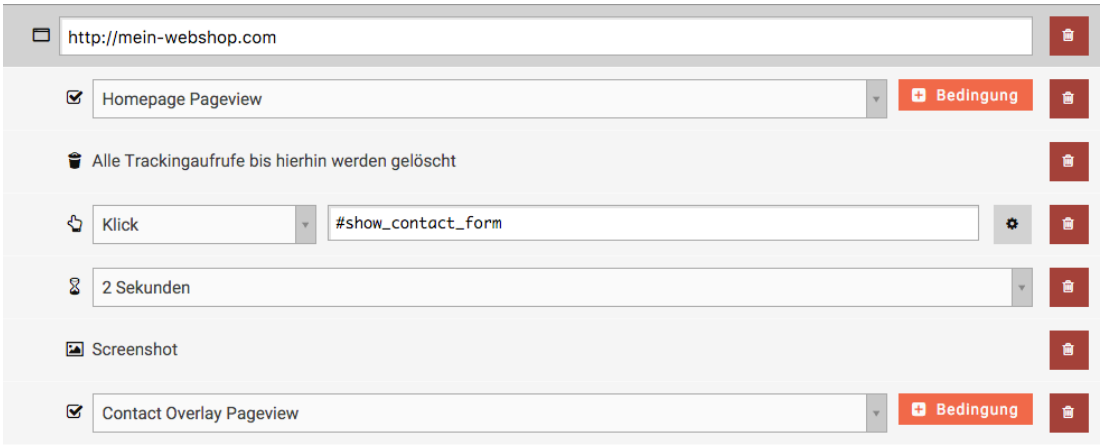


Abbildung 7: Szenario in Columbo

Columbo kann diese beiden Hauptfunktionen zurzeit nicht miteinander verknüpfen. Wünschenswert wäre es für Columbo, diese beiden Funktionalitäten zu verbinden, sodass automatisiert Seiten nach konkreten Tracking Funktionen durchsucht und diese ebenfalls getestet werden können. Zur weiteren Orientierung am Columbo Werkzeug werden Grundlegende Funktionalitäten, wie der Crawler, ebenfalls in den Prototypen übernommen und auch in den Anforderungen aufgeführt.

3.2 Szenarien

In diesem Abschnitt sollen für die Erweiterung der Funktionalitäten von Columbo mögliche Szenarien erläutert werden, anhand derer unterschiedlichste Interaktionen erkennbar werden, die für das Tracking relevant sind.

Tracking gefunden

In diesem Szenario wird beschrieben wie sich das Werkzeug verhält, wenn erfolgreich Tracking gefunden wurde. Hat das Werkzeug korrekt funktionierendes Tracking gefunden, dann soll der Anwender darüber informiert werden und eine Übersicht der gefundenen Tracking-Requests angezeigt bekommen. Diese Informationen sollen zusätzlich chronologisch einsortiert werden, um vorherige Ergebnisse mit den Aktuellen zu vergleichen.

Tracking nicht gefunden

In diesem Szenario wird nun beschrieben, was geschehen soll, wenn das Gegenteil von Szenario „Tracking gefunden“ 3.1.1 eintritt und eine Trackingfunktion nicht

korrekt funktioniert. Als Grundlage für diese Einsicht muss der Anwender ein Wissen darüber besitzen, welche Tracking Funktionen auf der Webseite eingebaut sind und welche nicht existieren. Dieses Grundwissen kann sich auf simple Informationen wie „alle Button-Clicks werden getrackt“ oder „alle Seitenaufrufe werden standardmäßig getrackt“ beschränken. Werden bei der Ausführung nun also fehlende oder fehlerhafte Tracking Aufrufe gefunden, kann dies zwei mögliche Gründe haben. Diese beiden Möglichkeiten werden im folgenden Teil genauer aufgeführt.

Zum einen kann es sein, dass diese Trackingfunktion nicht korrekt funktioniert und zusätzlich fehlerhafte Parameter sendet. Dies sollte durch die Tests und den Vergleich mit den vorherigen Testdurchläufen erkannt werden. Schlägt ein Test fehl, der beim vorherigen Testdurchlauf erfolgreich war, muss der Anwender benachrichtigt werden, da diese Trackingfunktion gegebenenfalls ausgefallen ist. Hier ist nun ein Eingreifen vom Anwender nötig um diese Funktion wiederherzustellen.

Die zweite Möglichkeit ist, dass diese Tracking Funktion nicht eingebaut ist. Wird also zum Beispiel kein Tracking für Buttons gefunden, da dieses nicht vom Anwender implementiert ist, soll der Nutzer einen Hinweis erhalten, dass diese Funktion für diese Seite empfehlenswert sein könnte. Der Nutzer erhält also durch die Nutzung des Werkzeuges eine Übersicht darüber, welche weiteren Tracking Funktionalitäten für diese Seite zurzeit nicht eingebaut sind. In dieser Übersicht kann der Nutzer auch sehen, wie viele Möglichkeiten er verpasst. Werden mehrere Seiten erkannt auf denen ein Button Tracking sinnvoll wäre, erhält diese Funktion eine höhere Anzahl verpasster Möglichkeiten. Durch dieses Hochzählen und Mitverfolgen von nicht vorhandenem Tracking kann eine Rangliste erstellt werden. Mit Hilfe dieser Rangliste kann der Anwender nun erkennen, welcher Einbau einer neuen Tracking-Funktion den größten Nutzen bringen würde.
Diese Art der Empfehlungen unterstützt den Anwender dabei sein Tracking Konzept auszubauen und auf weitere, ggf. bisher unbekannte Möglichkeiten auszudehnen.

3.3 Anforderungsanalyse

Aufgrund der zwei erstellten Szenarien ergeben sich funktionale- und nicht-funktionale Anforderungen an ein Werkzeug zum automatisierten Testen von Webtracking. In dem folgenden Teil dieses Kapitels werden diese Anforderungen aufgeführt und beschrieben. Diese Anforderungen definieren die Funktionen und Eigenschaften des Werkzeuges.

3.3.1 Funktionale Anforderungen

Anhand der funktionalen Anforderungen werden die gewünschten Funktionalitäten und das Verhalten des Systems beschrieben.

Durchsuchen von Webseite

Das System soll einem Anwender ermöglichen seine Webseite automatisiert nach weiteren Unterseiten zu durchsuchen lassen, um anschließend Tests auf den gefundenen Seiten durchzuführen. Dieses Durchsuchen sollte im Hintergrund geschehen, da das visuelle Anzeigen der Seite in einem normalen Browserfenster für die weitere Nutzung nicht notwendig ist.

Anpassen von Einstellungen

Für das Festlegen von Einstellungen benötigt der Anwender eine Oberfläche, welche Ihm Eingaben bezüglich des Startpunktes und des täglichen Limits des Crawlers ermöglicht.

Erkennung des Tracking

Auf denen durch den Crawler gefundenen Seiten soll geprüft werden, ob Tracking Funktionalitäten eingebaut sind. Hierfür muss zunächst erkannt werden, ob auf diesen Seiten Tracking Aufrufe an die Tracker gesendet werden.

Prüfung des Tracking

Nachdem Tracking erkannt wurde, muss zusätzlich noch getestet werden ob diese Funktionen auch den Anforderungen entsprechend arbeiten und richtige Informationen senden.

Ausgabe Ergebnisse

Die Ergebnisse der Erkennung und der Prüfung soll dem Anwender dargestellt werden können. Der Anwender soll hier sehen können, auf wie vielen und welchen Seiten Tracking überhaupt eingebaut ist und auf welchen Seiten Tests fehlgeschlagen sind.

Ausgabe Statistik

Zusätzlich zur Ausgabe der Ergebnisse sollte der Nutzer Einsicht in eine Statistik über die Ergebnisse der vorherigen Durchläufe erhalten können, die alle erfolgreichen und auch fehlgeschlagenen Tests dokumentiert.

Benachrichtigung bei Ausfall

Wird durch das System ein Ausfall des Webtracking festgestellt, soll der Anwender darüber informiert werden, um zeitnah eine Reparatur durchführen zu können. Diese Benachrichtigung könnte der Nutzer über verschiedene Wege erhalten könnte. Vorstellbar wären hier zum einen eine Benachrichtigung per E-Mail oder zum anderen eine Push-Benachrichtigung aufs Smartphone.

