Diff Viewer im Open Source Projekt Scenarioo

Bachelorarbeit

Abteilung Informatik

Hochschule für Technik Rapperswil

Frühjahrssemester 2016

Autoren:	Pascal Forster, Manuel Scheuber
Betreuer:	Prof. Dr. Markus Stolze, Institut für Software, HSR
Projektpartner:	Zühlke Engineering AG
Experte:	Thomas Kälin, Fachleiter Mobile Computing, BBV
Gegenleser:	Prof. Dr. Andreas Rinkel, HSR

Erstelldatum	Montag, 29. Februar 2016
Änderungsdatum	Freitag, 17. Juni 2016
Druckdatum	Freitag, 17. Juni 2016
Autoren	Pascal Forster (pforster@hsr.ch) Manuel Scheuber (mscheube@hsr.ch)

Eigenständigkeitserklärung

Wir erklären hiermit,

- dass wir die vorliegende Arbeit selber und ohne fremde Hilfe durchgeführt haben, ausser derjenigen, welche explizit in der Aufgabenstellung erwähnt sind oder mit dem Betreuer schriftlich vereinbart wurde,
- dass wir sämtliche verwendeten Quellen erwähnt und gemäss gängigen wissenschaftlichen Zitierregeln korrekt angegeben haben.
- dass wir keine durch Copyright geschützten Materialien (z.B. Bilder) in dieser Arbeit in unerlaubter Weise genutzt haben.

Ort, Datum:

Rapperswil, 17. Juni 2016

Namen, Unterschriften:

Pascal Forster Manuel Scheuber

Abstract

Mit dem immer grösseren Funktionsumfang heutiger Applikationen spielt die Korrektheit und Aktualität der Softwaredokumentation eine zentrale Rolle. Das von Zühlke Engineering AG gegründete Open Source Projekt Scenarioo deckt dieses Bedürfnis ab und erlaubt es, eine Softwaredokumentation anhand von Screenshots aus UI Tests automatisiert zu erzeugen. Durch das tägliche generieren der Softwaredokumentation entstehen unterschiedliche Softwaredokumentationsstände. Ist man nun an den Änderungen zwischen zwei Dokumentationsständen interessiert, so muss man sich Schritt für Schritt durch die beiden Dokumentationsstände durcharbeiten und die einzelnen Unterschiede müssen von blossem Auge identifiziert werden. Dieser Prozess ist nicht nur fehleranfällig, sondern auch äusserst zeitintensiv.

Ziel dieser Arbeit ist es, die bestehende Webapplikation Scenarioo mit einer sogenannten Diff Viewer Funktionalität zu erweitern. Diese Erweiterung soll den Benutzer im täglichen Gebrauch von Scenarioo möglichst effizient unterstützen, strukturelle und visuelle Unterschiede zwischen zwei Dokumentationsständen einfach zu erkennen. Auf gute Usability ist besonderen Wert zu legen.

In einer ersten Phase wurden zusammen mit dem Industriepartner und bestehenden Scenarioo-Kunden die exakten Anforderungen an das Endprodukt ausgearbeitet. Daraus resultierend konnte ein entsprechendes Benutzungs- und Architekturkonzept erstellt werden. Als die wichtigsten Neuerungen umgesetzt waren, wurde die Diff Viewer Erweiterung durch Testpersonen auf ihre Benutzerfreundlichkeit geprüft. Anhand der dadurch gewonnenen Erkenntnisse konnten wir das Benutzererlebnis noch weiter optimieren.

Das Resultat ist die produktiv verwendbare Diff Viewer Erweiterung zu Scenarioo, im Sinne eines Minimum Viable Product. Mit dem Diff Viewer kann der Benutzer beliebige Dokumentationsstände miteinander vergleichen und sieht auf jeder Dokumentationsebene signifikante Änderungen benutzerfreundlich dargestellt.

Management Summary

Ausgangslage

Scenarioo ist ein von Zühlke Engineering gegründetes Open Source Projekt, welches einen innovativen Ansatz zur umfassenden und anschaulichen Softwaredokumentation anhand von Screenshots verfolgt. Die Daten für die Dokumentation werden automatisiert aus User Interface Tests generiert. Die so generierte Dokumentation wird allen Projektbeteiligten in einer Weboberfläche zur Verfügung gestellt.

Durch das tägliche generieren der Softwaredokumentation entstehen unterschiedliche Softwaredokumentationsstände. Ist man nun an den Änderungen zwischen zwei Dokumentationsständen interessiert, so muss man sich Schritt für Schritt durch die beiden Dokumentationen durcharbeiten und die einzelnen Unterschiede müssen von blossem Auge identifiziert werden. Dieser Prozess ist nicht nur fehleranfällig, sondern auch äusserst zeitintensiv.

Ziel

Ziel dieser Arbeit war es, die bestehende Webapplikation Scenarioo mit einer sogenannten Diff Viewer Funktionalität zu erweitern. Diese Erweiterung soll dem Benutzer helfen, strukturelle und visuelle Unterschiede zwischen zwei Dokumentationsständen möglichst einfach zu erkennen. Auf gute Usability ist besonderen Wert zu legen.

Vorgehen, Technologien

In einem ersten Schritt wurden die User Stories in enger Zusammenarbeit mit unserem Industriepartner erarbeitet. Damit wir ein besseres Gespür für die unterschiedlichen Anforderungen erhielten, haben wir Benutzerinterviews mit Personen aus den verschiedensten Bereichen durchgeführt.

Danach wurde ein Benutzungs- und Architekturkonzept erstellt, welches die Integration der neuen Funktionalität in die bestehende Webapplikation detailliert beschreibt. Als auch dieser Meilenstein erreicht war, konnte mit der Implementation begonnen werden. Die auf Java und AngularJS basierende Webapplikation wurde gemäss den vorgängig priorisierten User Stories nach und nach ausgebaut.

Damit wir die nicht-funktionale Anforderung der Bedienbarkeit des Diff Viewers überprüfen konnten, wurde ein Usability Test mit verschiedenen Testpersonen durchgeführt. Aus den daraus gewonnenen Erkenntnissen konnten noch letzte Optimierungen am Endprodukt vorgenommen werden.

Ergebnis

Das Resultat ist eine produktiv verwendbare Erweiterung zu Scenarioo, im Sinne eines Minimum Viable Product. Mit dem Diff Viewer kann der Benutzer beliebig konfigurierbare Vergleiche zwischen zwei Dokumentationsständen durchführen. Die Unterschiede zwischen zwei Ständen werden einmalig berechnet und im entsprechenden Scenarioo-Format abgelegt. Sobald der Vergleichsvorgang abgeschlossen ist, werden dem Benutzer auf jeder Dokumentationsebene die signifikanten Änderungen benutzerfreundlich dargestellt.

In einem Progress-Bar ähnlichem Diff-Icon ist schnell ersichtlich, wie fest und was genau sich an einem einzelnen Element geändert hat. Die Diff Viewer Funktionalität ist auch in der Lage neue und gelöschte Elemente in der Weboberfläche entsprechend darzustellen.

Der Benutzer hat ausserdem die Möglichkeit, die Screenshots von zwei unterschiedlichen Dokumentationsständen miteinander zu vergleichen. Die Änderungen auf dem Screenshot werden farblich hervorgehoben und können bei Bedarf ausgeblendet werden. Wie dies aussehen kann, ist im nachfolgenden Screenshot ersichtlich.

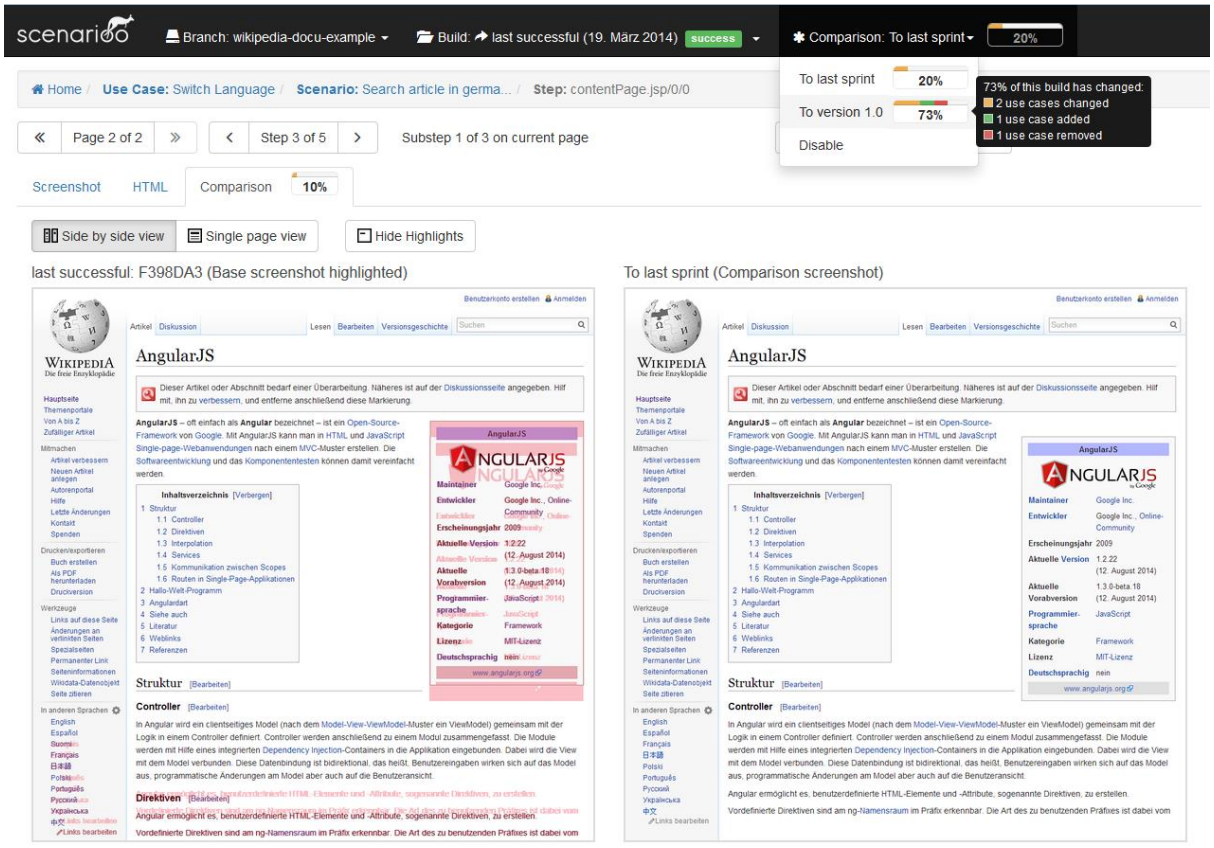


Abbildung 1: Beispiel einer Vergleichsansicht

Ausblick

Schon bei den ersten Benutzerinterviews wurde klar, dass wir nicht alle Anforderungen an den Diff Viewer im Rahmen dieser Arbeit umsetzen können. Deshalb haben wir uns auf die am höchsten priorisierten User Stories konzentriert. Dennoch möchten wir hier einen zusammenfassenden Überblick über die weiteren Möglichkeiten vom Scenarioo Diff Viewer geben.

1.1.1 Interaktiver Screenshot-Vergleich auf Client

Zusätzlich zum aktuellen Screenshot-Vergleich auf dem Server könnte man ein JavaScript Framework wie zum Beispiel Resemble.js auf dem Client einsetzen. Dieses erlaubt dem Anwender noch weitere dynamische Konfigurationsmöglichkeiten beim Screenshot-Vergleich. So könnte der Benutzer unter anderem konfigurieren, dass eine geänderte Farbe nicht als Unterschied dargestellt wird.

1.1.2 Oberfläche zur Konfiguration der Vergleiche

In der jetzigen Lösung erfolgt die Konfiguration der verschiedenen Vergleiche der Dokumentationsstände direkt in einer Konfigurationsdatei. In Scenarioo ist es üblich, dass die Konfiguration über die Weboberfläche erfolgt. Es wäre deshalb angenehmer, wenn die Vergleichs-Konfiguration über eine Benutzeroberfläche möglich wäre.

1.1.3 Weitere Werte vergleichen

Scenarioo kennt noch mehr Elemente als jene, welche momentan mit dem Diff Viewer verglichen werden. So könnte es beispielsweise auch spannend sein, zusätzlich zu Use Cases, Szenarios, Steps und Screenshots, auch Pages oder Metadaten miteinander zu vergleichen.

Aufgabenstellung

Aufgabenstellung Bachelorarbeit Abteilung I, FS 2016
Pascal Forster, Manuel Scheuber

Diff Viewer im Open Source Projekt „Scenarioo“

1. Betreuer & Praxispartner

Betreuer dieser Arbeit ist
Prof. Dr. Markus Stolze mstolze@hsr.ch

Co-Referent für diese Arbeit ist
Andreas Rinkel

Externer Experte für diese Arbeit ist
Thomas Kälin

Anwendungspartner dieser Arbeit ist
Zühlke Engineering AG
Wiesenstrasse 10a
8952 Schlieren (Zürich)

2. Ausgangslage

Siehe separates Blatt (Aufgabenstellung Zühlke)

3. Ziele der Arbeit

Siehe separates Blatt (Aufgabenstellung Zühlke)

4. Dokumentation

Über diese Arbeit ist eine Dokumentation gemäss den Richtlinien der Abteilung Informatik zu verfassen. Die Dokumentation ist vollständig auf CD/DVD in einem Exemplar abzugeben (Exemplar für das Sekretariat Informatik) Sowie ein Download-Link für Prof. Stolze und weitere Exemplare nach Absprache mit dem Co-Referentenund dem Experten.

Zudem ist eine kurze Projektresultatdokumentation im öffentlichen Wiki von Prof. M. Stolze zu erstellen.

5. Weitere Regeln und Termine

Im Weiteren gelten die allgemeinen Regeln zu Bachelor und Studienarbeiten „Abläufe und Regelungen Studien- und Bachelorarbeiten im Studiengang Informatik“ (HSR Intranet) https://www.hsr.ch/Ablaeufe-und-Regelungen-Studie.7479.0.html)

Der Terminplan ist hier ersichtlich (HSR Intranet) https://www.hsr.ch/Termine-Bachelor-und-Studiena.5142.0.html

6. Rechte

Die resultierende Software und Dokumentation wird als Komponente des Open-Source Projekts Scenarioo mit der entsprechenden Lizenz veröffentlicht. Es gelten die sich daraus ergebenden Rechte. Die Arbeit (ohne geheime Anhänge) wird von der HSR im E-Prints Respository der HSR (eprints.hsr.ch) elektronisch veröffentlich. Hierbei ist von den Studenten darauf besonders darauf zu achten, dass in Screenshots und an anderen Stellen keine Kundendaten oder Namen sichtbar werden. Zudem sollten Interviews sorgfältig anonymisiert werden. Die zu publizierende Version sollte dem Praxispartner zur Überprüfung der Einhaltung dieser Regeln mit der Abgabe der Arbeit vorgelegt werden. Allfällig notwendige Anpassungen an der zu publizierenden Version werden Praxispartner binnen 4 Wochen gemeldet und danach von den Studenten entsprechend innerhalb einer Woche umgesetzt. Titel, Abstract und Praxispartner der Arbeit dürfen von der HSR und Studierenden schon während der Arbeit kommuniziert werden.

7. Beurteilung

Eine erfolgreiche Bachelorarbeit zählt 12 ECTS-Punkte pro Studierenden. Für 1 ECTS Punkt ist eine Arbeitsleistung von ca. 25 bis 30 Stunden budgetiert. Entsprechend sollten ca. 350h Arbeit für die Bachelorarbeit aufgewendet werden. Dies entspricht ungefähr 25h pro Woche (auf 14 Wochen) und damit ca. 3 Tage Arbeit pro Woche pro Student.

Für die Beurteilung ist der HSR-Betreuer verantwortlich.

Die Bewertung der Arbeit erfolgt entsprechend der verteilten Kriterienliste.

Die definitive Aufgabenstellung wurde am 4.4.2016 beschlossen.

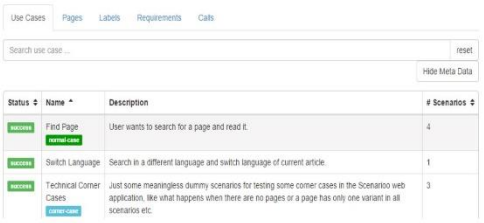
Rapperswil, 4.4.2016

Prof. Dr. Markus Stolze
Institut für Software
Hochschule für Technik Rapperswil

scenarioo

zühlke
empowering ideas

Diff Viewer im Open Source Projekt „Scenarioo“



Ansprechpartner

Zühlke Engineering AG
Wiesenstrasse 10a
8952 Schlieren (Zürich)

Ausgangslage

Das von uns gegründete Open Source Projekt *Scenarioo* verfolgt einen innovativen Ansatz, umfassende und anschauliche Software-Dokumentation anhand von Screenshots aus UI Tests automatisiert zu erzeugen. Die *Scenarioo* Webapplikation stellt die getesteten Benutzer-Szenarien allen Beteiligten in einem Software-Projekt immer aktuell und benutzerfreundlich anhand der Screenshots zur Verfügung.

Ziel der Arbeit

Die Webapplikation Scenarioo wird mit neuer Funktionalität erweitert um die Unterschiede zweier Snapshots der Systemdokumentation zu visualisieren. Signifikante Änderungen werden auf jeder Detailebene der Dokumentation ansprechend dargestellt. Es sollen sowohl strukturelle Änderungen der Use Cases und ihrer Szenarien wie auch visuelle Änderungen der Screenshots berücksichtigt werden. Dazu sollen möglichst effiziente Algorithmen zum Struktur- und Bildvergleich gefunden und eingesetzt werden (3rd Party Libraries oder falls notwendig eigene Implementation). Zentral ist dabei, dass der Benutzer die neue Funktionalität bei der täglichen Arbeit möglichst einfach und effizient nutzen kann. Auf gute Usability ist besonderen Wert zu legen.

Zu Beginn dieser Arbeit soll in Absprache mit dem Zühlke Team ein entsprechendes Benutzungs- und Architekturkonzept erstellt werden, welches danach iterativ umgesetzt und verbessert wird. Das Hauptziel der Arbeit ist die Erstellung einer produktiv verwendbaren Erweiterung zu Scenarioo, im Sinne eines „Minimum Viable Product“.

Randbedingungen

Webapplikation in AngularJS / JavaScript / HTML5 mit Java REST-Server
Open Source Entwicklung unter github.org

Bemerkungen:

Das Open Source Produkt *Scenarioo* wird bei uns bereits in einigen grossen Kundenprojekten eingesetzt. Eine Mitarbeit im Open Source Projekt Team ermöglicht Dir einen interessanten Einblick in unsere Projektarbeit und Zusammenarbeit mit erfahrenen Zühlke Mitarbeitern. Weitere Informationen zu Scenarioo: www.scenarioo.org

Beilagen

Scenarioo Flyer

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

I. Technischer Bericht

1 Einführung

Das dokumentieren von Software ist oftmals ein teures und mühsames Unterfangen. Wenn dann einmal alles initial dokumentiert ist, stellt sich noch die Frage, wer die Dokumentation aktualisiert, so dass sie auch in ein, zwei Jahren noch von Wert ist.

Das von Zühlke Engineering AG gegründete Open Source Projekt Scenarioo verfolgt einen innovativen Ansatz, umfassende und anschauliche Software-Dokumentation anhand von Screenshots aus User Interface Tests automatisiert zu erzeugen. Die so generierte Dokumentation der getesteten Szenarien wird allen Projektbeteiligten in einer Weboberfläche aktuell und mittels Screenshots zur Verfügung gestellt.

1.1 Problemstellung

Bei Scenarioo wird mit jedem Software Build die Dokumentation der Szenarien aktualisiert. Diese unterschiedlichen Dokumentationsstände werden archiviert und können jederzeit angesehen werden.

Möchte man nun zwei unterschiedliche Stände miteinander vergleichen, so muss man aktuell beide Builds in einem separaten Fenster öffnen und die Unterschiede manuell vergleichen. Dies ist sehr zeitaufwändig und auch nicht ganz trivial, da das menschliche Auge schnell eine Änderung übersieht.

1.2 Vision

Mit der „Diff Viewer“ Erweiterung im Projekt Scenarioo soll dem Benutzer der Vergleich von zwei Dokumentationsständen leicht gemacht werden. Das neue, in die bestehende Lösung zu integrierende Feature, soll den Benutzer möglichst effizient beim Erkennen von signifikanten Änderungen zwischen zwei Ständen unterstützen.

1.3 Ziele

Die zu entwickelnde Erweiterung soll die Unterschiede zwischen zwei Dokumentationsständen in der bestehenden Webapplikation sichtbar machen. Es sollen visuelle Änderungen auf den Screenshots wie auch strukturelle Änderungen in den Use Cases und ihrer Szenarien auf den verschiedenen Oberflächen von Scenarioo sichtbar gemacht werden. Um dieses Ziel zu erreichen, sollen möglichst effiziente Algorithmen zum Struktur- und Bildvergleich gefunden werden.

Das Hauptziel dieser Arbeit ist die Erstellung einer produktiv verwendbaren Erweiterung zu Scenarioo im Sinne eines Minimum Viable Product. Da die neue Funktionalität von

den unterschiedlichsten Anwendern benutzt wird, gilt es, besonderen Wert auf die Usability zu legen.

1.4 Rahmenbedingungen

Aus der gegebenen Aufgabenstellung können folgende Rahmenbedingungen entnommen werden:

- Die Usability der Erweiterung muss geprüft werden.
- Entwicklerkonventionen müssen gemäss Scenarioo Wiki eingehalten werden. https://github.com/scenarioo/scenarioo/wiki/Coding-guidelines
- Die Erweiterung soll in das bestehende Open Source Projekt Scenarioo integriert werden. Folgende Technologien kommen dabei zum Einsatz:
  - AngularJS
  - JavaScript
  - HTML5
  - Java REST Server
- Die Erweiterung soll produktiv einsetzbar sein.
- Die Lizenz von neu eingesetzten Frameworks muss mit der Lizenz vom Open Source Projekt Scenarioo kompatibel sein.

1.5 Vorgehen

Zu Beginn des Projekts geht es darum, die Anforderungen zu klären und in entsprechende User Stories zu fassen. Damit wir ein deutlicheres Bild von den wirklichen Bedürfnissen der Scenarioo Anwender erhalten, werden wir Benutzerinterviews mit Entwicklern und Anwendern von Scenarioo durchführen.

Sind die Anforderungen fürs Erste geklärt, werden wir eine passende Technologie zur Umsetzung der Funktionalität evaluieren.

Damit wir während der Entwicklungsphase keine bösen Überraschungen erfahren, werden wir schon vorgängig mit der evaluierten Technologie einige Funktionalitäten prototypmässig im Scenarioo umsetzen. So lernen wir das ausgewählte Framework und das bestehende Scenarioo Projekt besser kennen.

Sobald die gewünschte Funktionalität implementiert und getestet ist, werden wir zusammen mit unserem Industriepartner unser Feature einem Usability Test unterziehen. Dadurch erhoffen wir uns, wichtige Erkenntnisse zur Benutzerfreundlichkeit zu gewinnen und diese noch vor Abgabe der Arbeit umsetzen zu können.

Zu guter Letzt soll unsere Weiterentwicklung in die neuste Version von Scenarioo eingebettet werden. Denn unser Ziel ist es, am Ende den bestehenden und zukünftigen Scenarioo Anwendern eine produktive Erweiterung zur Verfügung zu stellen.

2 Stand der Technik

In der heutigen Zeit bieten Webseiten einen immer grösseren Funktionsumfang. Bei Google Docs ist beispielsweise ein ganzes Office-Paket online verfügbar. Ein solches Wachstum im Funktionsumfang bedeutet natürlich auch eine grössere Fehleranfälligkeit. Zudem wird es zunehmend schwieriger, einen Überblick über die gesamte Funktionalität zu haben.

Der einzige Weg, die oben genannten Probleme zu bewältigen, liegt in der Automation. In diesem Kapitel wird beschrieben, welche Ideen und Technologien verbreitet sind, um diese Automatisation zu bewerkstelligen.

Es gibt unzählige Lösungsansätze und Kombinationen davon. Am besten lassen sich die Abhängigkeiten zwischen den Technologien und Prinzipien in einer Grafik darstellen. Darauf sieht man, dass die momentane Version von Scenarioo vor allem das Prinzip „living documentation“ behandelt. Mit der Diff Viewer Komponente werden zusätzlich Regressionstests eingeführt. Die einzelnen Elemente werden in den Unterkapiteln erklärt. Natürlich ist es nicht möglich, sämtliche vorhandenen Lösungen zu dokumentieren. Wir beschränken uns auf besonders verbreitete oder für diese Arbeit interessante Ansätze.

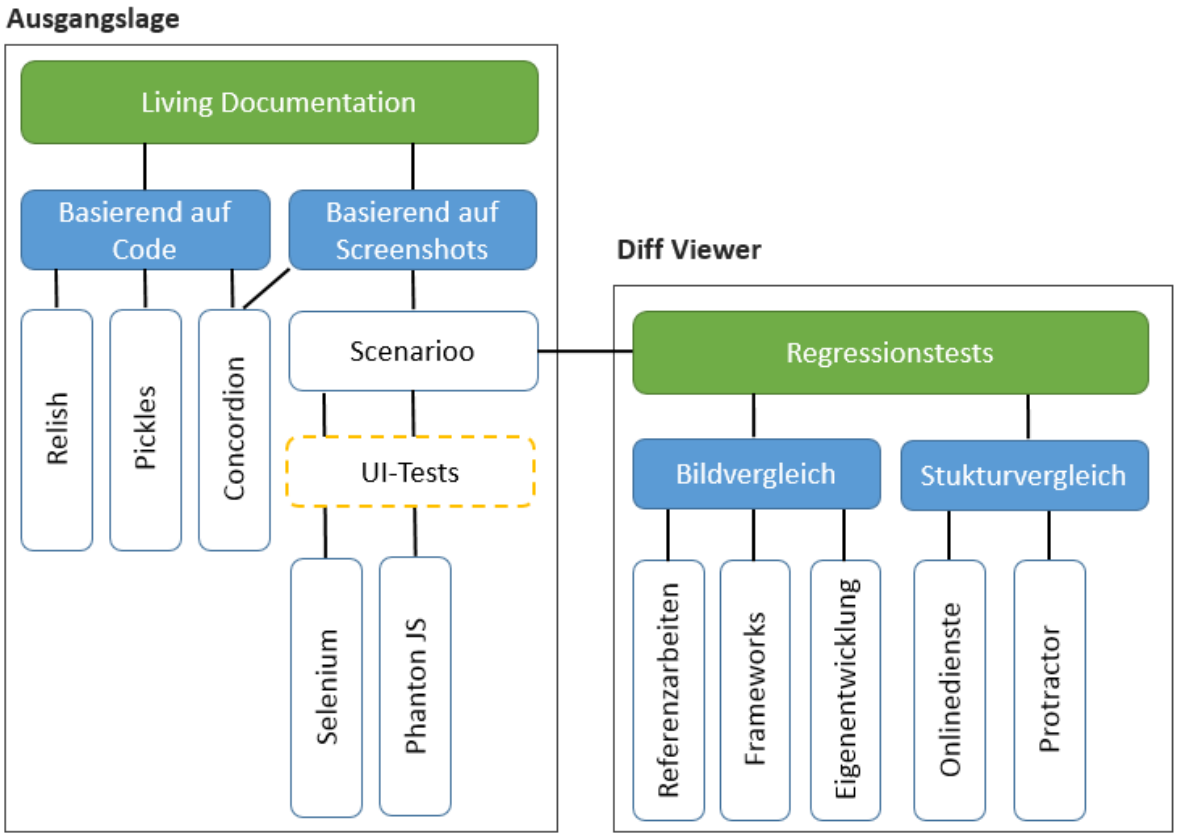


Abbildung 2: Übersicht Stand der Technik

2.1 Living Documentation

Living Documentation ist eine dynamische Methode einer Systemdokumentation, die aktuell, spezifisch und einfach zu verstehen ist. Die Dokumentation wird automatisch generiert und ist daher stets auf dem neuesten Stand. Die generierte Dokumentation ist in einer Form dargestellt, welche sämtliche Projektbeteiligte verstehen. Beispielsweise das Support Team, um bei Benutzerfragen auf die Dokumentation zurückgreifen zu können. (Rouse 2014)

2.1.1 Basierend auf Code

Produkte wie Relish (www.relishapp.com), Pickles (www.picklesdoc.com) oder unterstützen den Entwickler, eine solche Living Documentation zu erstellen. Sie analysieren den gesamten Code und erstellen daraus eine aktuelle Dokumentation. Der Entwickler hat die Möglichkeit, direkt im Code zusätzliche Kommentare mittels Markdown zu formatieren. Die generierte Dokumentation kann in unterschiedlichen Formaten exportiert werden. Beispielsweise als durchsuchbare Webseite oder als Word Dokument mit Inhaltsverzeichnis exportiert werden.

Concordion hat auf dem Bild eine Verbindung zu „Basierend auf Code“. Das liegt daran, dass Screenshots möglich sind, allerdings nie so Komfortabel wie bei den UI-Test Frameworks.

2.1.2 Basierend auf Screenshots

Auch Scenarioo verfolgt das Ziel einer Living Documentation. Im Gegensatz zu anderen Lösungen basiert die Scenarioo Dokumentation auf Screenshots der Webseite und wird nicht direkt aus dem Code generiert. Weitere Details zum Funktionsumfang von Scenarioo befinden sich im Kapitel Ausgangslage.

Es existieren unterschiedliche Frameworks, die mittels UI-Test automatisiert Screenshots erstellen. Zu den bekanntesten gehören beispielsweise Selenium (www.seleniumhq.org) oder PhantomJS (www.phantomjs.org). Es gibt einen grossen Unterschied zwischen diesen beiden Frameworks. PhantomJS funktioniert genau wie Chrome, ausser dass man das User Interface nicht sieht. Diese Technik nennt sich Headless Browser. CasperJS oder SlimerJS sind weitere Frameworks aus dieser Kategorie. Selenium hingegen verhält sich wie ein Benutzer und führt Mausklicks und Texteingaben auf der Webseite aus. Während der Test ausgeführt wird, kann man beobachten, wie die Webseite automatisiert bedient wird.

2.2 Regressionstest

Ein Regressionstest wird ausgeführt, um sicherzustellen, dass Änderungen aufgrund von Bug fixes oder Verbesserungen am Code die bestehende Funktionalität nicht beeinträchtigen. Durchgeführt wird ein solcher Test normalerweise von einem Test Engineer.

Üblicherweise wird bei einer neuen Version eines Softwareprodukts ein Set von Tests ausgeführt. Mittels Regressionstest vergleicht der Tester nun das Resultat mit dem Resultat einer Vorgängerversion. Gibt es im Vergleich keine Konflikte, ist der Regressionstest erfolgreich.

Die beste und sicherste Variante Regressionstests auszuführen ist per Automation. (Regression testing definition 2012)

2.2.1 Bildvergleich

2.2.1.1 IEEE Referenz zu Regressionstests bei Bildern

Es existieren unterschiedliche Ansätze, wie Regressionstest-Tools eingesetzt werden. Ein interessanter Ansatz wird in der Arbeit „Adaptive Random Testing for Image Comparison in Regression Web Testing“ (Selay 2014) erläutert.

Es wird ein Weg beschrieben, wie bei Screenshots-Vergleichen Fehler effizient erkannt werden, während insignifikante Änderungen vernachlässigt werden. Die verwendete Technik basiert auf den Charakteristiken von Fehlermustern aus Browser Layouts. Mittels adaptive random testing werden zufällig ausgewählte Punkte auf den Screenshots verglichen. Das Bild wird dazu in ein Gitternetz aufgeteilt. Der sogenannte FSCS ART Algorithmus spiegelt zwei bestimmte Punkte in Nachbarregionen. Dadurch werten alle Punkte bestimmt die verglichen werden. So kann mit hoher Performance bestimmt werden, ob die zwei Screenshots Unterschiede aufweisen. Gerade bei Layout Änderungen weist dieser Ansatz eine sehr hohe Fehlererkennungsquote auf.

Für unsere Arbeit können wir diesen Ansatz nicht übernehmen, da wir die genaue Abweichung der Screenshots in Prozent berechnen sollen. Vergleicht man nur einzelne Ausschnitte, ist das natürlich nicht möglich.

2.2.1.2 Vorgänger Arbeit an FHNW

Diese Arbeit wurde bereits an der Fachhochschule Nordwestschweiz durchgeführt. Im Rahmen dieser Arbeit wurde mit Java Funktionalität im Backend entwickelt um Scenarioo-Screenshots miteinander zu vergleichen. Durch den Verzicht auf Frameworks war dies sehr Aufwändig. Deswegen konnten die Strukturvergleiche nicht mehr behandelt werden.

Im Frontend wurden die JavaScript Framworks TwentyTwenty und ResembleJS für den Screenshot-Vergleich eingesetzt. Aus dieser Arbeit können wir lernen, dass es sehr zeitintensiv und komplex ist, einen Bildvergleich selber zu implementieren. (Visual Regression Testing in “Scenarioo” 2014)

2.2.1.3 BBC Wraith

Die Entwickler von BBC News haben mit Wraith ein Screenshot Comparison Tool implementiert, welches ein ähnliches Ziel wie der Diff Viewer verfolgt. Wraith kann auf zwei Arten benutzt werden. In der ersten Variante werden zwei URLs angegeben, die miteinander verglichen werden. Dabei werden automatisch Screenshots aufgenommen und miteinander verglichen. Die zweite Variante ist ein Vergleich über Zeit. Es kann eine URL angegeben werden von der regelmässig Screenshots gemacht werden. Diese Screenshots werden anschliessend mit dem ersten aufgenommenen Screenshot verglichen.

Da Wraith auf Ruby basiert können wir dies nicht in Scenarioo integrieren. Dennoch ist es gut zu wissen, dass ähnliche Lösungen existieren.

Wraith ist verfügbar unter: https://github.com/BBC-News/wraith

2.2.1.4 Frameworks oder Eigenentwicklungen für den Bildvergleich

Es gibt eine Vielzahl unterschiedlicher Frameworks für Bildvergleiche. Sucht man im Internet nach solchen Frameworks, trifft man hin und wieder sogar auf eine Eigenentwicklung. Wir werden uns in der Evaluation genauer mit der Auswahl eines solchen Frameworks beschäftigen.

2.2.2 Strukturvergleich

Für den Strukturvergleich an sich haben wir kein speziell dafür angefertigtes Framework gefunden. Es gibt allerdings Frameworks wie Protractor (www.protractortest.org) die für End-to-End Tests ausgelegt sind. Bei einem solchen Test wird gleichzeitig auch die Struktur der Webseite getestet.

Eine andere Möglichkeit wäre das Angebot von Onlinediensten zu nutzen. Seiten wie screenster (www.screenster.io) oder Fret (www.frontendtest.org) bieten die Möglichkeit Webseiten ohne eigenen Testserver zu testen. Dort können gewisse Abläufe konfiguriert werden, die wiederholt ausgeführt und getestet werden. Diese Dienste sind allerdings Kostenpflichtig wenn man bestimmte Features verwenden will. Die Preise bewegen sich zwischen CHF 0 (Basic) und CHF 265 (Professional).

3 Ausgangslage

Scenarioo ist ein Open Source Projekt, das schon vielerorts zum Einsatz kommt. Vorangetrieben wird Scenarioo von unserem Industriepartner, der Firma Zühlke. Scenarioo hat das Ziel die Dokumentation von Software mittels UI-Testing zu automatisieren.

Im Mai 2013 wurde der Quellcode erstmals veröffentlicht und seitdem kontinuierlich von ungefähr 14 Mitwirkenden weiterentwickelt. Dank vorhandener Schnittstellen für Java, C# und JavaScript und der Möglichkeit eigene APIs dafür zu entwickeln, lässt sich Scenarioo problemlos in ein bestehendes Softwareprojekt integrieren.

(Scenarioo Development Team 2014)

3.1 Nutzbare Funktionen

Automatisierte User Interface Tests sollen nicht nur Entwicklern und Testern zur Verfügung stehen. Scenarioo bringt diese Tests in eine Form, in der sie für sämtliche Projektbeteiligte verständlich und nützlich dargestellt werden. Eine Weboberfläche sorgt für die benutzerfreundliche Darstellung und bietet zudem folgende Funktionen:

- Use Case und ihre Szenarien darstellen
- Einzelne Interaktionsschritte und UI-Masken als Printscreen aufzeigen
- Kommunikation mit externen Systemen dokumentieren
- Abläufe mit beliebigen Metainformationen ergänzen
- Verschiedene Gebrauchsvarianten der gleichen Seite aufzeigen
- Status des aktuellen Builds darstellen

Unter der Adresse http://demo.scenarioo.org ist eine Live Demo verfügbar, um jederzeit die Funktionen vom neuesten Release zu testen.

3.2 Grundlegender Aufbau

Das folgende Übersichtsbild zeigt den grundlegenden Aufbau und Ablauf um eine automatisierte Dokumentation aus UI Tests mit Scenarioo zu erstellen und zu nutzen. Die darauf folgenden Abschnitte erklären die einzelnen Schritte und Teile der Dokumentation mit Scenarioo im Detail.

(Rolf Bruderer, Zühlke 2015)

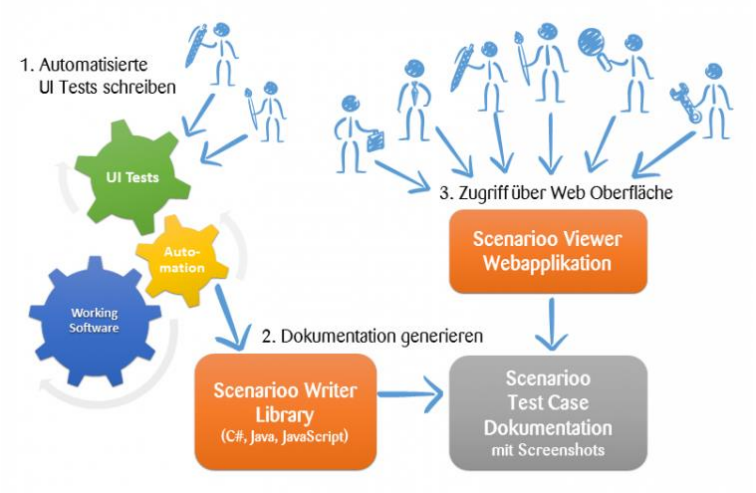


Abbildung 3: Grundlegender Scenarioo Aufbau (Rolf Bruderer, Zühlke 2015)

3.2.1 Schritt 1 – Schreiben und Strukturieren der UI Tests

Ein Entwickler schreibt für jeden Use Case, der in Scenarioo dargestellt wird, UI Tests mit dem Framework seiner Wahl. Ein weit verbreitetes Framework dafür ist beispielsweise Selenium. Pro möglichem Szenario wird dabei ein solcher UI Test geschrieben. Der Entwickler hat ausserdem die Möglichkeit, einzelne Szenarien mit benutzerdefinierten Labels zu versehen.