Empfehlungen anzeigen

Auf Basis der Ergebnisse des Werkzeuges sollen dem Anwender Empfehlungen angezeigt werden. Diese Empfehlungen sollen graphisch dargestellt werden und erkennbar machen, welche weiteren Tracking Funktionalitäten empfehlenswert sind. Hierfür müssen die Ergebnisse der Erkennung ausgewertet und anschließend zusammengefasst visualisiert werden.

3.3.2 Nicht-Funktionale Anforderungen

Nicht-funktionale Anforderungen beschreiben Attribute des funktionalen Verhaltens, also wie gut bzw. mit welcher Qualität das System seine Funktion erbringen soll. Ihre Umsetzung beeinflusst stark, wie zufrieden der Kunde bzw. Anwender mit dem Produkt ist und wie gerne er dieses einsetzt. Damit besitzen sie eine große Auswirkung auf den Verbrauch von Ressourcen, Entwicklung und Wartung. Zusätzlich dienen diese Anforderungen dazu, die Akzeptanz des Systems gegenüber dem Anwender zu erhöhen.

Verfügbarkeit

Die Zuverlässigkeit ist eine Grundvoraussetzung, die beachtet werden muss. Für die Akzeptanz des Systems ist die Zuverlässigkeit ebenfalls unerlässlich. Dabei muss das korrekte Verhalten des Systems und der dazugehörigen Oberfläche stets sichergestellt sein.

Aufgrund der Zuverlässigkeit sollte das System nur selten in einen fehlerhaften Zustand geraten können. Sollte dies jedoch trotzdem einmal geschehen, muss es dem Anwender möglich sein, das System wieder in einen korrekten Zustand zurückzusetzen.

Sollte es doch zu einem Fehler kommen, muss das System zusätzlich in einen gesicherten Zustand übergehen und alle Aktivitäten unterbrechen. Es wäre fatal, wenn es durch einen Fehler korrektes Tracking als fehlerhaft erkennt oder auch anders herum. Das zu entwickelnde Werkzeug sollte nur selten in solche Zustände geraten, da die Überprüfung und Überwachung des Tracking nicht unterbrochen

werden darf. Diese Überwachung muss ein konstanter Prozess sein, wobei Fehler und nicht Verfügbarkeit des Systems zu fehlerhaften Rückschlüssen auf die Funktionalität des Tracking führen könnte.

Sicherheit

Die Sicherheit der abgelegten Daten, spielt eine große Rolle für die Anwender. Da bei den Anwendergruppen hauptsächlich auf Unternehmen abgezielt wird und das Webtracking ein komplett intern stattfindender Prozess ist, müssen die gesammelten Daten geschützt werden.

Diese Daten enthalten sensitive Daten über Geschäftsprozesse und dürfen somit nicht für externe Personen oder Unternehmen einsehbar sein. Hier muss mit modernen Sicherheitsrichtlinien darauf geachtet werden, dass jede Person, die das System nutzt und damit interagiert, nur auf die Daten zugreifen kann, die für ihn auch bestimmt sind.

Für den Zugriff auf die Funktionalitäten der Anwendung muss der Anwender sich authentifizieren. Dazu stellt die Anwendung ein Login in Form eines Webformulars zur Verfügung. Mithilfe dieses Formulars kann der Nutzer sich unter Angabe seiner Nutzerdaten im System registrieren. Diese Informationen beinhalten dabei mindestens den für die Identifikation des Nutzers und die Authentifizierung notwendigen Benutzernamen und ein Passwort.

Wartbarkeit

Für eine hohe Software-Wartbarkeit muss eine tragfähige und flexible implementierte Architektur vorhanden sein, um eine Grundlage für die Wartung schaffen zu können. Des Weiteren sollte der Code verständlich und so geschrieben sein, dass kleine Änderungen an der Spezifikation auch nur kleine Änderungen am Code zur Folge haben. Eine detaillierte Dokumentation, wie zum Beispiel ein JavaDoc, unterstützen die Wartung, insbesondere durch Dritte oder weitere Entwickler.

Die Umgebung, in die sich das System integrieren soll, ist einem ständigen Wandel ausgesetzt. Tracking Systeme entwickeln sich kontinuierlich weiter und es erscheinen stetig neue auf dem Markt. Dem Anwender muss es also möglich sein, neue Versionen und Webtracking-Systeme in das System zu integrieren, ohne diese explizit einstellen zu müssen. Auf der anderen Seite muss dem Anwender das Entfernen veralteter oder außer Betrieb genommener Systeme genau so leicht gemacht werden wie das Einfügen.

Um dies gewährleisten zu können, ist eine lose Kopplung der einzelnen Tracking-Systeme notwendig. Dadurch stehen die grundlegenden Funktionalitäten während des Hinzufügens, Entfernens oder Austauschs der einzelnen Komponenten weiterhin zur Verfügung.

Zur Erweiterbarkeit gehört nicht nur der Ausbau von Funktionalitäten durch die Hardware, sondern auch die Erweiterung durch Software. Dem Entwickler wird hier durch definierte Schnittstellen ermöglicht, das System um weitere Funktionalitäten zu erweitern.
Im Rückblick auf die bisherigen nicht-funktionalen Anforderungen, muss es möglich sein, die Zuverlässigkeit und die Sicherheit nachträglich zu verbessern.

Benutzbarkeit

Bedingt durch den Umfang an Funktionen benötigt die zu entwickelnde Anwendung ein möglichst einfaches, benutzerfreundliches und eindeutig strukturiertes Layout. Beim Entwurf und der Entwicklung der Präsentationsschicht sollten deshalb mehrere Faktoren beachtet werden.

- Verwendung eines übersichtlichen und statischen Grundlayouts
- Kontinuierliche Verwendung des Layouts für alle Bestandteile
- Möglichst schmale und leichte Präsentationsschicht
- Unterstützung des Anwenders durch Texthinweise

Skalierbarkeit

Die Anwendung sollte möglichst unempfindlich gegenüber Änderungen des Datenmodells sein, um bei nötige Erweiterungen des Funktionsumfanges problemlos bewältigen zu können.

4 Design der Komponenten

Im vorherigen Kapitel wurden anhand von Szenarien Anforderungen für das zu entwickelnde Werkzeug definiert. Auf der Basis dieser Anforderungen sollen im folgenden Kapitel die einzelnen Komponenten entworfen werden. Das zu entwickelnde Werkzeug soll sich hierbei an den bereits existierenden Funktionen von Columbo orientieren und verschiedene Vorgehensweisen von diesem übernehmen.
Grundlegend gliedert sich der Entwurf in drei Hauptkomponenten (vgl. Abbildung 8). Diese Komponenten bilden die einzelnen Funktionalitäten des Werkzeuges ab und arbeiten automatisiert zusammen. Die Crawler-Komponente sucht nach Links auf Webseiten, der Tester überprüft die Webtracking-Requests und die Empfehlungskomponente verarbeitet anschließend die Testergebnisse.

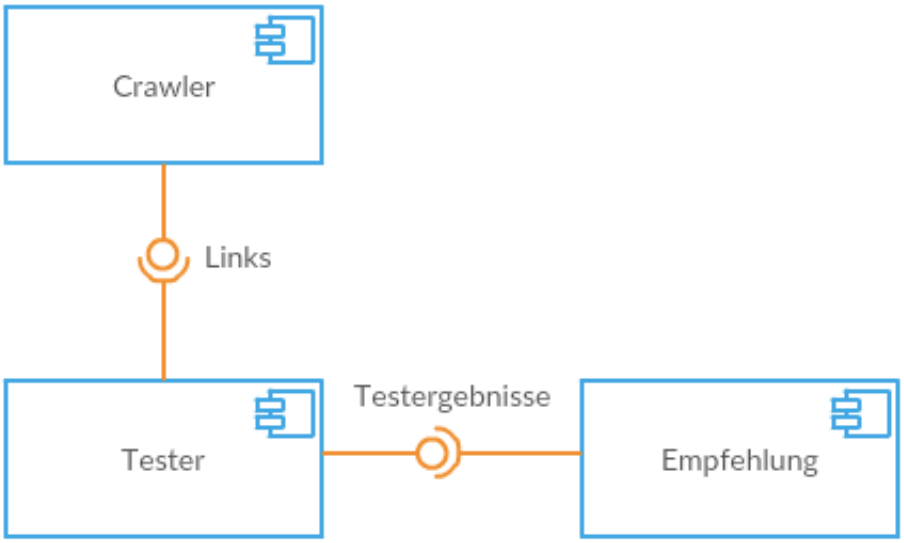


Abbildung 8: Darstellung der Zusammenarbeit der einzelnen Komponenten

Die genauen Funktionalitäten der einzelnen Komponenten werden im folgenden Kapitel erläutert und dargestellt

4.1 Hauptkomponenten

Im folgenden Abschnitt dieser Arbeit wird auf die wichtigsten Komponenten des Systems eingegangen. Dabei wird untersucht, wie diese Funktionalitäten entworfen werden könnten und mit welchen Frameworks dies möglich wäre. Beim Entwurf dieser Komponenten ist darauf zu achten, dass diese automatisiert zusammenarbeiten müssen können, die dafür nötigen Schnittstellen werden ebenfalls in diesem Teil des Kapitels erläutert.

4.1.1 Crawler-Komponente

Die Crawler Komponente soll die funktionale Anforderung des Öffnens einer Webseite und des Suchens nach neuen Links abdecken und die dafür nötigen Funktionalitäten bieten. Die Komponente des zu entwickelnden Systems muss dementsprechend eine Webseite öffnen können und anschließend weitere Seiten aufrufen, um sich so Schritt für Schritt durch eine Webseite zu arbeiten. Mithilfe dieser Komponente sollen die funktionalen Anforderungen für das Durchsuchen einer Webseite nach neuen Links und die Anpassbarkeit von Einstellungen bezüglich des Crawlers realisiert werden.

Der Crawler muss Webseiten aufrufen, diese aber nicht grafisch darstellen können. Zusätzlich muss der genutzte Browser eine API anbieten, diesen zu steuern und für das spätere Testen den Netzwerkverkehr aufzeichnen können. Hierfür bietet sich ein kopfloser Browser wie PhantomJS an. Kopflose Browser haben den Vorteil der versteckten Arbeit im Hintergrund. Bei normalen Browsern muss ein Fenster geöffnet werden, während bei PhantomJS alle Aktionen im Hintergrund, unsichtbar für den Anwendbar geschehen. Dies erfüllt bereits eine Teilanforderung des Durchsuchens von Webseiten, nämlich die versteckte Nutzung im Hintergrund. Damit können alle Funktionen, die für die funktionale Anforderung des Durchsuchens einer Webseite nach Links notwendig sind, umgesetzt werden. Diese Funktionen sind zum Beispiel Browserinteraktionen, wie das Öffnen einer Seite, das Anklicken eines Elements oder die Suche nach einzelnen Elementen auf der Webseite. PhantomJS beinhaltet damit in Zusammenarbeit mit Selenium auch fast alle benötigten Funktionen, um die Testkomponente im nächsten Punkt umsetzen können. Nur die Aufzeichnung des Netzwerkverkehrs muss über ein weiteres Framework namens BrowserMobProxy durchgeführt werden. Andere Browser als PhantomJS können diese Funktionalitäten ebenfalls bereitstellen, bieten dabei jedoch nicht die gleiche native Unterstützung als kopfloser Browser an. PhantomJS wurde als reiner kopfloser Browser entwickelt und auch in diese Richtung hin optimiert. Aus diesem Grund bietet sich PhantomJS für die Nutzung in dieser Komponente an. Durch die Zusammenarbeit mit Selenium kann der Browser jedoch jederzeit ausgetauscht werden, falls dies notwendig sein sollte.

Der Crawler muss mit Hilfe des kopflosen Browsers eine Liste an gefundenen Links auf einer Webseite erstellen und diese anschließend zusammenfassen. Dieses Zusammenfassen bezieht sich auf das Entfernen von weiteren URL-Parametern, sodass nur noch der Hostname und der Pfad der Seite enthalten sind. Die URL-Parameter sind für die Nutzung der Links irrelevant, da es nur um das Öffnen dieser Seite geht und die Parameter dafür nicht notwendig sind. Werden diese Parameter nicht entfernt kann es passieren, dass die gleichen Links, mit anderen URL-Parametern, mehrmals in der Liste enthalten sind. Diese Menge an Links muss für jede Domain erstellt werden und darf nur einzigartige URLs enthalten, da verschiedene Seiten trotzdem die gleichen Links enthalten können und die Liste ansonsten unzählige Duplikate derselben URL enthalten wird.

Für die Linkerkennung gibt es zwei mögliche Wege. Der erste Weg sucht direkt in den Webelementen nach neuen Links. Diese Links sind in den HTML-Hyperlink Tags der Elemente als Attribut hinterlegt. Um diese Attribute auslesen zu können, muss die Komponente alle Webelemente dieses Typs finden und anschließend zur Liste der gefundenen Links hinzufügen können. Dafür kann mit PhantomJS und Selenium ein Selektor mit Filter genutzt werden.

Die zweite Möglichkeit der Linkerkennung kann über systematisches Austesten und aussenden von Klick-Ereignissen an einzelne Webelemente realisiert werden. Bei dieser Vorgehensweise werden verschiedene Arten von Webelementen, wie zum Beispiel Bilder oder Buttons, angeklickt. Nach dem Klick muss für einen kurzen Moment gewartet werden und anschließend getestet werden, ob zu einer anderen Seite navigiert wurde. Ist dies der Fall, sollte die aktuelle Seite zur Liste der Links hinzugefügt werden.

Diese Liste an Links muss zusätzlich mit einem Parameter für eine Sortierung versehen werden, da der Crawler ansonsten immer wieder die gleichen Seiten besuchen könnte und dabei vereinzelte Seiten nie besucht werden. Dieser Parameter könnte zum Beispiel ein Zeitstempel sein, wann diese Seite zuletzt besucht wurde oder ein simpler Zähler, der bei jedem Besuch inkrementiert wird.

Eine beispielhafte Darstellung dieses Crawlers wird in Abbildung 9 dargestellt. Durch einen Browser, wie PhantomJS, wird eine erste Startseite aus dem Internet geladen. Der Crawler verarbeitet anschließend die geladene Seite und sucht dort nach weiteren Links. Hat er Links gefunden, bereinigt er diese und speichert diese gemeinsam mit dem Zähler der Besuche (Priorität) in einem Speicher ab. Diese Links werden anschließend in Abhängigkeit von der jeweiligen Priorität vom Planer ausgelesen. Dieser Planer gibt die Links anschließend wieder weiter an den Browser und der Kreislauf beginnt erneut.