Hier ein Beispiel, was ein Use Case und seine Szenarien beinhalten können, wenn Scenarioo bei Wikipedia eingesetzt würde:

Use Case: - Finde eine Seite

Szenarien:
- Finde mehrere Seiten
- Finde keine Seite
- Finde Seite indirekt über eine ähnliche Seite
- Finde die Seite direkt

Der Entwickler hat darauf zu achten, nur die essentiellen und für die Dokumentation relevanten Benutzerszenarien auszuwählen und zu testen. Pro Use Case sind das in der Regel nicht mehr als 3 bis 15 Test-Szenarien.

3.2.2 Schritt 2 – Automatische Generierung der Dokumentation

Die im Schritt 1 definierten Tests werden normalerweise jede Nacht von einem automatisierten Buildsystem ausgeführt und automatisch validiert. Dabei wird bei jedem Interaktionsschritt automatisch ein Printscreen des User Interfaces erstellt und in die Dokumentation aufgenommen. Eine Hilfe ist dabei die sogenannte Scenarioo Writer Library, welche die Dokumentationsdateien im Scenarioo Format ablegt.

Dadurch, dass der Build jede Nacht ausgeführt wird, entstehen mehrere Dokumentationsstände. Der Benutzer kann im System auswählen, welchen dieser Stände er gerne betrachten möchte.

3.2.3 Schritt 3 – Zugriff auf die Dokumentation mit der Webapplikation

Das Herzstück von Scenarioo ist die Webapplikation, welche die generierten Dokumentationsdaten aus dem Build für alle Beteiligten anschaulich präsentiert und benutzerfreundlich navigierbar macht. Wie in einem Bilderbuch können die einzelnen Benutzerszenarien Schritt für Schritt betrachtet werden. Es kann durch die Use Cases und ihre Szenarien navigiert werden. Beliebige Zusatzinformationen können betrachtet werden. Alle Informationen können komfortabel durchsucht werden, damit der Betrachter möglichst schnell auf seine gewünschten Use Cases, Szenarien, Screens und weitere Informationen zugreifen kann.

Die folgenden zwei Bilder zeigen zwei essentielle Ansichten der Webapplikation

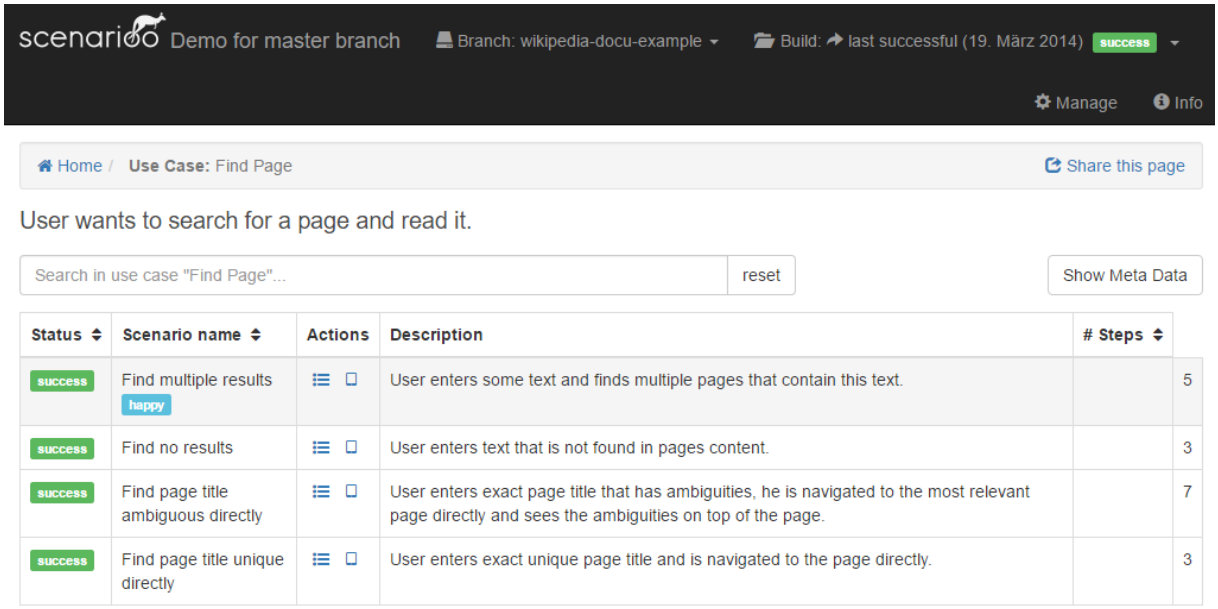


Abbildung 4: Szenarien im Wikipedia Beispiel (Demo Scenarioo 2016)

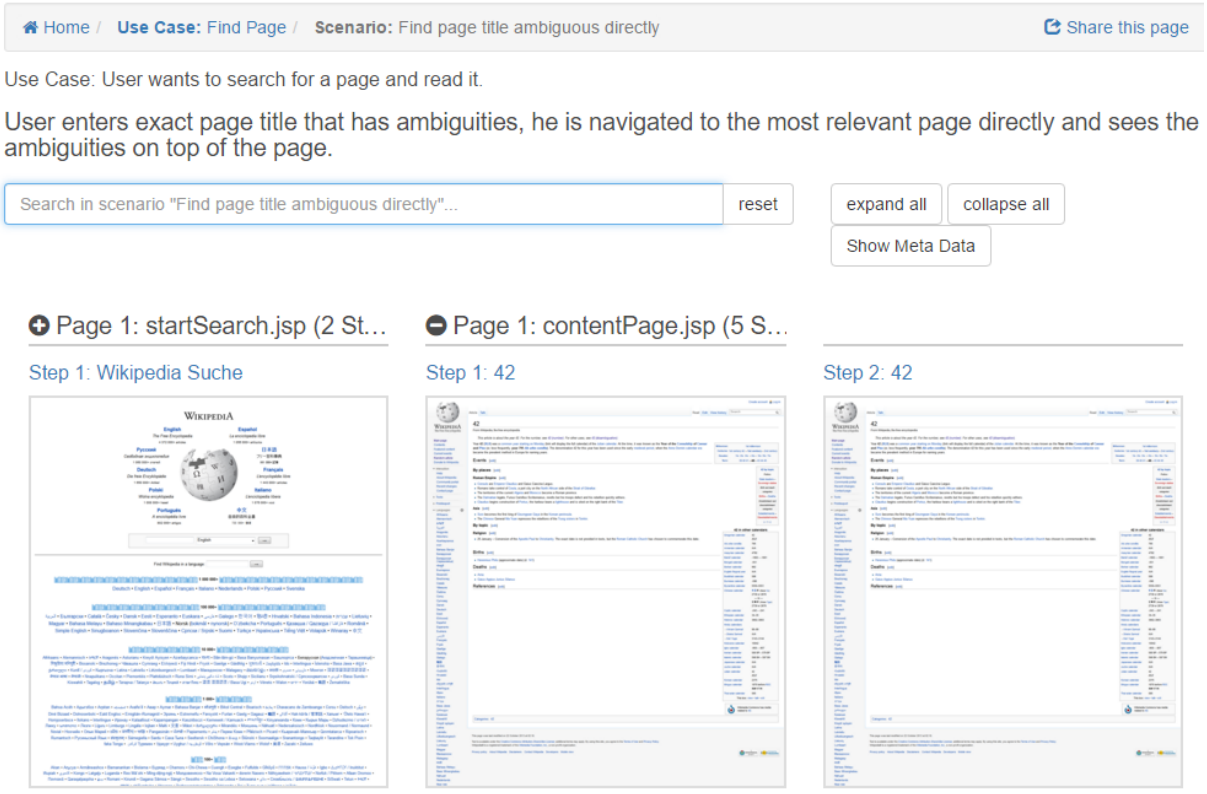


Abbildung 5: Übersicht Schritte (Demo Scenarioo 2016)

3.3 Nutzung von Scenarioo

Im Gespräch mit Scenarioo-Entwicklern und durch Benutzerinterviews konnten wir feststellen, für welche Benutzungsszenarien Scenarioo momentan verwendet wird. Dabei wurde für uns auch ersichtlich, dass es klare Unterschiede in der Verwendung zwischen den Benutzergruppen gibt.

Nachfolgend werden die wichtigsten Benutzergruppen und deren Anwendungsfälle aufgeführt.

3.3.1 Entwickler

Ein Entwickler programmiert die UI-Tests und ist somit verantwortlich für den Inhalt, der von Scenarioo dargestellt wird. Er kann dadurch zwei Fliegen mit einer Klappe schlagen. Er hat eine höhere Testabdeckung und gleichzeitig eine Dokumentation, die automatisch immer auf dem neuesten Stand ist.

Selber profitiert ein Entwickler auch von den Funktionen von Scenarioo. Er hat die Möglichkeit zu überprüfen, ob Use Cases durch seine Modifikationen auf einer bestimmten Seite beeinflusst wurden. Bei Änderungen am Webseitendesign kann er durch die betroffenen Seiten navigieren und sehen, ob die Darstellung überall stimmt. Mithilfe der Metadaten kann er kontrollieren ob bei den Seiteneffekten, wie zum Beispiel einem versendeten E-Mail, der Inhalt stimmt.

3.3.2 Tester

Dank Scenarioo ist ein Tester weniger auf den Entwickler angewiesen. Er kann jederzeit bei einem beliebigen Softwarestand nachschauen, wie sowohl der Ablauf als auch der Status der Tests war.

Auch die Metadaten bieten für den Tester einen erheblichen Mehrwert. Er kann darin erkennen, mit welchen Daten die Tests aufgerufen wurden und welche weiteren Systeme darin verwickelt sind.

Für den Tester, teils sogar auch für den Entwickler, ist es in Scenarioo einiges einfacher nachzuvollziehen, bei welchem Schritt ein Test fehlgeschlagen ist. Es wird visuell ersichtlich, wo genau der Fehler liegt. Man muss dafür nicht die langen und oft auch komplexen Log Dateien durchforsten.

3.3.3 Projektbeteiligte

Für die Benutzergruppe Projektbeteiligte haben wir mehrere Berufsgattungen zusammengefasst. Es kann sich dabei beispielsweise um Requirements Engineers, UX Designer, Supporter oder Projektleiter handeln. Diese benutzten Scenarioo auf eine ähnliche Art und Weise.

Benutzer aus diesen Berufsgattungen interessieren sich für die Szenarien in einem Use Case und deren genauen Ablauf. Dadurch können sie lernen zu verstehen, welche Optionen die Webanwendung ihren Endkunden bietet. Ausserdem ermöglicht es ihnen Support zu leisten, indem sie die vom Endkunden getätigten Schritte in Scenarioo nachvollziehen können. Es hilft zudem bei der Vorbereitung von Anpassungen oder dem Hinzufügen von neuen Funktionen in einem Projekt. Somit kann den Engineers genau gezeigt werden, was wo geändert werden soll.

4 Evaluation

4.1 Bildvergleich

4.1.1 Auf welchem Tier werden die Bilder verglichen

Bis ein Printscreen auf dem Frontend angezeigt wird, durchläuft er einen Prozess, der sich auf mehreren Servern (=Tiers) abspielt. Zuerst wird der Printscreen auf dem Buildserver generiert und abgespeichert. Im Anschluss darauf liest der Scenarioo Server das Verzeichnis mit dem Printscreens ein, um die Daten später über eine Restschnittstelle zur Verfügung stellen zu können. Zu guter Letzt liefert ein Node Server die aufbereiteten Daten an den Client, welcher die Webseite mithilfe des Angular Frameworks darstellt.

An welchem Ort in diesem Prozess können nun die Bilder verglichen werden? Die Antwort ist: überall. Deshalb listen wir hier die Vor- und Nachteile der jeweiligen Orte auf und begründen unsere Entscheidung.

4.1.1.1 Bildvergleich auf dem Build Server

Hier würde der Vergleich stattfinden, direkt nachdem die Testfälle ausgeführt würden und folglich die Screenshots abgespeichert wurden.

Vorteile:

- Das generieren der Printscreens und deren Vergleich erfolgt am gleichen Ort

Nachteile:

- Der UI-Test Entwickler muss sich selbst um den Bildvergleich kümmern
- Die Bildvergleichskomponente muss über die API vorgegeben werden, ansonsten muss auf dem Build Server eine Komponente evaluiert und implementiert werden
- Die Konfiguration der Diff Viewer Komponente kann nicht im Scenarioo Client erfolgen
  (Auf Umwegen würde es gehen, doch technisch gesehen macht es keinen Sinn)
- Der Buildvorgang dauert einiges länger
- Der Benutzer kann den Printsrceenvergleich nicht personalisieren

4.1.1.2 Bildvergleich auf dem Scenarioo Server

Hier würde der Vergleich stattfinden, nachdem der Scenarioo Server den abgespeicherten Build importiert hat.

Vorteile:

- Die Konfiguration der Diff Viewer Komponente kann im Scenarioo Client erfolgen.
- Weder der Client noch der Build Server müssen sich um den Bildvergleich kümmern.
- Eine einzige Komponente für den Bildvergleich reicht aus

Nachteile:

- Der Benutzer kann den Screenshot-Vergleich nicht personalisieren
- Der Scenarioo Server übernimmt zusätzliche Aufgaben

4.1.1.3 Bildvergleich auf dem Scenarioo Client

Der Bildvergleich auf dem Client ist kein Ersatz für den Bildvergleich auf dem Build- oder Scenarioo Server, da die Informationen vom Bildvergleich auch auf den höheren Ebenen, wie zum Beispiel auf Stufe Use Case, ersichtlich sein müssen. Es ist eher eine Ergänzung, um dem Benutzer mehr Konfigurationsmöglichkeiten betreffend Bildvergleich zu bieten.

Vorteile

- Der Benutzer kann den Bildvergleich personalisieren

Nachteile:

- Die Bilder müssen zweimal verglichen werden. Einmal auf dem Build- oder Scenarioo Server und einmal im Frontend
- Eine zusätzliche Bildvergleichskomponente für Angluar muss evaluiert und eingesetzt werden

4.1.1.4 Architekturentscheidung Bildvergleich Tier

Im Zusammenhang mit der Auswahl des Tiers, wo der Bildvergleich stattfinden soll und in Anbetracht der nicht-funktionalen Relevanz der Performance, haben wir uns für den Bildvergleich auf dem Scenarioo Server entschieden und nicht für den Bildvergleich auf dem Client oder dem Buildserver, damit UI-Entwickler diesen Vergleich nicht selber durchführen müssen und die Perfomance beim Client erhöht wird, weil die Bildvergleiche schon vorberechnet sind. Dafür akzeptieren wir die Konsequenz, dass der Benutzer den Bildvergleich nicht konfigurieren kann, was aber in einem späteren Schritt mit einer Komponente für Angular nachgeholt werden kann.

4.1.2 Evaluationskriterien

4.1.2.1 Muss Kriterien

Folgende Kriterien muss eine Bildvergleichskomponente zwingend erfüllen, damit sie auf dem Scenarioo Server eingesetzt werden kann.

- Java 6 Kompatibel
- Opensource Lizenz
- Performanter Bildvergleich

  Maximal 10 Sekunden pro Vergleich. Wünschenswert wäre weniger als 1 Sekunde
- Benötigte Funktionen
  - Berechnung der Abweichung zweier Bilder in Prozent
  - Erstellen eines Bildes welches die Unterschiede zwischen zwei Bildern darstellt

4.1.2.2 Kann Kriterien

Es wäre wünschenswert, wenn die Bildvergleichskomponente die folgenden Kriterien erfüllt.

- Erkennen, ob ein Element neu auf dem Bild ist oder ob sich dieses bloss verschoben hat
- Weitere Konfigurationsmöglichkeiten, wie z.B. Änderungen der Hintergrundfarbe ignorieren
- Keine Installation auf dem Server nötig
- Unterstützung von Windows

4.1.3 Übersicht Komponenten

Es gibt verschiedene Arten, wie Bildvergleiche mit Java implementiert werden. Bei der Suche nach entsprechenden Komponenten ist uns aufgefallen, dass viele Entwickler den Bildvergleich eigenhändig implementieren. Wir sind allerdings der Meinung, mit einer existierenden Lösung eine höhere Performance und eine grössere Stabilität zu erreichen. Die Komponente haben wir in zwei Kategorien unterteilt.

4.1.3.1 Java-Basierte Bibliothek

Unsere Wunschvorstellung wäre es, ein auf Java basierende Bibliothek zu verwenden. Hier wäre der grosse Vorteil, dass keine Softwarekomponente auf dem Server installiert werden müsste. Leider sind wir nicht wirklich fündig geworden. Es gibt gute Bibliotheken für die Bildverarbeitung wie beispielsweise ImageJ oder OpenCV. Diese eigenen sich allerdings besser für das Bearbeiten eines einzelnen Bildes, nicht aber für den Vergleich zwischen zwei Bildern. Die einzige Bibliothek, welche wir gefunden haben, stammt von

Nils Schütte, einem Entwickler von frontendtest.org. Er hat diese Bibliothek für den Vergleich von Printscreens, welche mittels Selenium generiert wurden, erstellt. Also genau für den gleichen Anwendungsfall, wie wir die Bibliothek einsetzen möchten.

4.1.3.2 Softwaresuiten zum Installieren

Der Vorteil bei Softwaresuiten zum Installieren liegt in der Performance. Diese wurden genau für diesen Zweck entwickelt und nutzen durch Parallelisierung die Ressourcen des Servers sehr gut aus. Der Nachteil, wie der Titel schon sagt, liegt darin, dass die Bildvergleichssoftware auf dem Server installiert werden muss. Der Zugriff erfolgt standardmässig über die Konsole. Doch meistens wird für jede Programmiersprache ein Interface angeboten, gegen das bequem programmiert werden kann. In unseren Tests werden wir pdiff und ImageMagick genauer anschauen.

4.1.4 Test der einzelnen Komponenten

Um die Komponenten zu testen haben wir ein Bild der Scenarioo Demo verwendet. In diesem Bild haben wir diverse Änderungen vorgenommen:

- Elemente verschieben
- Elemente hinzufügen/löschen
- Farben verändern
- Listensymbole ändern

Das Originalbild wird jeweils mit dem veränderten Bild verglichen. Dabei messen wir die Zeit und dokumentieren die Verwendung der Komponente sowie das Vergleichsergebnis.