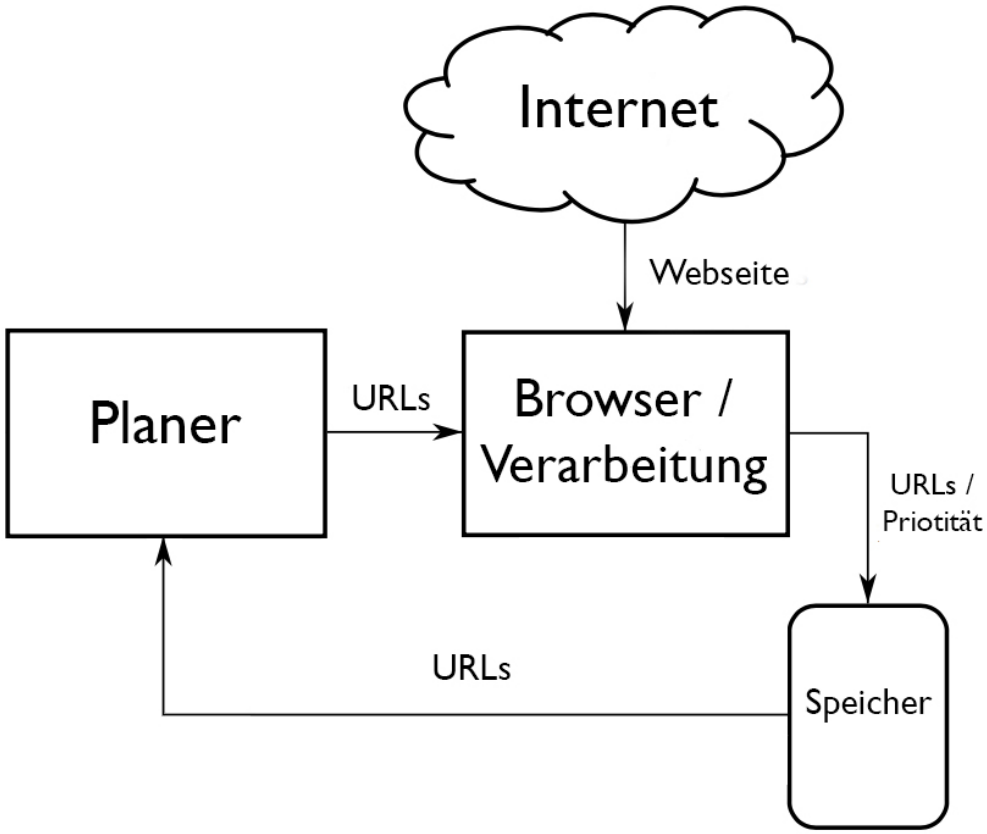


Abbildung 9: Crawler Entwurf

Diese Liste an gefundenen Links ist für die Darstellung der späteren Ergebnisse ebenfalls relevant und muss kontinuierlich aktualisiert und gesichert werden. Für die Speicherung dieser Liste bietet sich eine zweidimensionale Liste an, da nur der Link und die Priorität wichtig sind. Die gespeicherte Liste muss anschließend automatisiert von der Test-Komponente abrufbar sein um eine vollständige Automatisierung des Prozesses zu erreichen. Aus diesem Grund muss der Crawler den Tester darüber benachrichtigen, dass die Liste befüllt wurde, damit dieser mit seiner Arbeit beginnen kann. Diese Schnittstelle könnte über einen Getter des Crawlers realisiert werden. Dafür muss jedoch jedem Tester bei Beginn ein Crawler übergeben werden.

4.1.2 Test-Komponente

Ein wichtiger Bestandteil des Systems ist die Testkomponente zur Ausführung von Testfällen. Die Test-Komponente setzt hierbei direkt an die Arbeit des Crawlers an. Hat der Crawler erste Links gesammelt, kann die Test-Komponente direkt daran anknüpfen und startet mit der Menge an Links seine Arbeit. Hierfür müssen die

jeweiligen Links erneut aufgerufen werden, um dort anschließend Tracking-Aufrufe auszulösen. Bei jeder der zu betrachtenden Interaktionen muss der Netzwerkverkehr aufgezeichnet werden, um daraus entstandenes Tracking zu erfassen. Anschließend müssen die Parameter der gefundenen Trackingaufrufe mit den validen Daten abgeglichen werden. Valide Daten können hierbei Standardwerte sein, wie z.B. Eigenschaften des genutzten Browsers oder Gerätes, oder aber Werte aus Spezifikationen von Google Analytics. Seitenaufrufe besitzen beispielsweise für den Parameter für die Art der Interaktion (Hit-Typ) im Request immer den Wert „pageview“, andere Ereignisse wie Klicks oder Eingaben haben immer den Wert „event“. Sollte dies abweichen, wäre dieser Tracking-Aufruf bereits inkorrekt. Für die weitere Konzeption dieser Komponente wird davon ausgegangen, dass für das Webtracking Google Analytics genutzt wird.

Hierbei ist es sinnvoll, eine Mustererkennung zu implementieren um Tracking Funktionen anhand der gesendeten Parameter zu erkennen. Jeder Tracking Aufruf für ein Ereignis auf der Seite enthält bei Google Analytics immer eine Ereigniskategorie, Ereignisaktion und ein Ereignislabel. Auf Basis dieser drei Ereignisparameter sollte erkennbar werden, welche Tracking Funktionalität in dem Aufruf vorhanden ist.

Beispiel:
Ein Klick auf einen Button mit dem Bezeichner „Nachricht senden“ könnte folgenden Tracking-Aufruf auslösen:

Die URL des Tracking-Request lautet in diesem Fall: https://www.google-analytics.com/r/collect?t=event&ec=Button&ea=Nachricht%20senden%20&el=

Liest man daraus alle URL-Parameter aus erhält man folgende Liste:

- t (Hit-Typ): „event“
- ec (Ereigniskategorie): „Button“
- ea (Ereignisaktion): „Nachricht“
- el (Ereignislabel): nicht befüllt

In diesem Beispiel kann von der Ereigniskategorie und der Aktion darauf geschlossen werden, dass es sich hier um Tracking für Button-Klicks handelt. Dementsprechend sollte auf diese Daten ein Test für die Tracking-Funktion des Button-Tracking ausgeführt werden. Für die Testausführung müssen mehrere Standardtestfälle erstellt werde, welche die standardmäßigen Tracking-Funktionen von Google-Analytics wie Seitenaufrufe, Buttonklicks und Formularabsendungen abdecken.

Für die Ausführung der Testfälle muss vor Beginn des Testlaufes der Tester-Komponente übergeben werden wonach der Tester suchen soll. Soll dieser nach Button-Tracking suchen, führt dieser Klicks auf alle Buttons der Seite aus und prüft

anschließend, ob die daraus resultierenden Tracking-Aufrufe für diese Interaktion korrekt sind. Wird nun einer dieser Testfälle ausgeführt vergleicht der Tester die gefundenen Parameterwerte aus dem ausgelösten Tracking mit denen vorheriger Testläufe und mit Werten aus der Spezifikation von Google-Analytics.

Die Ergebnisse dieser Tests müssen anschließend dokumentiert und gespeichert werden, da diese zum einen in den folgenden Testläufen erneut verwendet werden und für die Benachrichtigung des Nutzers und deren Einblicke in den Verlauf relevant sind. Dabei kann auf die bereits durch den Crawler erstellte Liste zurückgegriffen werden.

4.1.3 Empfehlungskomponente

Ist eine Tracking Funktion komplett nicht implementiert und wird diese also beim Durchlauf der Seiten nicht entdeckt, ist es sinnvoll den Anwender darüber zu benachrichtigen.

Das Zählen von verpassten Tracking Möglichkeiten ist hierfür eine aussagekräftige Zahl. Eine verpasste Möglichkeit ist folgendermaßen definiert: Anzahl gefundener Tracking Aufrufe dieser Funktion geteilt durch die Anzahl an gefundenen Auslösepunkten. Eine Auflistung dieser einzelnen Funktionen mit den verpassten Möglichkeiten bietet anschließend eine Einsicht darüber, welche weiteren Tracking-Funktionen empfehlenswert sein könnten.

Ein Beispiel hierfür wäre ein nicht aktiviertes Button-Tracking von Google Analytics auf einer Webseite. Die Empfehlungskomponente würde dort wie folgt vorgehen:

- Die Komponente findet beim Durchsuchen einer Webseite 300 Buttons
- Auf alle 300 Buttons wird nacheinander ein Klick ausgeführt und getestet, ob dabei Button-Tracking ausgelöst wurde
- Keiner dieser Buttons löst einen dazu passenden Tracking-Aufruf aus
- Die Komponente liefert das Ergebnis: 300 verpasste Möglichkeiten für das Sammeln von Nutzerdaten über Button-Tracking

Für die Umsetzung dieser Komponente müssen pro möglicher Tracking-Funktion alle Elemente gezählt, werden welche möglicherweise einen Tracking-Aufruf dafür auslösen könnten.

5 Prototyp

Im letzten Kapitel wurden anhand von den definierten Anforderungen einzelne Komponenten definiert. Auf der Basis dieser Komponenten soll nun ein Prototyp entwickelt werden.

5.1 Technische Umsetzung

Der erste Teil dieses Kapitels befasst sich mit der technischen Umsetzung des Prototyps. Hier wird beschrieben wie die einzelnen Komponenten umgesetzt wurden und welche Probleme hierbei aufgetreten sind. Zusätzlich dazu wird genauer definiert, welche Einschränkungen der Prototyp besitzt und welche Komponenten nicht komplett umgesetzt wurden.

5.1.1 Einschränkungen

Der Prototyp fokussiert sich nur auf die Hauptfunktionen des zu entwickelnden Werkzeugs. Die Zusatzkomponente, das Empfehlungssystems, wird nicht entwickelt, da dies keine der grundlegenden Funktionalitäten des Werkzeugs ist.

Das Augenmerk wird hierbei auf den Crawler, den Tester und die automatisierte Zusammenarbeit dieser gelegt. Dies sind die Hauptfunktionen des Werkzeuges, welche während der Entwicklung des Prototyps auf jeden Fall realisiert werden sollen. Die Testkomponente wird für diesen Prototyp nur das Tracking von Seitenaufrufen und einfachen Button-Klick Ereignissen auf Korrektheit überprüfen. Das bedeutet, dass für den Prototyp nur Testfälle für diese Tracking-Ereignisse realisiert werden und der Tester damit auch nur diese beiden Tracking-Funktionen erkennt.

5.1.2 Implementierungsentscheidung

Erste Überlegungen zur Umsetzung der Interaktionen mit einer Webseite brachten die Erkenntnis, dass die Nutzung von reinem JavaScript und jQuery Code möglich ist. Eine Umsetzung mit Hilfe dieser Skriptsprachen würde die Nutzung des Selenium Frameworks überflüssig machen und dabei nur die wesentlichen Funktionen beinhalten, wobei diese genau an die benötigte Verhaltensweise angepasst werden können. Ein weiterer Grund für die Implementierung eigener Methoden ist der große

Funktionsumfang von Selenium. Selenium ist ein großes Framework mit einer Vielzahl an Funktionen, wobei die meisten dieser Funktionen für den Prototyp nicht relevant sind. Die Verwendung dieser selbstimplementierten Methode führt jedoch dazu, dass die Verbindung vom PhantomJS Browser mit dem BrowserMobProxy erschwert wird. Der PhantomJS Treiber für Selenium funktioniert besonders gut in Zusammenarbeit mit dem BrowserMobProxy, da dieser ursprünglich von den Machern von Selenium mitentwickelt wurde und noch immer auf Technologien von Selenium beruht.

Außerdem bietet die Entwicklung dieser Methoden einen hohen Mehraufwand im Vergleich zur Nutzung eines bestehenden Frameworks, da diese von Grund auf neuimplementiert werden müssten. Aus diesen Gründen wird auf die Implementierung von eigene JavaScript und jQuery Methoden zur Browserinteraktion verzichtet und stattdessen das Framework Selenium genutzt, da dieses die meisten benötigten Funktionen bereits bietet

5.1.3 Crawler

Der Crawler wird mit Hilfe von PhantomJS und Selenium umgesetzt. Selenium verwendet standardmäßig einen auf dem Rechner installierten Browser wie Firefox oder Chrome und öffnet bei der Verwendung dieses WebDrivers jeweils ein neues Browser-Fenster. Durch die Nutzung des PhantomJSDrivers für Selenium kann stattdessen der kopflose Browser PhantomJS genutzt werden. Durch diesen Treiber werden alle Interaktionen mit dem Browser im Hintergrund ausgeführt und sind für den Anwender nicht sichtbar. Als Erstes müssen die Eigenschaften des PhantomJS Browsers definiert werden, hierfür wird eine Größe des Fensters festgelegt.

Der Crawler führt konstant eine zweidimensionale Liste an gefundenen Seiten. Diese Liste darf nur einzigartige Links enthalten. Somit werden doppelte Einträge vermieden. Diese Liste enthält zusätzlich zu den Links noch einen Integer Wert. Vor jedem neuen Seitenaufruf durchsucht der Crawler die Liste nach dem kleinsten Wert, dies garantiert, dass alle Links gleichmäßig besucht und nicht immer dieselben Seiten aufgerufen werden. Für diese Suche läuft eine einfache Schleife über alle Elemente der Liste und vergleicht die Integer Werte. Am Ende der Suche liefert die dafür implementierte Methode das Element mit dem kleinsten Zähler zurück.

Die Angaben der Fenstergröße sowie die Startseite und ein Limit an zu durchsuchenden Seiten wird dem Crawler im Konstruktor übergeben. Anschließend kann der Crawler Thread gestartet werden. Der Crawler muss jetzt die im Konstruktor übergebene Startseite über Selenium in PhantomJS öffnen. Dies wird dadurch realisiert, dass eine Methode des PhantomJS Treibers von Selenium zum Aufrufen einer Seite ausgeführt wird. Diese Methode kommuniziert anschließend mit

PhantomJS und öffnet dort den übergebenen Link. Diese erste Seite wird zu der Liste der besuchten Seiten hinzugefügt und erhält einen Wert von 0, da diese Seite noch nicht abgearbeitet wurde.

Dort beginnt die Suche nach weiteren Links, hierfür wurden im vorherigen Kapitel zwei mögliche Wege vorgestellt. Die erste Möglichkeit ist die Suche nach allen Elementen des HTML-Tags des Typs Hyperlink, welche ein HREF-Attribut besitzen. Dieses Attribut beinhaltet einen Link zu einer anderen Seite, wenn dieser Hyperlink-Tags ein klickbares Element beinhaltet. Ein a-Tag könnte beispielsweise wie folgt aussehen:

<a href="http://example.com?user=1" Link zu Example.com

Diese Art der Suche kann mit Selenium umgesetzt werden, da der PhantomJS Treiber für Selenium eine Methode zum Finden aller Elemente nach einem bestimmten Filter anbietet. Dieser Filter beschränkt die Suche nach Elementen mit dem Tag Namen a für Hyperlink, und liefert als Ergebnis alle a-Tags als eine Liste von Web Elementen zurück. Der Methodenaufruf zum Suchen nach Hyperlink-Tags sieht wie folgt aus:

List<WebElement> links = driver.findElements(By.tagName("a"));

Der gefundene Link muss anschließend bereinigt werden, das heißt, dass alle weiteren URL-Parameter entfernt werden müssen. Im obigen Beispiel ist dies der Teil „?user=1“ des HREF-Attributs. Werden diese Parameter nicht entfernt können identische Links mehrmals in die Liste der gefundenen Links aufgenommen werden. Das Bereinigen der Links geschieht also nur um garantieren zu können, dass die Liste wirklich nur einzigartige Links enthält. Dazu werden aus dem HREF-Attribut alle Werte beginnend mit einem Fragezeichen oder eine Raute entfernt. Als Resultat enthalten alle gefundenen Links nur noch den Hostnamen und den Pfad.

Diese Links können anschließend weiterverwendet werden und werden der Liste an Links mit einem Wert von 0, markiert als noch nicht besucht, hinzugefügt. Die Startseite muss anschließend als besucht markiert werden, hierfür wird der Wert in der Liste der Links für diesen Eintrag inkrementiert. Nach diesem ersten Durchlauf an einer Seite holt der Crawler neue Links aus der Liste beginnend mit dem kleinsten Wert. Anschließend startet der obige Prozess erneut von Beginn an, bis das im Konstruktor angegebene Limit erreicht ist.

Die zweite Möglichkeit probiert, verschiedene Elemente anzuklicken und prüft, ob sich die aktuelle Seite verändert hat und damit Browser zu einer neuen Seite weitergeleitet wird. Hierfür müssen bei jedem Aufruf der Seite alle Elemente eines bestimmten Typs, welche möglichweise ein Link beinhalten können, geholt und getestet werden. Mithilfe von Selenium können, wie beim vorherigen Weg, alle

Elemente eines Typs gefunden und in einem Array aus Webelementen zwischengespeichert werden. Der Crawler sendet anschließend über Selenium ein Klick Ereignis an das erste Element und wartet. Während dieses Wartens wird der Thread des Crawlers für kurze Zeit pausiert. Nach dieser Wartezeit wird mithilfe des Treibers überprüft, ob sich der Link der aktuellen Seite von der vorherigen Seite unterscheidet. Ist ein Unterschied vorhanden, wird die aktuelle Seite der Liste der gefundenen Links hinzugefügt und ebenfalls mit einer 0, für noch nicht bearbeitet, markiert. Nach der erfolgreichen Navigation zu einer neuen Seite muss zunächst zur Ursprungsseite zurückgekehrt, werden um dort nach weiteren Links suchen zu können. Die Weiterverwendung des bereits erstellten Arrays an Webelementen dieser Seite löst beim Senden eines Klick-Ereignis an eines der Elemente eine Exception aus. Diese Exception beruht darauf, dass sich die Webelemente während der Verwendung des Treibers verändert haben könnten und dementsprechend nicht mehr verwendbar sind. Bedingt durch diesen Umstand muss bei jedem neuen Aufruf der Ursprungsseite dieses Array an Webelementen komplett neu erstellt werden. Erst danach kann der Crawler das folgende Element anklicken und mit einem neuen Durchlauf beginnen. Sind alle Webelemente abgearbeitet holt der Crawler neue Links aus der Liste beginnend mit dem kleinsten Wert. Anschließend startet dieser Prozess erneut, bis das im Konstruktor angegebene Limit erreicht ist.