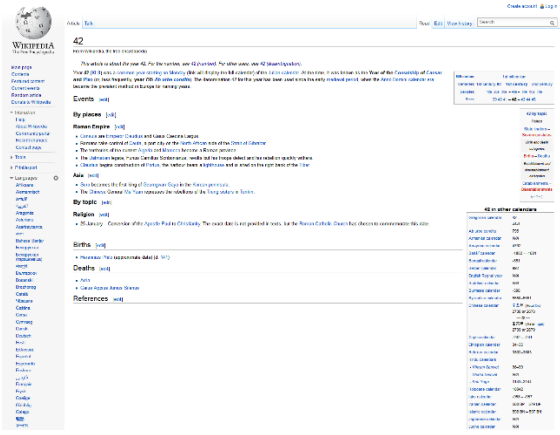


Abbildung 6: Testbild Original

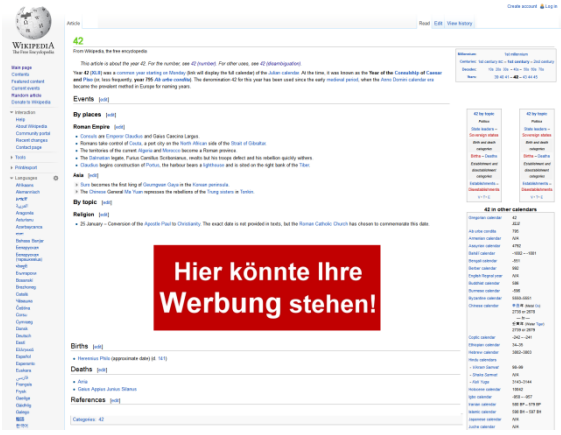
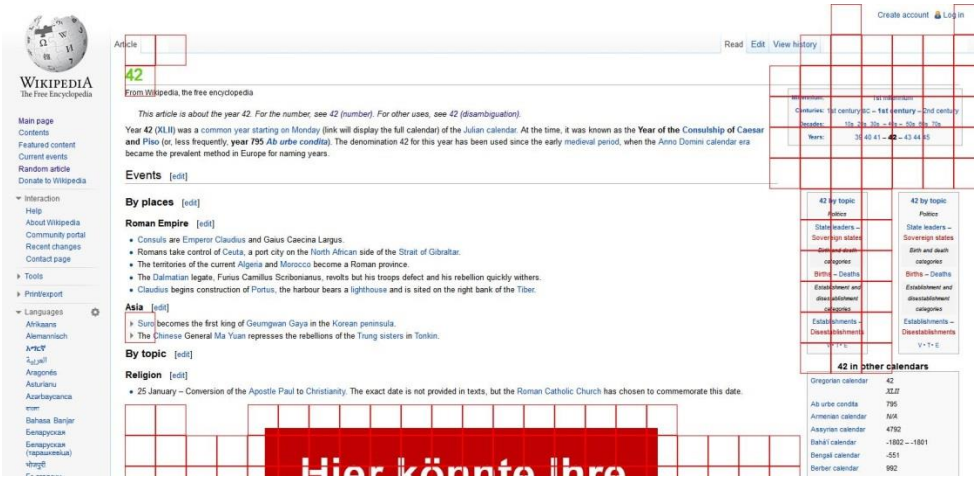
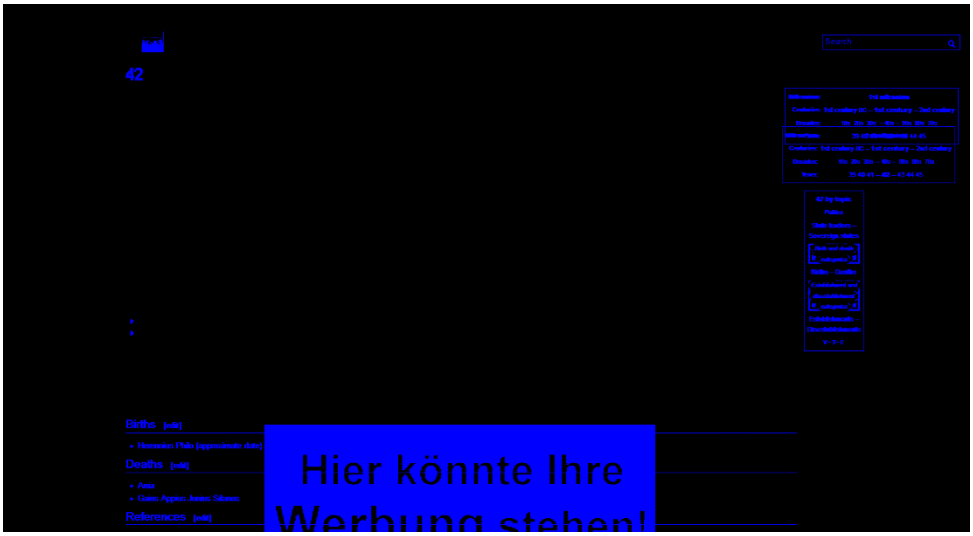


Abbildung 7: Testbild mit Änderungen

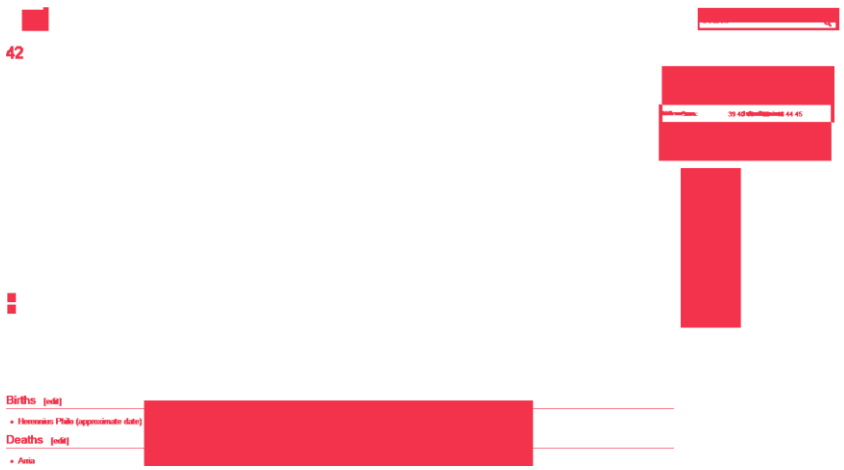
4.1.4.1 frontendtest.org Eigenentwicklung

Beschreibung	Nils Schütte von frontendtest.org hat diese Bibliothek entwickelt, um durch Selenium Tests entstandene Printscreens zu vergleichen.
Details	http://www.frontendtest.org/blog/screenshot-comparison/
Lizenz	MIT License (MIT)
Performance	Dauer Vergleich Testbild: 10 Sekunden bei 50px Quadrat (wie Anzeige unten) 4 Sekunden bei 150px Quadrat (Auch einigermassen gut erkennbar) 1,5 Sekunden bei 5000px Quadrat (Ganzes Bild auf einmal vergleichen)
Konfigurations-möglichkeit	Das ganze Bild wird in Rechtecke unterteilt und verglichen. Dabei kann folgendes eingestellt werden: • Breite/Höhe des Rechtecks • Ab wie viel Prozent Abweichung ein Unterschied dargestellt werden soll.
Anzeige	 Das Bild zeigt mit einem roten Raster an, wo sich die Änderungen befinden.
Berechnung Abweichung	Die Abweichung in Prozent wird standardmässig nicht ausgegeben, könnte aber problemlos dazuprogrammiert werden.
Verwendung	Die gesamte Library besteht aus 172 Zeilen und ist dementsprechend übersichtlich. Es kann als eigene Klasse eingebunden werden. Allerdings werden teilweise noch alte Technologien aus Java 6 verwendet, welche abgeändert werden müssten.

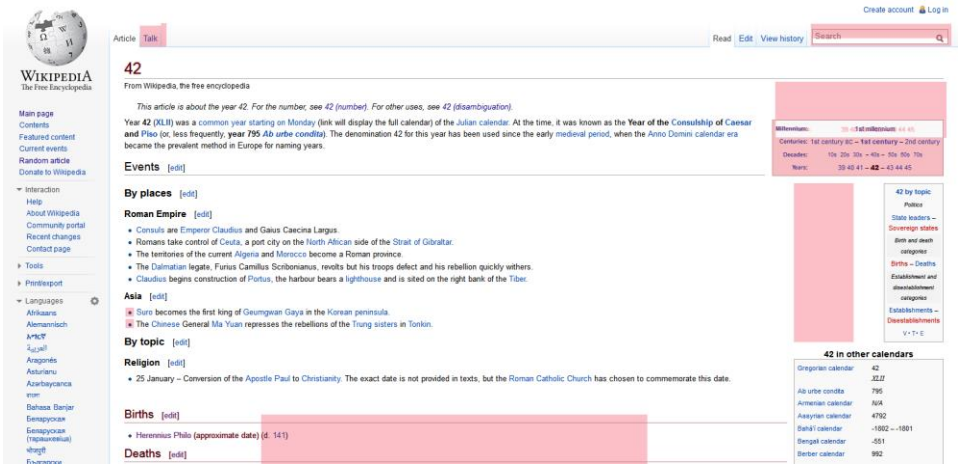
4.1.4.2 Perceptual Image Diff

Beschreibung	PerceptualDiff ist ein Bildvergleichstool, das ein rechnerisches Modell des menschlichen Sehvermögens benutzt, um zwei Bilder zu vergleichen.
Details	http://pdiff.sourceforge.net/
Lizenz	GNU General Public License
Performance	Dauer Vergleich Testbild: 28 Sekunden Änderungen an der Konfiguration brachten keinen Performancegewinn
Konfigurations-möglichkeit	• verbose: Ausgabe Detailinformationen • fov <deg>: Blickfeld in Grad (0.1 bis 89.9) • threshold <p>: Pixelgrenze p unter welcher Änderungen ignoriert werden • gamma <g>: Wert um rgb nach linear space zu konvertieren • luminance <l>: Weisse Leuchtdichte • output <o.ppm>: Die Unterschiede werden in diese Datei geschrieben
Anzeige	 Unterschiede werden in blau dargestellt während der Hintergrund schwarz ist.
Berechnung Abweichung	Die Abweichung wird in der Anzahl Pixel anstelle von Prozent ausgegeben und müsste entsprechend umgerechnet werden.
Verwendung	Pdiff muss zuerst auf dem Betriebssystem installiert werden. Es müsste via Konsolenaufruf aus dem Java Code aufgerufen werden.

4.1.4.3 ImageMagick

Beschreibung	ImageMagick ist eine Softwaresuite zur Erstellung und Bearbeitung von Rastergrafiken.
Details	http://www.imagemagick.org/script/compare.php
Lizenz	ImageMagick: Apache 2.0 license Im4java: LGPL
Performance	Dauer Vergleich Testbild: ca. 0.5-1 Sekunde
Konfigurations-möglichkeit	• metric type: Unterschiede zwischen Bildern mit dieser Metrik berechnen • transparent-color: Transparente Farbe bestimmen. • verbose: Detailinformationen zum Bild ausgeben • compose : Hintergrundauswahl : Transparent oder Bild … und viele weitere. Nachlesbar auf der Herstellerseite
Anzeige	 Es wird ein transparentes Bild abgespeichert, welches nur die Unterschiede darstellt. Der Vorteil des transparenten Bildes liegt in der geringen Dateigrösse von nur 6 Kilobyte. Es besteht auch die Möglichkeit das Ursprungsbild als Hintergrund zu haben, wie bei GraphicsMagick dargestellt. Das abgespeicherte Bild braucht dann allerdings 300 Kilobyte mehr Platz.
Berechnung Abweichung	Die Abweichung wird in Prozent ausgegeben. Es kann die Metrik gewählt werden, nach welchen Bedingungen die Unterschiede berechnet werden.
Verwendung	ImageMagick muss zuerst auf dem Betriebssystem installiert werden. Im4java dient allerdings als pure-java interface zur ImageMagick commandline.

4.1.4.4 GraphicsMagick

Beschreibung	GraphicsMagick ist ein Fork von ImageMagick, der mehrere Vorteile verspricht: • Er soll effizienter sein: Schnellere Berechnung mit weniger Ressourcen • Kleiner und leichter: Die Installationsdateien sind um Faktor 3-5 kleiner • Er wird von den grössten Photoseiten wie Flickr oder Etsy verwendet
Details	http://www.graphicsmagick.org/
Lizenz	X11-style license (MIT License)
Performance	Dauer Vergleich Testbild: ca. 0.5 Sekunden
Konfigurationsmöglichkeit	Ähnlich wie ImageMagick. Parallele Ausführung wird besser unterstützt. Die Compose Funktion fehlt. Eine transparente Ausgabe der Unterschiede ist daher nicht möglich.
Anzeige	 Es wird ein Bild abgespeichert, bei dem die Unterschiede transparent überlagert werden.
Berechnung Abweichung	Die Abweichung wird in Prozent zusammen mit anderen Werten ausgegeben. Der Eigentliche Prozentwert muss herausgefiltert werden.
Verwendung	GraphicsMagick muss zuerst auf dem Betriebssystem installiert werden. Im4java gilt auch als pure-java interface zur GraphicsMagick commandline. Gm4java erweitert die Bibliothek durch weitere Funktionen wie Parallelisierung, erweitert.

4.1.5 Entscheid Bildvergleich

Im Zusammenhang mit der Auswahl der Bildvergleichskomponente und in Anbetracht der nicht-funktionalen Anforderung der Performance, haben wir uns für den Bildvergleich mit ImageMagick und dem Interface Im4java entschieden und nicht für den Bildvergleich mit PerceptualDiff oder der frontendtest.org Eigenentwicklung, damit wir die Unterschiede performant als transparentes Bild abspeichern können und ein sauberes Interface zum Java Code haben. Dafür akzeptieren wir die Konsequenz, dass der Administrator zuerst die Installation von ImageMagick auf dem Server vornehmen muss.

Mit dem Entscheid für ImageMagick müssen wir festlegen, ob wir ImageMagick selbst, oder dessen Fork GraphicsMagick verwenden wollen.

Der grosse Vorteil von ImageMagick gegenüber GraphicsMagick liegt darin, das Diff-Bild transparent abspeichern zu können. Bei unserem Testbild ist das transparente Bild nur 6KB gross. Also um 294KB kleiner als das nicht transparente Bild. Dadurch erhöht sich die Performance beim Laden der Bilder und es braucht viel weniger Speicherplatz auf dem Server. Darum hätten wir uns für ImageMagick entschieden, wenn uns nicht ein folgenschwerer Mangel aufgefallen wäre. ImageMagick hat extrem Mühe beim Vergleich von Bildern, die unterschiedlich gross sind, was öfters auftreten kann. GraphicsMagick hingegen kann unterschiedlich grosse Bilder problemlos vergleichen. Aus diesem Grund werden wir GraphicsMagick für den Bildvergleich einsetzen, was gleichzeitig auch einen kleinen Performancegewinn gegenüber ImageMagick einbringt.

4.2 Datenhaltung

Die Datenhaltung im Scenarioo wird mit XML-Dateien und nicht über eine Datenbank gehandhabt. An diesem Umstand soll auch das neue Diff-Feature nichts ändern. Deshalb sollen die neu zu speichernden Informationen auch in XML-Dateien abgelegt werden.

Die bestehende Ordner- und Dateistruktur wird im nachfolgenden Bild dargestellt. Weitere Informationen zur Struktur können im Scenarioo Github Wiki gefunden werden. https://github.com/scenarioo/scenarioo/wiki/Scenarioo-Writer-Documentation-Format

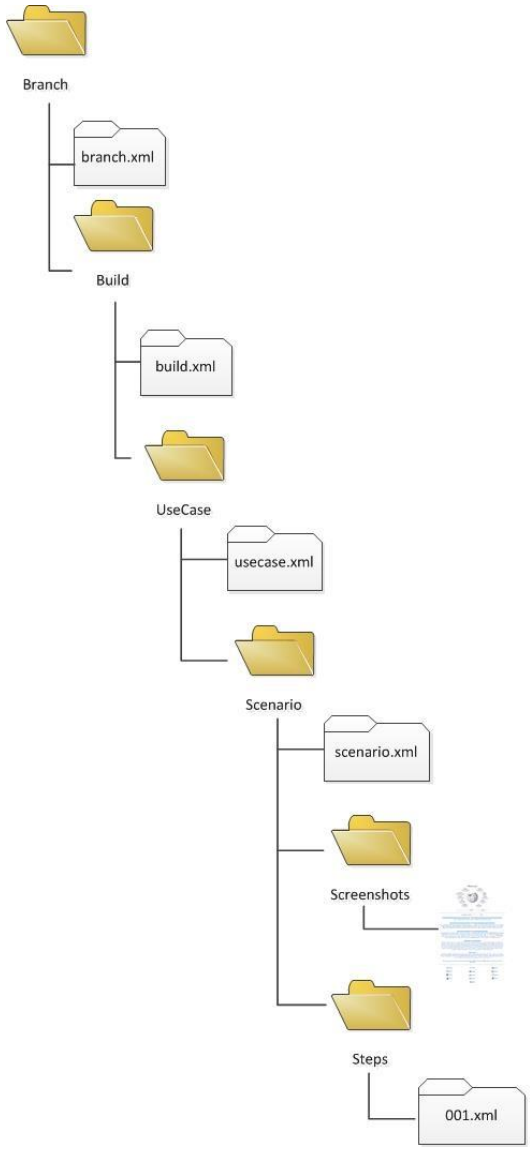


Abbildung 8: Dateistruktur im Scenarioo (Scenarioo Struktur 2016)

Für die Speicherung der Daten sehen wir drei unterschiedliche Möglichkeiten. Alle drei haben ihre Vor- und Nachteile, welche wir hier gerne auflisten.

4.2.1 Alles in einer XML-Datei

Bei diesem Ansatz würden alle Diff-Daten in einer einzigen XML-Datei gespeichert werden. Diese XML-Datei würde in der Ordnerstruktur unter /Branch/Build abgelegt werden.