5.1.4 Auswahl Crawler Methode

Der Crawler bietet zwei mögliche Wege Links zu finden. Beide Wege bieten Vor- und Nachteile. Für den Prototyp muss entschieden werden, welche dieser beiden Methoden die bessere Wahl ist. Diese Entscheidung wird im folgenden Textteil erläutert.

Methode 1: Suchen nach Hyperlink Webelementen

Die mithilfe des Selenium Treibers gefunden Elemente können schnell verarbeitet werden, da nach dem erstmaligen Seitenaufruf alle Elemente bereits vorhanden sind. Hier ist kein erneutes Laden der Seite notwendig und alle Elemente können direkt abgerufen werden. Andere Varianten des Verlinkens zu neuen Seiten, wie zum Beispiel in den Kontext der Seite eingebundene JavaScript Codes, werden hierbei jedoch nicht entdeckt. Dies ist damit zu begründen, dass diese Methode nur die einzelnen Elemente betrachtet und nicht den kompletten Kontext.

Methode 2: Austesten von Klick-Ereignissen

Das systematische Aussenden von Klick-Ereignissen an Webelemente deckt fast alle Links der Seite ab. Hierbei werden auch Weiterleitungen entdeckt, welche über JavaScript in den Kontext einprogrammiert wurden. Dieses Ausprobieren erfordert aber deutlich mehr Zeit, da hierbei nach jedem gesendeten Klickereignis gewartet werden muss. Zusätzlich dazu muss bei erfolgreichem Aufrufen die Startseite und die Liste der dortigen Webelemente komplett neugeladen werden.

Fazit

Aufgrund der sich bei der Entwicklung gezeigten schlechteren Performanz und der daraus resultierenden Durchlaufdauer für einen einzigen Suchlauf auf einer Seite der zweiten Methode, wird für den Prototyp die erste Methode verwendet. Die zweite Methode ist jedoch trotzdem ein guter Ansatz, um alle Links entdecken zu können und kann möglicherweise für eine bessere Performanz optimiert werden.

5.1.5 Request

Um einen Request objektiv besser abbilden zu können muss für diesen eine eigene Klasse erstellt werden. Mit Hilfe dieser Klasse kann der Tester einfacher auf die einzelnen Parameter des Tracking-Request eingehen und ein Request als Objekt auslesen. Der Konstruktor dieser Klasse nimmt einen Request-Link als String an und liest anschließend alle benötigten Parameter aus. Der übergebene String wird im ersten Schritt genutzt, um damit ein URL Objekt zu erstellen. Anschließend wird aus diesem Request eine LinkedHashMap erstellt, welche alle URL Parameter mit deren entsprechenden Werten beinhaltet.

Die Methode splitQuery wird vom Konstruktor automatisch mit dem erstellten URL Objekt aufgerufen und dekodiert die Request Query in ihre einzelnen Parameter. Die Query des übergegebenen URL Objekts wird dort nach jedem &-Symbol zerteilt und die resultierenden einzelnen Stücke der Query in einem String Array gespeichert. In dem Array befinden sich nun Strings wie zum Beispiel: el=Beispiel.

Jeder String dieses Arrays wird anschließend in einer Schleife abgearbeitet, wobei Parametername und Wert anhand des Gleichheitszeichens getrennt werden. Die getrennten Werte müssen vor der Speicherung in der Map noch mithilfe des der URLDecoders dekodiert werden, da Request Queries in der Regel kodiert gesendet werden. Der Parametername wird danach als Schlüssel und der Parameterwert als Wert in der Map abgelegt.

```
private Map<String, String> splitQuery(URL url) throws UnsupportedEncodingException {
        query_pairs = new LinkedHashMap<String, String>();
        String query = url.getQuery();
        String[] pairs = query.split("&");

        for (String pair : pairs) {
                int idx = pair.indexOf("=");
                query_pairs.put(URLDecoder.decode(pair.substring(0, idx), "UTF-8"),
                URLDecoder.decode(pair.substring(idx + 1), "UTF-8"));
        }

        return query_pairs;
}
```

Das Auslesen der einzelnen Parameter wird mithilfe dieser Map durchgeführt, um später flexibel weitere Parameter auslesen zu können. Das Auslesen ist durch diesen Weg nur ein simpler Get-Aufruf an die Map mit dem gesuchten Parameternamen als Übergabewert. Beispiel: query_pairs.get("el") liefert den Wert aus dem obigen Beispiel „Beispiel“.

5.1.6 Traffic Analyzer

Der Traffic Analyzer wird mit Selenium und BrowserMobProxy umgesetzt und kümmert sich um das Aufrufen von gefundenen Links, Interaktionen mit der Webseite und dem Aufzeichnen von ausgelösten Request.

Dem Analyzer wird im Konstruktor der ihm zugewiesene Crawler und die für den Browser gewünschte Auflösung übergeben. Der im Analyzer genutzte Selenium Browser Treiber wird beim Erstellen im Konstruktor so konfiguriert, dass dieser allen Netzwerkverkehr über den BrowserMobProxy Server sendet. Dieser ProxyServer wird ebenfalls im Konstruktor erstellt und anschließend gestartet. Der Proxy wird danach zu einem SeleniumProxy umgewandelt, sodass dieser dem Selenium Browser Treiber als Eigenschaft eingestellt werden kann und auch von diesem Browser genutzt wird. Der Proxy Server läuft permanent im Hintergrund bis der Traffic Analyzer beendet wird.

Da der gesamte Netzwerkverkehr des Selenium Browsers über einen Proxy geleitet wird, kann dieser direkt aufgezeichnet und zwischengespeichert werden. Da für das Testen von Webtracking Aufrufen nur Request Daten wichtig sind, wird der Proxy so konfiguriert, dass dieser nur Request-Informationen aufzeichnet und alle weiteren Netzwerkdaten ignoriert.

Der Analyzer wird zeitgleich mit dem Crawler als Thread gestartet und wartet auf die Benachrichtigung des Crawlers, dass er Seiten gefunden hat. Sobald der Crawler seine Linkliste gefüllt hat, weckt dieser den Analyzer Thread auf, welcher sich anschließend die aktuelle Linkliste holt. Anschließend beginnt der Analyzer seine Arbeit, und startet über den Proxy die Aufzeichnung aller Request-Aufrufe in einem HTTP Archive und öffnet den ersten Link über den Selenium Firefox Treiber.

Die Nutzung des Firefox Treibers ist bedingt durch Inkompatibilitäten zwischen dem Selenium PhantomJS Treiber und BrowserMobProxy notwendig. Dadurch kann dieser Prozess aktuell nicht versteckt im Hintergrund ausgeführt werden und wird sichtbar in einem Fenster durchgeführt.

Nach dem Laden der Seite wird die Aufzeichnung der Request-Informationen gestoppt und die erstellte HAR aus dem Proxy zwischengespeichert. Dieses HTTP Archive beinhaltet nun alle Request-Aufrufe die während des Seitenaufrufs gesendet wurden. Dieses HAR muss noch gefiltert werden, weil es zusätzlich zu den Tracking Aufrufen noch weitere Aufrufe beinhaltet. Aus diesem Grund werden alle Aufrufe, die nicht google-analytics und collect entfernt. Die Aufrufe, die jedoch an Google Analytics gesendet werden, werden zu einem Request Objekt. Hierfür wird der obigen Request Klasse der komplette Request-Link übergeben und für jeden gefundenen Tracking-Aufruf ein neues Request Objekt erstellt. Die Gesamtheit der gefundenen Tracking-Aufrufe wird anschließend an den Tester übergeben, welcher die Request-Informationen des Seitenaufrufes auf Korrektheit prüft. Anschließend wird die Liste von gefundenen Aufrufen zurückgesetzt und das Testen vom Button Tracking begonnen.