Die XML-Struktur könnte wie folgt aussehen:

```
<builds>
    <build>
        <id>branch2-build1</id>
        <diffinfo>
            <changed>2</changed>
            <added>1</added>
            <removed>1</removed>
        </diffinfo>
        <usecases>
            <usecase>
                <id>TechnicalCornerCases</id>
                <diffinfo>
                    <changed>2</changed>
                    <added>1</added>
                    <removed>0</removed>
                </diffinfo>
                <scenarios>
                    <scenario>
                        <id>Findnoresults</id>
                        <diffinfo>
                            <changed>1</changed>
                            <added>0</added>
                            <removed>0</removed>
                        </diffinfo>
                        <steps>
                            <step>
                                <id>step1</id>
                                <diffinfo>
                                    <printscreen>24%</printscreen>
                                    <metadata>false</metadata>
                                    <action>true</action>
                                    <screenshotA>001.png</screenshotA>
                                    <screenshotB>002.png</screenshotB>
                                    <screenshotCompare>001.png</screenshotCompare>
                                </diffinfo>
                            </step>
```

Abbildung 9: XML-Ausschnitt mit der Variante alles in einer XML-Datei

Vorteile

- Ideal für Diff Overview: Es muss nur ein File eingelesen werden
- Kein Overhead durch Dateistruktur
- Daten von einem Buildvergleich können einfach gelöscht werden

Nachteile:

- Die XML-Datei kann unter Umständen sehr gross werden
- Es muss immer alles geladen werden, auch wenn einem nur die oberste Ebene (Us Case) interessieren würde

4.2.2 Separate Dateistruktur

In dieser Variante werden die Daten in viele kleine XML-Dateien analog zur bestehenden XML-Struktur angelegt. Damit es zu keinen Konflikten mit den bestehenden Daten kommt, werden die Daten in einer separaten Ordnerstruktur abgelegt. Dies könnte dann wie folgt aussehen:

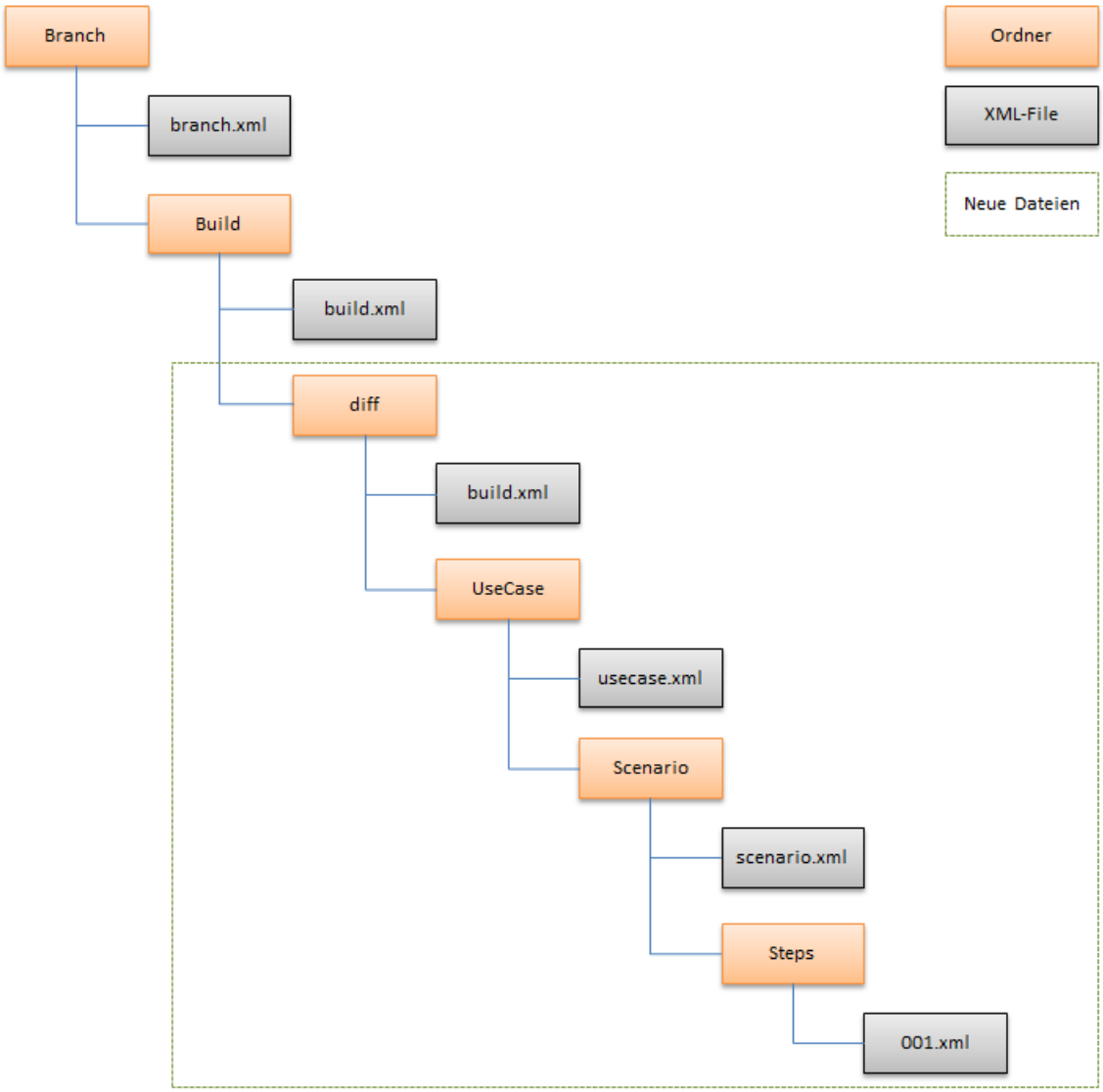


Abbildung 10: Abbildung der separaten XML-Struktur

In den jeweiligen XML-Dateien würden jetzt nur die Diff-Informationen stehen.

Für einen Use Case könnte dies dann so aussehen:

```
<usecase>
    <id>TechnicalCornerCases</id>
    <diffinfo>
        <changed>2</changed>
        <added>1</added>
        <removed>0</removed>
    </diffinfo>
</usecase>
```

Abbildung 11: XML für ein Use Case mit Diff-Informationen

Vorteile:

- Kleine XML-Dateien

- Selektives laden der Informationen möglich
    - Daten von einem Buildvergleich können einfach gelöscht werden

Nachteile:

- Mühsamer um alle Informationen zusammenzusuchen
- Zusätzliche Dateistruktur pro Buildvergleich

4.2.3 In bestehende XML-Struktur integriert

Als dritte Variante ziehen wir in Betracht, die Daten in bestehende XML-Dateien zu integrieren. Dazu würde in den XML-Dateien ein neues Element mit den Diff-Informationen eingefügt.

Bei einem Use Case könnte das resultierende XML dann wie folgt aussehen:

```
<useCase>
    <name>LookUpPhoneNumber</name>
    <description>Look up phone number of "Hot Pizza Delivery".</description>
    <details/>
    <labels/>
    <diffs>
        <diffinfo>
            <branch>branch2-build1</branch>
            <build>build1</build>
            <changed>2</changed>
            <added>1</added>
            <removed>1</removed>
        </diffinfo>
    </diffs>
</useCase>
```

Abbildung 12: XML für ein Use Case mit Diff-Informationen integriert

Vorteile:

- Bestehende Funktionalität kann erweitert/verwendet werden
- Selektives laden der Informationen ist möglich

Nachteile:

- Diff-Informationen müssen beim Löschen aus jeder Datei gelöscht werden

4.2.4 Entscheid Datenhaltung

Im Zusammenhang mit der XML-Datenhaltung, und in Anbetracht der nicht-funktionalen Anforderungen der Effizienz und der Wartbarkeit, haben wir uns für die Variante 2 „Separate Dateistruktur“ und nicht für die Varianten 1 und 3 entschieden, um eine gute Performance und übersichtlichen Code zu erreichen. Dafür akzeptieren wir die Konsequenz, dass wir die REST-API erweitern müssen und eigene Entities definieren müssen.

4.3 Stepvergleich

Beim Buildvergleich werden auf unterster Ebene die Steps miteinander verglichen. Damit wir die richtigen zwei Steps miteinander vergleichen, ist es wichtig, dass diese eindeutig identifiziert werden können. Sollte dies nicht möglich sein, machen alle weiteren Vergleiche keinen Sinn.

4.3.1 Anhand von Bildern

Eine Möglichkeit die Steps zu identifizieren, wäre anhand der Screenshots. Man würde den Screenshot vom Step A vom Build 1 mit allen Steps im entsprechenden Szenario vom Build 2 vergleichen und dann den Step A dem Step zu ordnen, welcher die grösste Ähnlichkeit im Bildvergleich hatte.

Anhand einer Grafik lässt sich das besser erkennen.

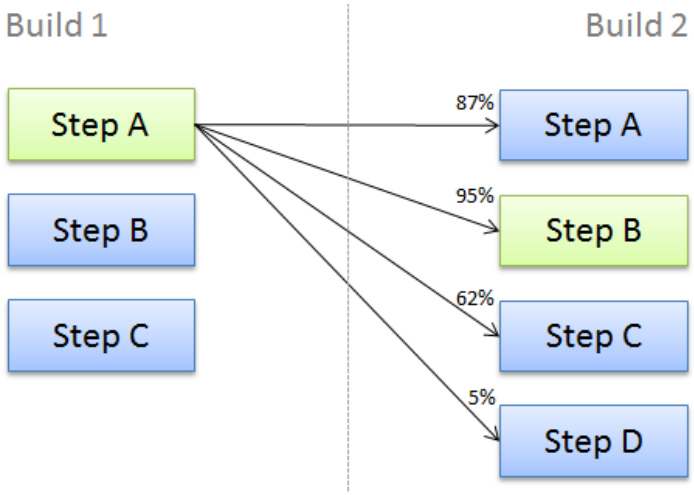


Abbildung 13: Stepidentifizierung anhand von Screenshots

Das grosse Problem bei diesem Ansatz ist die Frage, ab wie viel Prozent ein Step neu ist und ab wie viel Prozent er nur geändert hat? Es könnte sein, dass in obiger Abbildung eigentlich Step A vom Build 2 auf Step A vom Build 1 passen würde, dieser Step allerdings eine grössere Änderung am Design erfahren hat und deshalb jetzt fälschlicherweise dem Step B zugewiesen wird.

Vorteile:

- Rückwärtskompatibilität
- Bereits vorhandene Step-Informationen sind ausreichend

Nachteile:

- Schlechte Performance, da Bildvergleich notwendig
- Keine exakte Zuweisung in jedem Fall möglich
- Stepreihenfolge nicht in jedem Fall bestimmbar

4.3.2 Anhand von neuem Step Identifier

Leider gibt es aktuell noch keine Möglichkeit, einen Step über mehrere Builds hinweg eindeutig zu identifizieren. Deshalb schlagen wir vor, dass beim Speichern der Step-Informationen ein neuer Parameter übergeben werden muss, welcher später eine eindeutige Identifizierung vom Step zulässt.

Es gibt verschiedene Wege, einen solchen Step Identifier zu erstellen. Beispielsweise könnte einem Step jeweils manuell ein Identifier zugeteilt werden. Eine weitere Variante mit geringerem Aufwand wäre, den Identifier automatisch anhand der Screen Annotations (Actions) generieren zu lassen.

Vorteile:

- Steps können anhand von Identifier eindeutig zugewiesen werden
- Es werden immer die gleichen Steps miteinander verglichen
- Die Stepreihenfolge kann bestimmt werden
- Kein Performanceverlust durch zusätzliche Bildvergleiche

Nachteile:

- Es muss ein Identifier beim Speichern der Steps mitgegeben werden

4.3.3 Entscheid Stepvergleich

Im Zusammenhang mit der Stepidentifizierung und in Anbetracht der nicht-funktionalen Anforderung der Funktionalität, haben wir uns für einen neu einzuführenden Stepidentifier und nicht für den Bildvergleich entschieden um die Richtigkeit bei der Stepzuweisung zu erreichen. Dafür akzeptieren wir eine Anpassung beim Speichern der Steps.

Bei der Besprechung dieses Entscheids, wurde bekannt, dass Zühlke bereits über die Einführung von einem eindeutigen Stepidentifier nachdenkt. Allerdings möchten sie einen solchen Schritt erst zusammen mit dem neuen Dokumentationsformat 3.0 einführen.

Mit dem nun gefällten Entscheid, werden wir die Steps so gut es geht mit den bestehenden Indexen wie PageOccurrence, Page und InPageIndex identifizieren. So kann leider nicht garantiert werden, dass immer die richtigen zwei Steps miteinander verglichen werden. Dies wird aber so in einer ersten Version in Kauf genommen.

5 Proof of Concept Prototyp

Anschliessend an die Evaluation haben wir einen Proof of Concept Prototyp umgesetzt um zu überprüfen, ob sich die getroffenen Architekturentscheidungen nahtlos in die bestehende Systemlandschaft von Scenarioo integrieren lassen.

5.1 Aufbau Testumgebung

Es war uns wichtig, unseren Prototyp direkt in Scenarioo zu integrieren. Deshalb bestand der erste Schritt aus dem Aufbau einer Testumgebung. Wir verwendeten dazu ein bestehendes Beispielprojekt, welches die Scenarioo Writer Libary nutzt um aus Selenium Webtests eine Scenarioo Dokumentation anzulegen. Dieses Beispielprojekt nennt sich pizza-delivery und beinhaltet eine Webseite um Pizza zu bestellen. Details zum Projekt befinden sich auf folgender Github Seite:

https://github.com/scenarioo/pizza-delivery

5.2 Datenhaltung

Scenarioo arbeitet nicht mit einer Datenbank sondern persistiert die Daten in XML-Dateien. Aus diesem Grund war es unerlässlich in unserem Prototyp zu prüfen, ob wir die Diff-Infos in diesen XML-Dateien abspeichern können. Dadurch konnten wir in Erfahrung bringen, wie bei Scenarioo mit XML gearbeitet wird und wie wir am effizientesten Daten abspeichern und auslesen können.

5.3 Strukturvergleich

Wir haben ein Architekturkonzept erstellt, welches den Strukturvergleich mit Klassen- und Sequenzdiagrammen beschreibt. Die entsprechenden Dokumente können im Architekturabschnitt eingesehen werden. Um dieses Konzept zu überprüfen, haben wir den Strukturvergleich rudimentär implementiert und getestet. Dadurch konnten wir sicherstellen, dass unser Konzept funktioniert und die geforderten Resultate liefert.

5.4 Bildvergleich

Die Bildvergleichskomponente haben wir bereits in der Evaluation ausgiebig getestet. Im Prototyp ging es zusätzlich um die Abklärung, wie dieses Framework elegant in den bestehenden Code eingefügt werden kann. Dazu haben wir in Scenarioo verwendete Komponenten wie Gradle oder log4j auch beim Bildvergleich eingesetzt. Gleichzeitig haben wir erste Unit Tests für den Bildvergleich gemäss Programmierrichtlinien geschrieben.

5.5 Frontend

Beim Prototyp wollten wir einen Architekturdurchstich erreichen. Das bedeutet, die Daten aus den XML-Dateien auszulesen und via REST-Schnittstelle an das Frontend zu übergeben und entsprechend darzustellen. Dazu haben wir die bestehende REST-Schnittstelle auf dem Scenarioo Server analysiert und die Diff-Infos darin integriert. So konnten wir auf dem Client bereits anzeigen, wie sich die Struktur von einem Use Case geändert hat.

6 Umsetzungskonzept

6.1 Datenhaltung

Die Datenhaltung wird in der bestehenden Applikation bereits mit XML- Dateien erledigt. Das heisst für uns, dass wir unsere Daten auch im XML-Format abspeichern werden.

Für den Ort der Datenspeicherung stehen uns mehrere Möglichkeiten offen. Grob gesagt können wir die Diff-Informationen in die bestehenden Scenarioo-Daten integrieren oder eine separate XML-Struktur für die Diff-Informationen aufbauen.

Welches davon die bessere Variante ist, muss zuerst gründlich evaluiert, besprochen und getestet werden.

6.2 Bildvergleich

Um herauszufinden, ob sich ein Screenshot verändert hat, müssen wir einen Bildvergleich der beiden Screenshots durchführen. Für diesen Zweck soll ein entsprechendes Framework evaluiert werden. Die Evaluation wird anhand von diversen nicht-funktionalen wie auch funktionalen Anforderungen durchgeführt.

Sobald ein geeignetes Framework evaluiert ist, wird dieses in einem Prototyp in die bestehenden Scenarioo Umgebung rudimentär eingebaut. So können wir überprüfen, ob das Tool wirklich alle Anforderungen auch mit realen Daten erfüllt und ob es sich überhaupt in das Projekt integrieren lässt.

Wenn klar ist, dass bei der Evaluation die richtige Wahl getroffen wurde, kann mit dem Feinschliff begonnen werden. Das heisst, es muss die richtige Vergleichsmetrik gefunden werden und der gesamte Bildvergleich muss sauber implementiert werden. Ebenso muss dieses Feature unbedingt mit Testfällen abgedeckt werden.

6.3 Strukturvergleich

Im Strukturvergleich gilt es neue oder gelöschte Elemente zu erkennen. Das heisst, wir müssen wissen, welcher Use Case beispielsweise in einem Build neu hinzugekommen ist oder entfernt wurde. Diesen Vergleich werden wir im Gegensatz zum Bildvergleich komplett selber implementieren.

Dabei gilt es zu beachten, dass die Strukturvergleich-Komponenten gut testbar und erweiterbar sein müssen. Auch hier werden wir wieder zuerst die grobe Logik entwerfen und dann implementieren.

6.4 Grafisches Design

Beim grafischen Design gilt es zwei wichtige nicht-funktionale Anforderungen zu berücksichtigen. Das Design soll sich nahtlos in die bestehende Lösung einfügen und neue Funktionalität soll benutzerfreundlich sein.

Damit wir diese beiden Ziele erreichen, werden wir darauf achten, möglichst viele bestehende Elemente von der Webseite wieder zu verwenden. So hat der Benutzer einen kleineren Lernaufwand, wenn er die ihm bekannten Elemente wiederverwenden kann.

Die Bedienbarkeit werden wir nach Umsetzung der Funktionalität durch einen Usability Test überprüfen lassen. Die daraus resultierenden Erkenntnisse werden dazu verwendet, die Bedienbarkeit noch weiter zu optimieren.

6.5 End-To-End Tests

Scenarioo wird dazu verwendet, Use Cases und deren Szenarien von einer Webseite Schritt für Schritt mit Screenshots zu dokumentieren. Dies kann Scenarioo natürlich auch auf sich selber anwenden und sich so selbst dokumentieren.

Wir möchten diesen Ansatz weiterverfolgen. Die neu ermöglichten Use Cases vom neuen Diff-Feature sollen ebenfalls in Scenarioo dokumentiert werden. Dazu müssen wir für jegliche Anwendungsfälle die entsprechenden End-To-End Tests schreiben, welche daraus die Dokumentation generieren.

7 Resultate, Bewertung und Ausblick

7.1 Zielerreichung

Zusammengefasst sind folgende Ziele in der Aufgabenstellung definiert:

- Das Hauptziel der Arbeit ist die Erstellung einer produktiv verwendbaren Erweiterung zu Scenarioo, im Sinne eines „Minimum Viable Product“.
- Unterschiede zweier Snapshots der Systemdokumentation zu visualisieren
- Signifikante Änderungen werden auf jeder Detailebene der Dokumentation ansprechend dargestellt
- Es sollen sowohl strukturelle Änderungen der Use Cases und ihrer Szenarien wie auch visuelle Änderungen der Screenshots berücksichtigt werden
- Es sollen möglichst effiziente Algorithmen zum Struktur- und Bildvergleich gefunden und eingesetzt werden
- Auf gute Usability ist besonderen Wert zu legen

In Zusammenarbeit mit Scenarioo-Entwicklern und Scenarioo-Benutzern konnten wir die exakten Anforderungen an das Endprodukt ausarbeiten. Dadurch konnten wir feststellen, welche Änderungen signifikant sind und wie diese visualisiert werden sollen. In einer intensiven Evaluationsphase fanden wir mit GraphicsMagick ein optimales Werkzeug für den Screenshot-Vergleich. Sowohl die Performance bei der Berechnung, als auch die Visualisierung der Screenshot-Unterschiede haben unsere Erwartungen übertroffen. Die Berechnung sämtlicher Unterschiede im grössten existierenden Scenarioo-Kundenprojekt dauert gerade einmal 15 Minuten.

Ein Usability-Test hat gezeigt, dass mit dem Diff Viewer Strukturelle- sowie visuelle Änderungen umgehend erkannt werden. Ausserdem konnten wir mit den dadurch gesammelten Erfahrungen die Usability nochmals steigern.

Zühlke konnte uns bestätigen, dass die Ziele vollumfänglich und gemäss ihrer Priorisierung erreicht wurden. Der erreichte Funktionsumfang im Sinne eines MVP entspricht ihren Vorstellungen und die Qualität der Funktionen, insbesondere bezüglich Usability und UI Design, sei hervorragend.

7.2 Weiterentwicklung

7.2.1 Mögliche Features

Es gibt diverse Möglichkeiten wie der Diff Viewer weiterentwickelt werden kann. Schon nach den Benutzerinterviews wurde uns klar, dass wir niemals alle Ideen im Minimum Viable Product umsetzten können.

Sämtliche User Stories ab der 2. Priorität sind Features, die aus Zeitgründen noch nicht umgesetzt sind. Grob zusammengefasst sind folgende Features noch nicht entwickelt:

- Ausweitung der Vergleiche auf Metadaten, HTML Code, oder Annotations
- Gesamtübersicht über alle Änderungen
- Build-Vergleich zur Laufzeit anstossen
- Bessere Page- und Stepübersicht. Dies gilt nicht nur für die Diff Viewer Ansicht.

7.2.2 Support

Unser Fork wird in den offiziellen Scenarioo-Master-Branch integriert. Bei Fragen kann man sich jederzeit an die Scenarioo-Community wenden:

https://github.com/scenarioo/scenarioo

7.3 Persönliche Berichte und Danksagung

Die persönlichen Berichte und die Danksagung werden nicht veröffentlicht.

II. SW-Projektdokumentation

1 Versionierung

Version	Datum	Verantwortlicher	Änderung
0.1	29.02.16	mscheube	Dokument angelegt
0.2	07.03.16	pforster	Kapitel Risiken eingefügt
0.3	07.03.16	mscheube	Kapitel Projektmanagement eingefügt
0.4	14.03.16	pforster	Projektplanung überarbeitet
0.5	21.03.16	mscheube	Kapitel Einführung eingefügt
0.6	21.03.16	pforster	Kapitel Stand der Technik eingefügt
0.7	28.03.16	mscheube	Nicht-funktionale Anforderungen eingefügt
0.8	11.04.16	mscheube	Evaluation Datenhaltung und Stepvergleich eingefügt
0.9	11.04.16	mscheube	Kapitel Design eingefügt
0.10	11.0416	pforster	Evaluation Bildvergleichskomponente eingefügt
0.11	11.04.16	pforster	Kapitel Proof Of Concept eingefügt
0.12	22.05.16	pforster	Beschreibung vom Änderungsgrad eingefügt
0.13	23.05.16	pforster	Usability Test eingefügt
0.14	07.06.16	pforster	Implementation ergänzt
0.15	08.06.16	pforster	Code Abgrenzung in Implementation eingefügt
0.16	08.06.16	pforster	Stand der Technik -> Ausgangslage. Neue Stand der Technik eingefügt
0.17	08.06.16	mscheube	Kapitel Barrierefreiheit eingefügt
0.18	08.06.16	mscheube	Kapitel Designkonzept eingefügt
0.19	08.06.16	mscheube	Abstract und Management Summary eingefügt
0.20	09.06.16	pforster	Testing im Kapitel Implementation eingefügt
0.21	15.06.16	pforster	Gesamtdokumentation überarbeiten
0.22	16.06.16	mscheube	Gesamtdokumentation überarbeiten
0.23	17.06.16	Mscheube	Finale Version

2 Vision

Die Vision wird im Technischen Bericht im Kapitel „Einführung“ beschrieben.

3 Anforderungsspezifikation

3.1 Funktionale Anforderungen

3.1.1 Benutzerinterviews

Um die Anforderungen zu bestimmen, führten wir Gespräche mit Scenarioo Benutzern und Entwicklern aus folgenden Berufsgattungen:

- Expert Software Engineer. Sehr erfahrener Web/UI-Spezialist
- Solution Designer bei einem grösseren Kunden der Scenarioo im Einsatz hat
- Principle Consultant. Sehr erfahrener UX-Experte und Business Analyst
- Lead Software Architekt, auch in Projekten mit Scenarioo
- Software Architekt und Leiter des Scenarioo-Entwicklungsteams
- Expert Software Engineer und Mitentwickler des Open Source Projektes Scenarioo

Wir befragten die Interviewkandidaten sowohl nach der jetzigen Nutzung von Scenarioo als auch nach den Anforderungen für das neue Diff Viewer Feature. Wir haben die Aussagen der Kandidaten fortlaufend protokolliert, sowie für spätere Detailanalysen auf Video festgehalten. Der Fragebogen sowie eine anonymisierte Auswertung der wichtigsten Punkte befinden sich im Anhang.

Im Kapitel „Stand der Technik“ ist dokumentiert, wie Scenarioo momentan von den unterschiedlichen Benutzergruppen verwendet wird. Die Anforderungen an das neue Feature sind in den nachfolgenden Benutzerszenarien und User Stories beschrieben.

3.1.2 Mögliche Benutzerszenarien

3.1.2.1 Agile Tester: Änderungen im letzten Sprint

Ein Tester möchte wissen, welche Änderungen im letzten Sprint (oder am letzten Tag oder seit dem letzten Release) vorgenommen wurden, d.h. in welchen Use Cases, in welchen Szenarien, in welchen Screenshots gab es welche wesentliche Änderungen. Der Tester will dann diese Änderungen einerseits verifizieren (gab es evtl. ungewollte Änderungen?) als auch als Ausgangslage/Anhaltspunkt für weitere manuelle Tests nutzen.

3.1.2.2 Veränderungen an CSS-Styles durch Entwickler

Ein Entwickler hat die CSS-Styles für eine Page angepasst. Er möchte nun überprüfen können, ob sich seit dem letzten Scenarioo Build grosse Änderungen auf Screenshots ergeben haben.

3.1.3 User Stories

3.1.3.1 Use Case Diagramm

Im nachfolgenden Use Case Diagramm sind die Epics des Diff Viewer Features dargestellt.

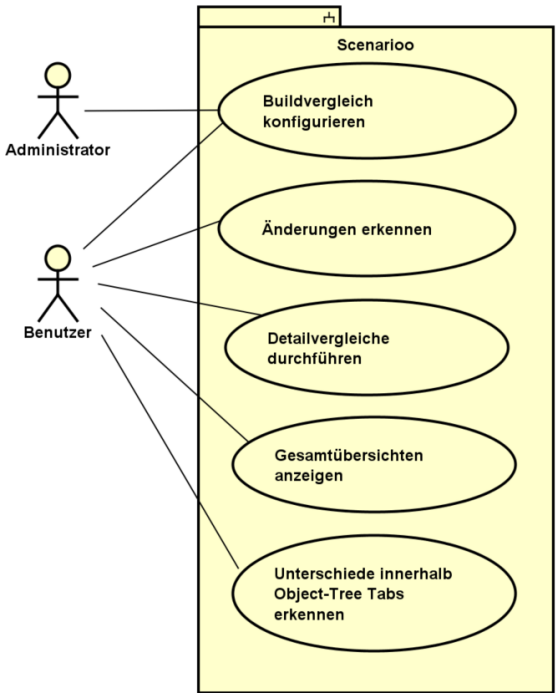


Abbildung 14: Use Case Diagramm Diff Viewer Feature

3.1.3.2 Rollen

Die zwei Rollen von Scenarioo werden in der nachfolgenden Tabelle beschrieben:

Tabelle 1: Rollenbeschreibung

Rolle	Beschreibung
Benutzer	Ist diejenige Person welche auf die Weboberfläche von Scenarioo zugreift, um Informationen abzurufen.
Administrator	Der Applikationsverantwortliche, welcher die Applikation betreibt, konfiguriert und wartet.

3.1.3.3 Prioritäten

In Zusammenarbeit mit Zühlke haben wir die User Stories mit folgenden Prioritätsstufen priorisiert.

Tabelle 2: Prioritätsstufen

Priorität	Beschreibung
1. Priorität	Wird, falls zeitlich möglich, im Rahmen dieser Bachelorarbeit umgesetzt
2. Priorität	Wird umgesetzt, falls alle User Stories erster Priorität fertiggestellt sind und noch genügend Zeit vorhanden ist.
3. Priorität	Wird umgesetzt, falls alle User Stories zweiter Priorität fertiggestellt sind und noch genügend Zeit vorhanden ist.
4. Priorität	Wird anschliessend an diese Arbeit von Scenarioo Contributors entwickelt

3.1.4 Epics und User Stories

Alle Epics und User Stories sind gemäss den Anforderungen von Scenarioo-Entwicklern und Scenarioo-Usern erfasst worden. In einer ersten Phase wurden diese von Zühlke abgesegnet und priorisiert. Vor der Implementierung einer User Story haben wir diese jeweils auf GitHub als Issue veröffentlicht. Dabei haben wir zusätzlich die Akzeptanzkriterien sowie die vorgesehenen Testfälle dokumentiert. Dadurch kann die Scenarioo-Community den Fortschritt laufend verfolgen und erhält die Möglichkeit, Ideen einzubringen und zu kommentieren. Die Issues sind unter folgender URL aufrufbar:

https://github.com/scenarioo/scenarioo/milestones/3.x%20Diff%20Viewer%20Release

Normalerweise sind bei User Stories keine Bilder enthalten. Sobald unser UI-Prototyp vorhanden war, haben wir bei gewissen User Stories Designentwürfe hinterlegt. Gerade bei Lesern, die Scenarioo nicht so gut kennen, erhöhte dies das Verständnis enorm.

Epic 1: Buildvergleich konfigurieren
Als Administrator will ich konfigurieren können wie Builds verglichen werden, so dass die Diff Funktion speziell auf unser Projekt angepasst werden kann.
Akzeptanzkriterien • Es soll angegeben werden können, mit welchen Builds verglichen werden soll • Es besteht die Möglichkeit, immer mit dem letzten Build zu vergleichen • Es kann angegeben werden, welche Metadaten verglichen werden • Der Benutzer kann auswählen welchen vorberechneten Vergleich er ansehen möchte • Die Diff-Informationen sollen ausgeblendet werden können

User Story 1.1: Vergleich-Builds konfigurieren
Als Administrator will ich konfigurieren können mit welchen Builds der aktuelle Build verglichen werden kann, sodass die Diff Funktion automatisch auf diese angewendet wird.
1. Priorität (noch ohne GUI) 4. Priorität (umsetzen vom GUI)

User Story 1.2: Latest Build als Vergleich-Build auswählen
Als Administrator will ich konfigurieren können, ob immer mit dem letzten Build verglichen werden soll, sodass der letzte Build nicht jedes Mal neu gesetzt werden muss.
1. Priorität (noch ohne GUI) 4. Priorität (umsetzen vom GUI)

User Story 1.3: Comparable-Build auswählen
Als Benutzer will ich jederzeit auswählen können mit welchem der vorberechneten Vergleich-Builds verglichen werden soll, sodass nur Änderungen zwischen dem aktuellen Build und dem ausgewählten Build angezeigt werden.
1. Priorität

User Story 1.4: Metadatenauswahl konfigurieren
Als Administrator will ich konfigurieren können, welche Metadaten in diesem Projekt mit dem Diff Viewer verglichen werden, sodass nur die wichtigen, eher statischen Metadaten Änderungen angezeigt werden
4. Priorität

User Story 1.5: Vergleichs-Modus beenden
Als Benutzer will ich aus dem Vergleichs-Modus wieder in den normalen Modus wechseln können, sodass ich keine Diff-Informationen mehr sehe.
1. Priorität

User Story 1.6: Übersicht Buildvergleiche
Als Administrator will ich eine Übersicht aller Buildvergleiche, sodass ich sehe wo aktuell gerade ein Buildvergleich stattfindet, welche abgeschlossen sind oder wo Probleme beim Buildvergleich aufgetreten sind.
3. Priorität

User Story 1.7: Vergleich-Builds zur Laufzeit vergleichen
Als Benutzer will ich zur Laufzeit den Vergleich zweier Builds anstossen können, sodass ich nicht auf vorkonfigurierte Vergleich-Builds eingeschränkt bin.
2. Priorität

User Story 1.8: Comparison Panel auf der Use Case Übersicht
Als Benutzer will ich auf der Startseite erkennen können, welche vorberechneten Vergleiche vorhanden sind, sodass ich sofort sehe ob es sich lohnt einen Vergleich zu starten.
4. Priorität

Epic 2: Änderungen erkennen

Als Benutzer will ich erkennen können, was sich im Vergleich zu einem anderen Build geändert hat, sodass ich leicht zu den Details dieser Änderung navigieren kann.

Akzeptanzkriterien

- Auf einen Blick soll erkennbar sein, was sich im Vergleich zu einem anderen Build geändert hat, was neu ist und was entfernt wurde
- Es soll möglich sein weitere Detailinformationen anzuzeigen, bevor man eine Ebene tiefer navigieren muss
- Anhand von einem Änderungsgrad soll erkenntlich sein, wie schwerwiegend die Änderung ist
- Schwerwiegende Änderungen sollen farblich hervorgehoben werden

User Story 2.1: Use Case Änderungen erkennen

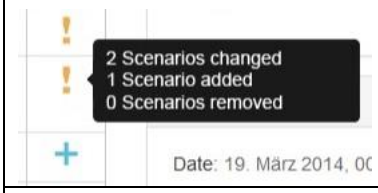
Als Benutzer will ich erkennen können, ob Use Cases Änderungen haben, neu sind oder entfernt wurden, sodass ich sofort sehe, welche Use Cases von Änderungen betroffen sind.

Status	Name	Description	# Scen
success	Donate	Donate money to Wikipedia.	1
success	Find Page normal-case	User wants to search for a page and read it.	4
success	Switch Language	Search in a different language and switch language of current article.	1
success	Technical Corner Cases corner-case	Just some meaningless dummy scenarios for testing some corner cases in the Scenarioo web application, like what happens when there are no pages or a page has only one variant in all scenarios etc.	4
success	Buy a Product	Find a product and buy it	1
success	Feedback	Select a product and give a feeback to it.	3

1. Priorität

User Story 2.2: Use Case Änderungsdetails anzeigen

Als Benutzer will ich mir Zusatzinformationen zu einem Use Case anzeigen lassen, sodass ich erkenne, ob Szenarien in diesem Use Case hinzugefügt, gelöscht oder geändert wurden.



1. Priorität

User Story 2.3: Szenario Änderungen erkennen

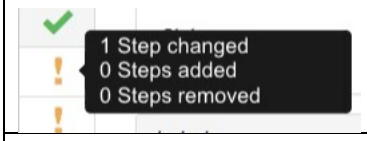
Als Benutzer will ich erkennen können, ob Szenarien Änderungen haben, neu sind oder entfernt wurden, sodass ich sofort sehe, welche Szenarien von Änderungen betroffen sind.

Status	Scenario name	Actions	Description	# St
success	Find multiple results happy		User enters some text and finds multiple pages that contain this text.	5
success	Find no results		User enters text that is not found in pages content.	3
success	Find page title ambiguous directly		User enters exact page title that has ambiguities, he is navigated to the most relevant page directly and sees the ambiguities on top of the page.	7
success	Find page title unique directly		User enters exact unique page title and is navigated to the page directly.	3

1. Priorität

User Story 2.4: Szenario Änderungsdetails anzeigen

Als Benutzer will ich mir Zusatzinformationen zu einem Szenario anzeigen lassen, sodass ich erkenne, ob Steps in diesem Szenario hinzugefügt, gelöscht oder geändert wurden.



1. Priorität

User Story 2.5: Szenarioübersicht: Page Änderungen erkennen

Als Benutzer will ich mir in der Szenarioübersicht Zusatzinformationen zu einer Page anzeigen lassen, sodass ich erkenne, ob die darin enthaltenen Steps Änderungen haben, neu sind oder entfernt wurden

1. Priorität

User Story 2.6: Szenarioübersicht: Step Änderungen erkennen

Als Benutzer will ich mir in der Szenarioübersicht Zusatzinformationen zu einem Step anzeigen lassen, sodass ich erkenne, ob der Screenshot, die Metadaten oder die Screen Annotation geändert hat

1. Priorität

User Story 2.7: Vergleichsdaten nach Reimport entfernen

Als Benutzer will ich einen Build Reimportieren können ohne das mir danach falsche Vergleichsdaten angezeigt werden, sodass alte Vergleichsdaten entweder gelöscht werden oder als ungültig markiert sind

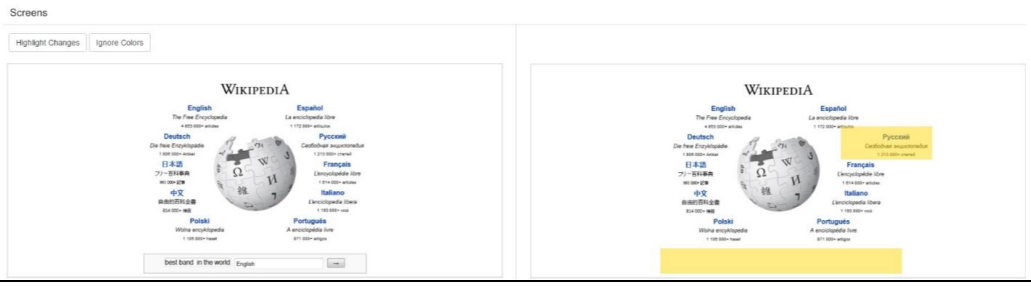
3. Priorität

User Story 2.8: Nach Änderungsgrad sortieren
Als Benutzer will ich die beiden Listen "Use Cases" und "Szenarien, nach dem Änderungsgrad sortieren können, sodass ich z.B. die Einträge mit den grössten Veränderungen schnell und einfach prüfen kann, ohne die ganze Liste durchzugehen.
1. Priorität 4. Priorität: Die Sortierung der Liste soll gespeichert werden und gilt auch für Unterseiten

User Story 2.9: Visuelle Qualifikation der Änderungen
Als Benutzer will ich anhand von Farben erkennen wie schwerwiegend die Änderungen sind, sodass ich darauf aufmerksam gemacht werde wo besonders viel geändert hat.
1. Priorität

Epic 3: Detailvergleiche durchführen
Als Benutzer will ich alle Informationen in einem Step mit denen eines anderen Builds einfach vergleichen können, sodass ich sofort sehe was sich bei diesem Step geändert hat.
Akzeptanzkriterien • Es ist auf einen Blick ersichtlich, was sich auf dem aktuellen Screenshot im Vergleich zum Screenshot des anderen Builds geändert hat • Die Metadaten lassen sich gegenüberstellen und vergleichen • Es wird ersichtlich, ob sich in einem Step die Screen Annotation unterscheidet

User Story 3.1: Step Änderungsübersicht
Als Benutzer will ich sofort erkennen können, ob und wie stark sich beim geöffneten Step der Screenshot im Vergleich zum Screenshot des Vergleich-Builds verändert hat, sodass ich nicht manuell nachschauen muss, ob eine Änderung vorliegt.
1. Priorität

User Story 3.2: Screenshot-Vergleich
Als Benutzer will ich eine Gegenüberstellung der Screenshot seines Steps anzeigen können, sodass ich einfach erkennen kann, welche Unterschiede die Screenshots aufweisen.

1. Priorität

User Story 3.3: Screen Annotation Vergleich

Als Benutzer will ich eine Gegenüberstellung der Screen Annotations eines Steps anzeigen können, sodass ich einfach erkennen kann, welche Unterschiede die Screen Annotations aufweisen.

4. Priorität

User Story 3.4: Metadaten Vergleich

Als Benutzer will ich eine Gegenüberstellung der Metadaten eines Steps anzeigen können, sodass ich einfach erkennen kann, welche Unterschiede die Metadaten aufweisen.