Für das Testen des Button Trackings auf einer Webseite müssen zuerst über den Selenium Treiber alle Elemente des Typs Button gefunden werden. Hat der Analyzer eine Liste aller gefundenen Buttons, beginnt dieser jeden Button anzuklicken. Dazu wird eine neue Aufzeichnung des Netzwerkverkehrs gestartet und über den Treiber Klick-Ereignisse an die Buttons gesendet. Der Thread wartet anschließend für einen kurzen Moment und stoppt die Aufzeichnung. Die dabei gesammelten Request-Aufrufe werden danach an den Tester übergeben, der diese anschließend überprüft. Anschließend muss die Ursprungsseite neu geladen werden, da es möglich ist, dass wir in dem kurzen Wartezeitraum zu einer neuen Webseite weitergeleitet wurden. Durch dieses erneute Laden der Seite muss die Liste an Buttons ebenfalls aktualisiert werden. Ansonsten wird eine Exception von Selenium geworfen, dass sich die Elemente in der Zwischenzeit verändert haben könnten. Abschließend werden die gefundenen Tracking-Aufrufe zurückgesetzt und der nächste Button aus der Liste angeklickt.

Wurden alle Buttons der Seite angeklickt, wird eine neue Seite aus der Ergebnisliste des Crawlers geöffnet und der obige Prozess beginnt erneut mit dem Seitenaufruf.

Ist die Liste komplett abgearbeitet, prüft der Analyzer, ob es eine neue, erweiterte Liste des Crawlers gibt und startet anschließend erneut diese Abarbeitung. Gibt es keine neue Liste, startet der Analyzer mit der existierenden Liste und arbeitet diese erneut von vorne ab.

5.1.7 Request Tester

Der Request Tester bekommt im Konstruktor eine Liste an Request Objekten und einen Identifikator, wobei dieses Request Objekte aufgezeichnet wurde übergeben. Der Identifikator besteht aus einem String. Bei diesem Prototyp ist dieser beschränkt auf „pageview“ und „button“. Der Identifikator entscheidet, welcher Test auf das jeweilige Request Objekt ausgespielt werden soll.
Der Pageview-Identifikator löst hierbei den Pageview Test aus. Dieser Test überprüft ob der Hit-Typ des Request „pageview“ entspricht, ob die gesendete Auflösung dem des Browsers gleicht und ob der richtige Pfad mitgesendet wurde. Befindet sich in der Liste der Request-Objekte mindestens ein Request, der diese Bedingungen erfüllt, so ist der Testfall erfolgreich und das Tracking damit funktionsfähig und die gesendeten Daten valide. Besteht kein Request diesen Test, wird auf der Konsole eine Fehlermeldung ausgegeben und eine Log-Datei erstellt. Diese Log-Datei beinhaltet einen Zeitstempel, den Link sowie die getestete Tracking Funktion.

Der Button-Identifikator löst den Button Test aus. Dieser Test überprüft, ob der Hit-Typ des Request „event“ entspricht, da es sich hierbei um ein konkretes Ereignis handelt. Zusätzlich wird überprüft, ob die Event Kategorie den Begriff „button“ beinhaltet. Auch hierbei gilt die selbe Regel wie oben: befindet sich in der Liste der Request-Objekte mindestens ein Request, der diese Bedingungen erfüllt, so ist der Testfall erfolgreich und das Tracking damit funktionsfähig und die gesendeten Daten valide. Besteht kein Request diesen Test, wird auf der Konsole eine Fehlermeldung ausgegeben und eine Log-Datei erstellt. Diese Log-Datei beinhaltet einen Zeitstempel, den Link sowie die getestete Tracking Funktion. Zusätzlich zu diesen Informationen werden ebenfalls die Event Kategorie, Aktion und das Label mitgespeichert.

5.2 Evaluation

Im folgenden Abschnitt wird die Realisierung des Prototyps der Anwendung auf ihre Funktionalitäten hin evaluiert. Dazu werden die tatsächlichen Anforderungen mit den in den Szenarien geschilderten verglichen.

5.2.1 Erfüllung der Anforderungen

Im dritten Kapitel wurden anhand von Beispielszenarien Anforderungen ermittelt. Hierbei ist zu beachten, dass nicht alle Anforderungen erfüllt werden konnten, da nur Grundfunktionalitäten für den Prototyp des Werkzeuges implementiert wurden. Im Folgenden werden funktionale und nicht-funktionale Anforderungen auf ihre Umsetzung überprüft.

Funktionale Anforderungen

Funktionale Anforderung	Status	Beschreibung
Durchsuchen von Webseite	Erfüllt	Der Prototyp kann automatisiert Seiten öffnen und nach neuen Links suchen.
Anpassen von Einstellungen	Nicht erfüllt	Der Prototyp bietet keine grafische Oberfläche. Alle Ergebnisse und Interaktionen werden über die Konsole ausgeführt.
Erkennung des Tracking	Erfüllt	Der Prototyp kann automatisiert Tracking-Aufrufe erkennen und dabei aufzeichnen.
Prüfung des Tracking	Erfüllt	Der Prototyp kann die Parameter der erkannten Tracking-Aufrufe überprüfen.
Ausgabe Ergebnisse	Erfüllt	Der Prototyp gibt die Ergebnisse der Testläufe über die Konsole aus und speichert diese zusätzlich dazu in einer Log-Datei.
Ausgabe Statistik	Nicht erfüllt	Der Prototyp kann keine Statistik der letzten Durchläufe ausgeben und darstellen.
Benachrichtigung bei Ausfall	Teilweise erfüllt	Der Prototyp gibt die Ergebnisse der Testläufe über die Konsole aus, sendet jedoch keine Benachrichtigung per E-Mail oder anderem Medium.
Empfehlungen anzeigen	Nicht erfüllt	Der Prototyp besitzt kein Empfehlungssystem, da dies für den Prototyp nicht notwendig war.

Nicht-Funktionale Anforderungen

Anhand der erfüllten funktionalen Anforderung konnte die Erstellung eines Prototyps für ein Werkzeug zum automatisierten Testen von Webtracking Systemen nachgewiesen werden. Nun ist noch die Qualität dieser Anlage durch die nicht-funktionalen Anforderungen zu evaluieren.

Die Erweiterbarkeit des Systems ist durch die getroffenen Designentscheidungen gewährleistet. Die Zuverlässigkeit des Systems lässt sich anhand der getroffenen Entscheidungen erkennen. Hierzu zählt das Designs des Systems, mit der strikten Trennung der Komponenten und deren Kommunikation untereinander sowie der modulare Aufbau im inneren der Komponenten.

Der Sicherheit und Nutzbarkeit konnte nicht realisiert werden, da der Prototyp keine Oberfläche bietet.

Die Wartbarkeit des Systems ist dadurch gewährleistet, dass der komplette Code mit JavaDoc auskommentiert wurde und jederzeit neue Testparameter in den Test aufgenommen werden können.

5.2.2 Funktionsweise des Prototyps

In diesem Abschnitt sollen die Bedienung und die verwendeten Abläufe des realisierten Prototyps vorgestellt werden. Dabei werden insbesondere die Hauptfunktionalitäten wie das Starten der automatischen Suche nach neuen Links und Tracking-Aufrufen erläutert.

Da der Prototyp nur über die Konsole funktioniert, muss dem Prototypen nur im Aufruf der Startfunktion eine Startseite, eine Auflösung für den Browser und ein Limit an zu durchsuchenden Seiten übergeben werden. Danach beginnt das Werkzeug seine Arbeit und benötigt keine weiteren Eingaben des Nutzers. Der Nutzer erhält alle benötigten Informationen über die Konsole und kann diese, wenn nötig, ebenfalls aus den Log-Dateien auslesen.