4. Priorität

User Story 3.4: HTML Vergleich

Als Benutzer will ich eine Gegenüberstellung der HTML Daten eines Steps anzeigen können, sodass ich einfach erkennen kann, welche Unterschiede die HTML Daten aufweisen.

4. Priorität

User Story 3.5: Gelöschte Steps darstellen

Als Benutzer will im Vergleichsmodus die Detailansicht eines gelöschten Steps sehen können, sodass ich während dem durchklicken durch die Steps bemerke, dass ein Step im Vergleich zum anderen Build fehlt

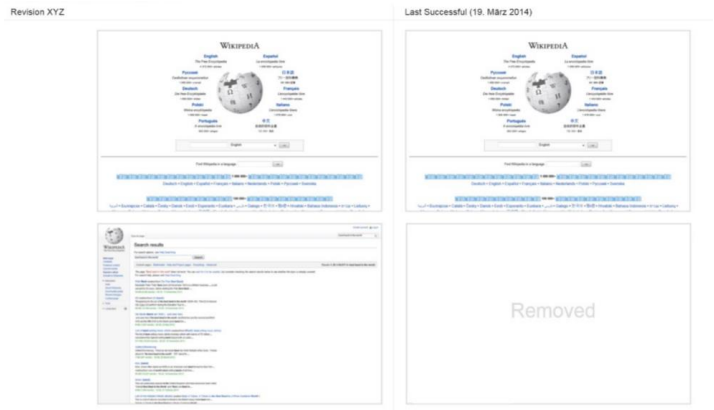
4. Priorität

Epic 4: Gesamtübersichten anzeigen

Als Benutzer will ich weitere Übersichten, sodass ich nicht jeden Step einzeln vergleichen muss, sondern auf einer höheren Ebene schon unterschiede erkennen kann.

Akzeptanzkriterien

- Es gibt eine Ansicht, bei der die Screenshots aller Steps eines Szenarios denen des gleichen Szenarios eines anderen Builds gegenübergestellt werden
- Es gibt eine Ansicht in der ich sehe ob sich Screenshots, Actions oder Metadaten geändert haben. Diese Ansicht will ich beliebig Sortieren oder Filtern können

User Story 4.1: Szenario Steps Gegenüberstellung
Als Benutzer will ich eine Gegenüberstellung aller Steps eines Szenarios anzeigen können, sodass ich sofort erkennen kann, ob sich im Ablauf etwas geändert hat oder die Screenshots Unterschiede aufweisen.

2. Priorität

User Story 4.2: Gesamtübersicht Änderungen
Als Benutzer will ich eine sortierbare und filterbare Ansicht in Tabellenform mit Informationen, ob sich Screenshots, Screen Annotations oder Metadaten geändert haben, sodass ich die Steps mit Unterschieden sofort erkenne und effizient durchsehen kann.

In dieser Form soll die User Story nicht umgesetzt werden. Denn für zu grosse Datensätze wäre diese Tabellendarstellung nicht brauchbar. Ein entsprechender Ersatz wird im Epic 5 beschrieben.

User Story 4.3: Punktediagramm für Use Case Änderungen
Als Benutzer will ich eine Darstellung der Use Case mit Änderungsgrad in einem Punktediagramm, sodass ich sofort erkenne wo die grössten Änderungen sind.
3. Priorität

Epic 5: Diff Funktionalität innerhalb Object-Tree Tabs
Als Benutzer will ich die Diff Funktionalität innerhalb der Object-Tree Tabs nutzen können, sodass ich Änderungen Use Case Übergreifend erkennen kann.
Akzeptanzkriterien • Die Stuktur der Object-Tree Tabs enhält Diff-Informationen
4. Priorität

User Story 5.1: Diff Funktionalität innerhalb Pages
Als Benutzer will ich bei der Page Stuktur Diff-Informationen dargestellt haben, sodass ich erkennen kann welche Pages von einer Änderung betroffen sind.
2. Priorität

User Story 5.2: Diff Funktionalität innerhalb Labels
Als Benutzer will ich bei der Label Stuktur Diff-Informationen dargestellt haben, sodass ich erkennen kann welche Labels von einer Änderung betroffen sind.
4. Priorität

User Story 5.3: Diff Funktionalität innerhalb Requirements
Als Benutzer will ich bei der Requirement Stuktur Diff-Informationen dargestellt haben, sodass ich erkennen kann welche Requirements von einer Änderung betroffen sind.
4. Priorität

User Story 5.4: Diff Funktionalität innerhalb Calls
Als Benutzer will ich bei der Call Stuktur Diff-Informationen dargestellt haben, sodass ich erkennen kann welche Calls von einer Änderung betroffen sind.
4. Priorität.

User Story 5.5: Diff Funktionalität innerhalb UI Elements
Als Benutzer will ich bei der UI Elements Stuktur Diff-Informationen dargestellt haben, sodass ich erkennen kann welche UI Elements von einer Änderung betroffen sind.
4. Priorität

Weitere Use Cases ohne Epic

User Story X.1: PDF Reports erstellen
Als Benutzer will ich ein PDF generieren können, welches die Unterschiede zwischen zwei Builds aufzeigt, sodass ich dadurch eine Dokumentation erhalte, die aufzeigt was sich zwischen zwei Releases geändert hat.
4. Priorität

User Story X.2: Review Funktionalität
Als Benutzer will kennzeichnen können dass ich eine bestimmte Änderung angeschaut habe, sodass im Team sofort ersichtlich ist das für diese Änderung bereits ein Review durchgeführt wurde
4. Priorität

3.2 Nicht-funktionale Anforderungen

Die nicht-funktionalen Anforderungen wurden grösstenteils der Aufgabenstellung entnommen.

3.2.1 Funktionalität

NFA1	**Richtigkeit**	Die Vergleiche zwischen zwei Builds sollen korrekt sein. Das heisst, es werden stets die richtigen zwei Steps miteinander verglichen, soweit es mit dem aktuell vorhandenen Identifier möglich ist.
NFA2	**Interoperabilität**	Die neue Funktionalität soll in die bestehende Scenarioo Lösung integriert werden. Folgende Technologien sollen dabei zum Einsatz kommen: • AngularJS • JavaScript • HTML5 • Java 6 • Java REST Server Die Lizenz von neu eingesetzten Frameworks muss mit der Lizenz vom Open Source Projekt Scenarioo kompatibel sein. Die bisherige Grundfunktionalität von Scenarioo soll auch ohne die Installation von Zusatzkomponenten weiterhin funktionieren.
NFA3	**Sicherheit**	Die Sicherheit der bestehenden Applikation soll durch die neue Funktionalität nicht beeinträchtigt werden. Aktuell gibt es im Scenarioo noch keine logische Sicherheitseinschränkungen.

3.2.2 Zuverlässigkeit

NFA4	**Reife**	Die neue Diff Viewer Funktionalität wird automatisiert und manuell getestet, um mögliche Fehler aufzudecken. Bei Testläufen mit Kundendaten sollen keine Fehler auftauchen.
NFA5	**Fehlertoleranz**	Alle neuen Eingabemöglichkeiten werden so gestaltet, dass der Benutzer keine Fehleingaben machen kann.

3.2.3 Benutzbarkeit

NFA6	**Bedienbarkeit**	Die neue Erweiterung soll für jede Benutzergruppe bedienbar sein. Um dies zu erreichen, werden Usability Tests durchgeführt.
NFA7	**Attraktivität**	Das neue Feature soll an das bestehende Scenarioo Design anknüpfen. Die Diff Viewer Resultate sollen übersichtlich dargestellt werden.
NFA8	**Verständlichkeit**	Die dargestellten Unterschiede sollen verständlich sein.

3.2.4 Effizienz

NFA9	**Zeitverhalten**	Es sollen möglichst effiziente Algorithmen zum Struktur- und Bildvergleich gefunden und eingesetzt werden. Dabei kann es sich um eine Third Party Library oder um eine Eigenentwicklung handeln. Die Berechnung der Vergleichsdaten darf Zeit kosten, sollte aber linear zu den Anzahl Steps sein. Das Laden einer Seite darf nicht merklich länger dauern als ohne Diff-Daten.

3.2.5 Wartbarkeit

NFA10	**Analysierbarkeit**	Es sollen wichtige Informationen in die Logfiles geschrieben werden.
NFA11	**Modifizierbarkeit**	Das neue Feature soll so gebaut werden, dass ohne grössere Umbauten weitere Vergleiche eingebaut werden können.
NFA12	**Stabilität**	Die neue Diff Viewer Funktionalität soll mit Unit Tests abgedeckt werden.
NFA13	**Konformität**	Die Entwicklerkonventionen vom Scenarioo Wiki müssen eingehalten werden. https://github.com/scenarioo/scenarioo/wiki/Coding-guidelines Vergleichsdaten sollen in einer separaten Dateistruktur abgelegt werden. Vergleichsdaten sollen im XML-Format abgespeichert werden.
NFA14	**Dokumentation**	Die Diff Viewer Funktionalität sowie deren Verwendung soll im Github Wiki dokumentiert sein.

3.2.6 Übertragbarkeit

NFA15	**Installierbarkeit**	Das neue Feature soll ohne grossen Aufwand in das bestehende Scenarioo Source Repository eingefügt werden können. Die Installation von Zusatzkomponenten soll Dokumentiert werden.

4 Barrierefreiheit

Nützliche Informationen schön gestaltet zu präsentieren reicht bei weitem nicht aus. Die ansprechendste und informativste Webseite nützt einem nichts, wenn diese nicht bedienbar ist. Deshalb rückt das Thema Barrierefreiheit bei der Entwicklung von Software den Menschen ins Zentrum.

Besonders im Webbereich gibt es eine grosse Breite an Endnutzern. Darunter gibt es einige Menschen, welche mit Einschränkungen wie zum Beispiel Farbsehschwäche, Blindheit oder motorischen Störungen zu kämpfen haben. Es gibt aber auch sogenannte Poweruser, welche sich schon sehr gut mit der Anwendung auskennen und ihre Ziele in möglichst wenigen Schritten erreichen möchten.

Das Thema Barrierefreiheit hat das Ziel, Applikationen für alle Benutzergruppen zugänglich zu machen. So wird niemand ausgeschlossen und alle können vom angebotenen Service profitieren. Dies bedeutet natürlich immer auch einen gewissen Mehraufwand im Projekt.

Bei der Entwicklung einer Webseite müssen gemäss der Stiftung „Zugang für alle“ insbesondere die nachfolgend aufgelisteten Punkte beachtet werden.

Tabelle 3: Zehn Punkte für eine barrierefreie Webseite

Accessibility ist Chefsache	Die Zugänglichkeit der Website liegt nicht nur in der Verantwortung der Entwicklungsabteilung, sondern gehört in jede Internetstrategie.
Accessibility ist integraler Bestandteil des Projekts	Barrierefreiheit soll von Beginn an in einem Projekt berücksichtigt werden. So kann Accessibility mit geringen Mehrkosten umgesetzt werden. Wird Accessibility erst am Schluss eines Projektes berücksichtigt, so ist der Aufwand für die Nachbesserungen deutlich grösser.
Design for All	Das Prinzip „Design for all“ stellt den User in den Mittelpunkt. Unabhängig von Einschränkungen soll eine Website entwickelt werden, welche von der grösstmöglichen Anzahl Benutzer sinnvoll gebraucht werden kann. Textversionen für blinde Anwender oder ähnliche Alternativen sollten in jedem Fall vermieden werden.
Trennung zwischen strukturiertem Inhalt und Layout	HTML ist keine Programmier- oder Seitenbeschreibungssprache! Eine strikte Trennung von strukturiertem Inhalt (HTML) und dem Layout (CSS) ist der Grundstein für jede barrierefreie Website.
Richtlinien	Die internationalen Richtlinien des W3C, die WCAG 1.0, sind konform zur Schweizer Gesetzgebung und eignen sich für alle Bereiche der öffentlichen Hand sowie für private Unternehmen.
Komplexität	Die Komplexität sollte immer dem Inhalt angemessen und keinesfalls unnötig hoch sein.

Geräteunabhängigkeit	Die Geräteunabhängigkeit ermöglicht unterschiedlichen assistierenden Technologien wie beispielsweise einem Screen Reader, PDA oder Mobiltelefon den Zugriff auf die Inhalte der Website.
Tests	Ob sich eine Website auch für Menschen mit Behinderungen eignet, kann am besten mit Betroffenen selbst überprüft werden. Denn der grösste Teil der Barrieren kann nur von Menschen mit Behinderungen und dem Einsatz assistierender Technologien erkannt werden.
Schulung	Accessibility wird zu einem grossen Teil durch Systeme (Content Management System) und Layoutvorlagen beeinflusst. Ein Teil der Accessibility wird aber immer durch die Autoren und Redaktoren bestimmt. Sie müssen geeignet geschult werden, damit sie die Anforderungen an eine barrierefreie Website sinnvoll umsetzen können.
Nachhaltigkeit	Accessibility wird nicht einmal entwickelt und bleibt danach konstant. Es ist immer darauf zu achten, dass neue Inhalte oder neue Erweiterungen ebenfalls zugänglich erstellt werden.

(10 Punkte für eine barrierefreie Website 2016)

Leider wurde das Thema Barrierefreiheit bei der initialen Entwicklung von Scenarioo zu wenig gewichtet. Dennoch wollten wir bei unserer Arbeit einige Punkte aufgreifen. Ein immer wieder diskutierter Punkt war die Farbsehschwäche von Nutzern. Wir haben deshalb die von uns neu entworfenen Webseitenelemente, wie zum Beispiel die Diff Viewer Icons, so gestaltet, dass sie einen genügend hohen Kontrast zwischen Vorder- und Hintergrundfarbe aufweisen. Zusätzlich haben wir darauf geachtet, dass die grafischen Informationen, wo möglich, auch in schriftlicher Form verfügbar sind.

5 Designkonzept

5.1 Vorgehen

Nachdem wir die Anforderungen der Benutzer in einer ersten Version erfasst hatten, haben wir einen klickbaren UI-Prototyp entworfen. Auf Grund der nicht-funktionalen Anforderungen der Benutzbarkeit, haben wir uns dazu entschieden, möglichst viele bestehende UI-Elemente wiederzuverwenden. So wollen wir erreichen, dass das neue Feature harmonisch in das bestehende Design einfliesst und nicht störend wirkt.

Der UI-Prototyp wurde in der Scenarioo-Community und vom Projektteam intensiv diskutiert. Aus den Diskussionen entstanden weitere Designvorschläge und auch neue User Stories.

Die neuen Designvorschläge wurden umgesetzt und in einem Usability Test geprüft. Dieser Test gab uns wichtige Erkenntnisse darüber, ob wir die oben angesprochene nichtfunktionale Anforderung der Benutzbarkeit erfüllen können. Die gewonnenen Resultate wurden wiederum in das Design eingearbeitet.

In den nachfolgenden Abschnitten wird die Entwicklung vom grafischen Design vom UI-Prototyp bis hin zum finalen Design dokumentiert werden.

5.2 UI-Prototyp

5.2.1 Auswahl Vergleichsbuild

Abbildung 15: Entwurf Auswahl vom Vergleichsbuild

Im Headerbereich soll ein neues Dropdown Menu eingebaut werden. Dieses listet alle zum aktuell ausgewählten Build verfügbaren Vergleichsbuilds auf.

5.2.2 Übersicht Use Cases

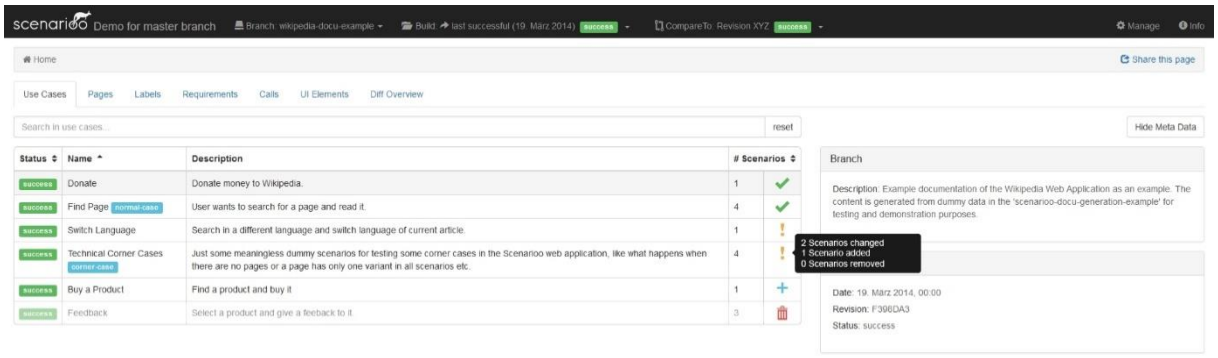


Abbildung 16: Entwurf Übersicht Use Cases

In die bestehende Übersichtstabelle der Use Cases wird neu eine separate Spalte mit Diff-Informationen eingebaut. Anhand von Icons soll ersichtlich sein, ob ein Use Case neu, gelöscht, modifiziert oder unverändert ist.

Fährt man mit der Maus über das Ausrufezeichen, so werden detailliertere Diff-Informationen in einem Tooltip dargestellt. Darin ist ersichtlich, wie viele Szenarien vom entsprechenden Use Case hinzugefügt, gelöscht oder modifiziert wurden.

Use Cases, welche im Vergleichsbuild noch vorhanden waren, im aktuellen Build aber nicht mehr existieren, werden leicht ausgegraut dargestellt.

5.2.3 Übersicht Scenarios

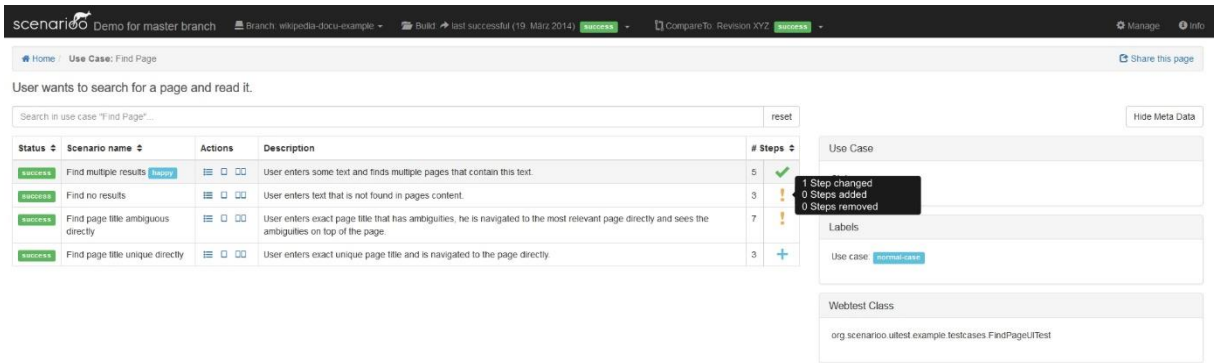


Abbildung 17: Entwurf Übersicht Scenarios

Die Übersichtsseite für Szenarien ist im Grundsatz genau gleich aufgebaut wie jene für Use Cases.

Neu hat der Benutzer auf dieser Seite die Möglichkeit, eine Ansicht mit der Gegenüberstellung von Steps aufzurufen. Dazu haben wir ein weiteres Icon ( ) in die Spalte Actions eingebaut.

Wie die Seite mit der Gegenüberstellung der Steps aussieht, kann im entsprechenden Abschnitt weiter unten entnommen werden.