5.2.3 Erkenntnisse

Die Evaluierung hat gezeigt, dass die Erstellung eines Werkzeugs zur automatisierten Testdurchführung bei Webtracking-Systemen machbar ist. Dies wird aus den umgesetzten Anforderungen und der Arbeitsweise des Prototyps deutlich. Zusätzlich dazu hat sich gezeigt, dass die im Design beschriebenen Komponenten, die für den Grundbetrieb des Prototyp benötigt werden, umgesetzt werden konnten.

Ebenfalls zeigte die Evaluation, dass die bisherige Realisierung nur einen kleinen Teil der Funktionalitäten bereitstellt. Es fehlen noch die Empfehlungsmöglichkeiten für Webtracking Funktionen sowie eine nutzerfreundliche Oberfläche. Erst durch die Erfüllung dieser Funktionalitäten erhält das System für den Anwender einen Mehrwert.

Die Problemfelder, die sich beim Erstellen des Prototyps ergaben, konnten nicht zufriedenstellend gelöst werden. Hier sind insbesondere die Inkompatibilität zwischen dem Selenium PhantomJS Treiber und dem BrowserMobProxy zu nennen.
Auf Grundlage dieser Problemfelder können weitere Entwicklungen stattfinden, in denen diese Probleme gelöst und weitere entworfene Funktionalitäten ausgebaut werden können.

6 Ergebnisse

Dieses Kapitel bildet den Schluss dieser Arbeit. Hier wird eine Zusammenfassung der gesamten Arbeit und ein Ausblick ausformuliert.

6.1 Zusammenfassung

Im Rahmen dieser Arbeit sollte die Realisierung eines automatisierten Testwerkzeuges für Webtracking Systeme untersucht werden. Basis hierfür bildete das Framework Selenium in Zusammenarbeit mit dem Proxy von BrowserMob und PhantomJS. Die Wahl fiel auf diese Werkzeuge, da diese im Gegensatz zu anderen einige Vorteile besaßen. Dazu gehören unter anderem die große Flexibilität von Selenium bezüglich der Verwendung von verschiedensten Treibern für Webbrowser und die guten Programmierschnittstellen. Ein weiterer Vorteil dieser Werkzeuge ist die von den Entwicklern beabsichtigte Zusammenarbeit zwischen Selenium und BrowserMobProxy.

Im Verlauf der Arbeit stellte sich jedoch heraus, dass Selenium nicht alle Funktionalitäten zur Verfügung stellt, die für die Realisierung benötigt wurden. Aus diesem Grund wurden externe Bibliotheken eingebunden und einige Funktionen zusätzlich implementiert.

Am Anfang dieser Arbeit wurden zwei Szenarien analysiert, durch die verschiedene Anforderungen an das System und daraus resultierende benötigte Komponenten abgeleitet werden konnten.

6.2 Ausblick

Aus den Erfahrungen, die in der Evaluation gesammelt wurden, ergaben sich weitere Erkenntnisse, die der Weiterentwicklung des Systems dienen können. Diese werden im folgenden Ausblick aufgeführt.

Die Arbeit hat gezeigt, dass ein Werkzeug zum automatisierten Testen von Webtracking Systemen hohes Potential besitzt. Insbesondere die Entwicklung eines Empfehlungssystems ist essentiell für die zukünftigen Möglichkeiten dieses Werkzeugs. Für die Zukunft wäre es denkbar, dass diese Funktionalität die zweite Hauptfunktion neben dem Testen des Tracking, werden könnte. Mit Hilfe dieser

Funktion können Tracking Konzepte erweitert und optimiert werden, während gleichzeitig das bereits vorhandene Tracking getestet und permanent überwacht wird.

Aktuell bietet der Prototyp nur eine Erkennung und Prüfung von Tracking-Funktionen von Google-Analytics und unterstützt keine weiteren Anbieter. Eine Erweiterung auf neue Tracking-Tools ist jedoch möglich. Hierfür muss die Erkennung erweitert und auf das neue Tracking-Tool optimiert werden. Zusätzlich dazu müssen auch weitere Testfälle erstellt werden, welche die Parameter der gefundenen Tracking-Aufrufe überprüfen. Dies ist natürlich mit Aufwand verbunden. Zum Glück funktionieren jedoch alle Tracking-Werkzeuge grob nach demselben Prinzip.

Ein weiterer essentieller Punkt stellen zukünftige Bedienkonzepte dar. Eine saubere und leicht übersichtliche Oberfläche bietet dem Anwender einen wesentlichen Mehrwert. Die Integration in das bereits bestehende Werkzeug Columbo bringt dies bereits mit sich. Durch die Erweiterung der dortigen Oberfläche mit Analysefunktionen und Statistikoperationen könnten aus den gesammelten Daten weitere Informationen herausgearbeitet werden, wie zum Beispiel die Zuverlässigkeit des Tracking-Systems.

Inwiefern aus diesen zukünftigen Ideen nützliche Funktionen entstehen, die dem Anwender einen Mehrwert bringen, bleibt abzuwarten. Des Weiteren wird sich zeigen, wie die Aspekte der Sicherheit, Wartbarkeit und Stabilität des Systems mit in die weitere Entwicklung einfließen. Mit Fragestellungen dieser Art werden sich mit Sicherheit in der Zukunft weitere wissenschaftliche Arbeiten beschäftigen.

Abbildungsverzeichnis

Literatur- und Quellenverzeichnis

Google, Benutzerdefinierte Tags. *2016*. Available at: https://support.google.com/tagmanager/answer/6107167?hl=de [Accessed August 25, 2016].

Google, 2016a. Google Analytics Features. Available at: https://www.google.com/intl/de_ALL/analytics/premium/features.html [Accessed June 14, 2016].

Google, 2016b. Tracking Code Overview. Available at: https://developers.google.com/analytics/resources/concepts/gaConceptsTrackingOverview [Accessed July 17, 2016].

Hidayat, A., 2016. PhantomJS. Available at: http://phantomjs.org [Accessed August 25, 2016].

InfoTrust, 2016. Tag Inspector. Available at: https://www.taginspector.com [Accessed July 12, 2016].

Jacken, C., 2010. Black-White-Grey-Box. *04.01.2010*. Available at: http://swtest-blog.de/black-white-grey-box/comment-page-1 [Accessed August 23, 2016].

Johnson, S., 2008. Google Analytics: UTM Link Tagging Explained. Available at: http://www.intownwebdesign.com/google/google-analytics-utm-link-tagging-explained.html [Accessed July 16, 2016].

Lightbody, P., 2016. BrowserMobProxy. Available at: https://bmp.lightbody.net [Accessed August 25, 2016].

Markus Schneider, Matthias Enzmann, M.S., 2014a. Web-TrackIng-Report 2014. Available at: https://www.sit.fraunhofer.de/fileadmin/dokumente/studien_und_technical_reports/Web_Tracking_Report_2014.pdf [Accessed June 16, 2016].

Peaks&Pies, 2016a. Columbo Dokumentation. Available at: https://docs.columbo.io/de/scenarios [Accessed April 14, 2016].

Peaks&Pies, 2016b. Tag Testing Matrix.

Peterson, E.T., 2015. *Data Quality and the Digital World*

Schallaböck, J., 2014. Was ist und wie funktioniert Webtracking? *10.06.2014*. Available at: https://irights.info/artikel/was-ist-und-wie-funktioniert-webtracking/23386 [Accessed August 17, 2016].

Selenium, 2016. Selenium. Available at: http://www.seleniumhq.org [Accessed August 25, 2016].

To, L., 2014. How to harness the awesomeness of Google Tag Manager. Available at: https://3qdigital.com/google/harness-awesomeness-google-tag-manager/ [Accessed July 16, 2016].

W3C, 2012. HTTP-Archive. Available at: https://dvcs.w3.org/hg/webperf/raw-file/tip/specs/HAR/Overview.html [Accessed August 25, 2016].

W3Tech, 2016a. Usage of tag managers for websites. Available at: https://w3techs.com/technologies/overview/tag_manager/all [Accessed July 16, 2016].

W3Tech, 2016b. Usage of traffic analysis tools for websites. Available at: https://w3techs.com/technologies/overview/traffic_analysis/all [Accessed July 16, 2016].

Yeskow, A., 2015. An Ultimate Guide to Google Tag Manager. Available at: http://positionly.com/blog/inbound-marketing-manager [Accessed August 24, 2016].

Versicherung über Selbstständigkeit

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

Hamburg, den _____ _____