5.2.4 Übersicht Steps

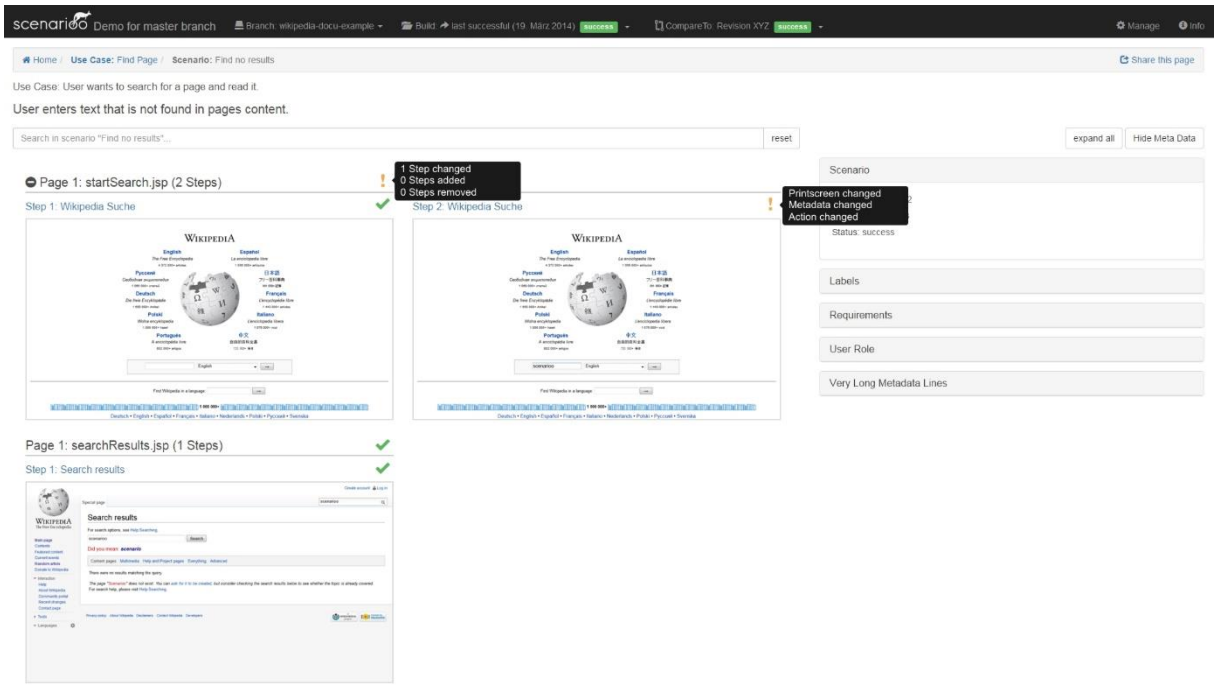


Abbildung 18: Entwurf Übersicht Steps

Auf dieser Seite werden alle Schritte von einem Szenario gruppiert in Pages dargestellt. Die Pages können aufgeklappt werden, so dass sämtliche Schritte dargestellt werden.

Auch hier kommt wieder das Konzept mit den Icons und den Tooltips zum Zuge. Es soll für den Benutzer ersichtlich sein, ob sich etwas auf einer Page oder einem Step geändert hat.

Fährt man mit der Maus über ein Icon oberhalb von einem Step, so werden dem Benutzer die Diff-Informationen vom Step dargestellt. Dies können sein: prozentuale Änderung am Screenshot, Änderungen an Actions (Annotationen), Änderungen an Metadaten, und so weiter.

5.2.5 Stepansicht

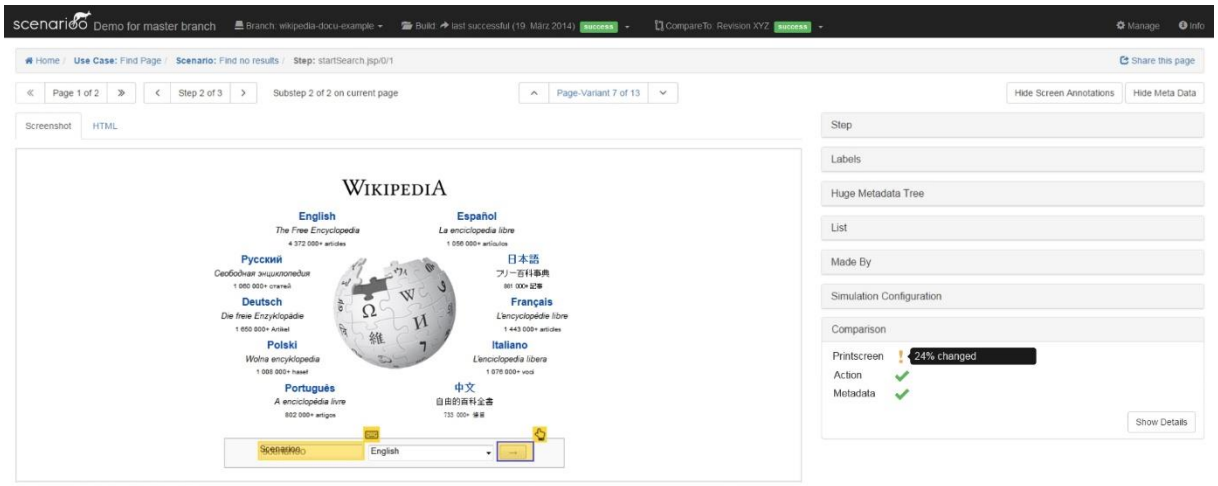


Abbildung 19: Entwurf Stepansicht

Auf der rechten Seite der Stepansicht gibt es neu eine Comparison-Box, welche die Diff-Informationen von einem Step darstellt.

Mit dem Button in der Comparison-Box kann auf die Stepdetailansicht gesprungen werden.

5.2.6 Stepdetailansicht

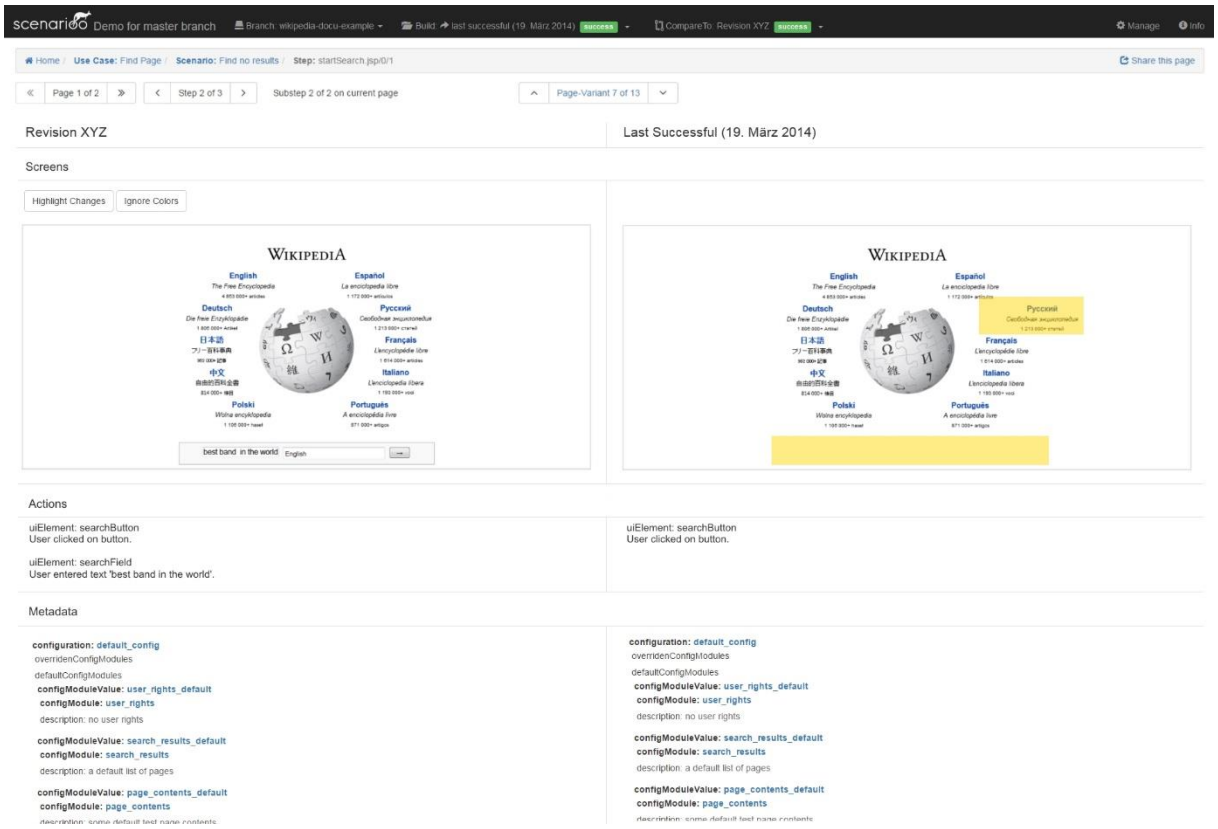


Abbildung 20: Entwurf Stepdetailansicht

Die Stepdetailansicht ist eine komplett neue Seite. Sie erhält dieselben Navigationselemente wie die Stepansicht.

Der Benutzer sieht auf dieser Seite eine Gegenüberstellung aller Änderungen von einem Step zweier Builds. Im oberen Bereich werden die Screenshots verglichen und allfällige Änderungen werden farblich hervorgehoben. Im unteren Teil gibt es eine Gegenüberstellung von Actions (Annotations) und Metadaten. In einer ersten Version muss der Benutzer von Auge vergleichen, was sich geändert hat.

5.2.7 Gegenüberstellung Steps

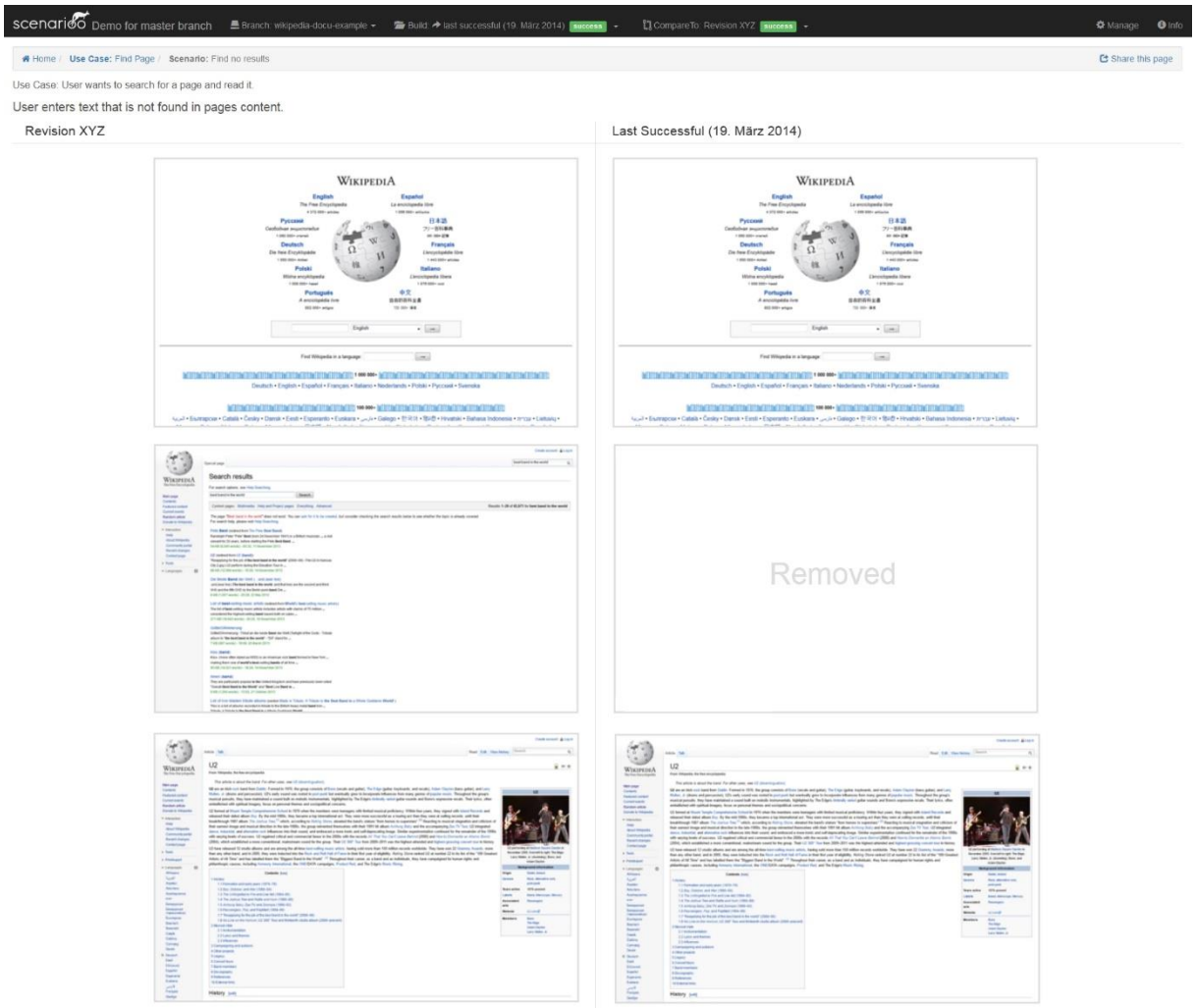


Abbildung 21: Entwurf Gegenüberstellung der Steps

Die Gegenüberstellung der Steps ist eine komplett neue Seite. Sie kann von der Szenarioübersicht über das entsprechende Action-Icon aufgerufen werden.

Auf dieser Seite werden alle Steps eines Szenarios von zwei Builds gegenübergestellt. Dies ermöglicht dem Benutzer einen schnellen Überblick über alle Änderungen in einem Szenario zu gewinnen.

Schritte, welche in einem der beiden Steps nicht mehr vorhanden sind, werden als Platzhalter mit den Texten „Removed“ oder „Added “ in die Auflistung eingefügt.

Bei einem Klick auf einen Screenshot öffnet sich die entsprechende Stepansicht.

5.2.8 Übersicht Diff-Informationen

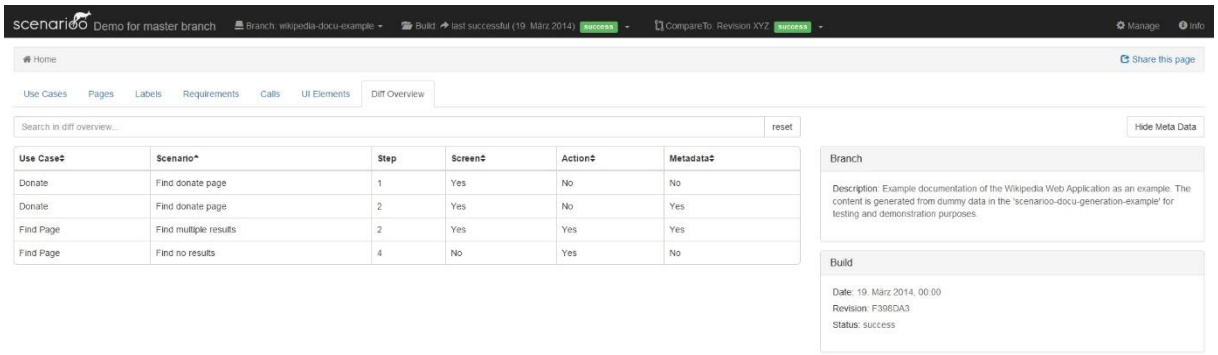


Abbildung 22: Entwurf Übersicht Diff-Informationen

Diese Übersichtsseite enthält alle in einem Buildvergleich erkannten Änderungen und stellt diese in Form einer Tabelle dar. Die Diff-Informationsseite soll in einem eigenen Tab auf dem Startscreen eingebettet werden.

Der Benutzer kann die Tabelle beliebig nach einem Use Case oder Szenario filtern. Ebenso erhält er die Möglichkeit, die Tabelle nach seinen Ansprüchen zu sortieren. Dies erlaubt ihm, schnell die gewünschten Änderungen einzusehen.

5.3 Anpassungen nach Besprechung

Den oben gezeigten Designentwurf haben wir mit Zühlke besprochen. In diesem Kapitel werden die Resultate dieser Besprechung aufgezeigt.

5.3.1 Icons überarbeiten

5.3.1.1 Ansatz 1: Icons ändern und farblich hervorheben

5.3.1.1.1 Icons ändern

Die Wahl der Icons war ein zentraler Punkt der Besprechung. Es wurden dazu mehrere Entwürfe erstellt. Ein erster Ansatz war, die Icons vom UI-Prototyp beizubehalten und das Kübel Icon durch ein Minus Icon zu ersetzen. Ebenso gab es eine Änderung in der Farbgebung.

Tabelle 4: Überarbeitete Icons

Icon	Beschreibung
	Steht für ein unverändertes Item
	Steht für ein geändertes Item
	Steht für ein hinzugefügtes Item
	Steht für ein gelöschtes Item

5.3.1.1.2 Änderungsgrad farblich hervorheben

Im ursprünglichen Designentwurf sieht man zwar, dass sich ein Use Case, Szenario oder Step geändert hat, allerdings nicht um wie viel Prozent. Wir möchten dem mit der Einführung einer Änderungsrate entgegenwirken. Die Änderungsrate soll, wo immer Diff-Informationen angezeigt werden, signalisieren, wie fest sich etwas verändert hat.

Anhand der Änderungsrate wird das entsprechende Icon oder HTML-Element farblich abgestuft. Es sind folgende Stufen eingeplant:

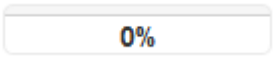
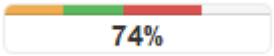
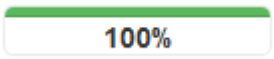
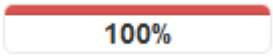
Tabelle 5: Änderungsgrad und entsprechende Farbtöne

Änderung	Beschreibung
0-20%	Geringe Änderung
20-40%	Mässige Änderung
40-60%	Mittlere Änderung
60-80%	Erhebliche Änderung
80-100%	Grosse Änderung

5.3.1.2 Ansatz 2: Progress Bar ähnliches Diff-Icon

Nach einer weiteren Absprache mit Zühlke und deren Designern kamen wir zum Entschluss, dass Icons, die nur aus einem Symbol bestehen, zu wenig Informationen liefern. Der neu ausgearbeitete Entwurf sieht einem Progress Bar ähnlich und zeigt Benutzer den exakten Änderungswert. Dieses Design erlaubt dem Benutzer schnell einen schnellen Überblick über die Art der Änderung zu gewinnen.

Tabelle 6: Progress Bar Diff-Icons

	Dieses Element enthält keine Änderungen.
	Dieses Element hat sich insgesamt um 74% geändert. Die Farben orange, grün und rot signalisieren dem Benutzer, dass dieses Element geänderte (orange), hinzugefügte (grün) und gelöschte (rot) Kind-Elemente enthält.
	Dieses Element wurde hinzugefügt
	Dieses Element wurde gelöscht

5.3.2 Seiten in Tabs

Die Seite „Gegenüberstellung Steps“ und die Seite „Übersicht Steps“ sollen beide in je einen Tab ausgelagert werden. So kann man einfach zwischen der Gegenüberstellung und der normalen Auflistung wechseln. Aus zeitlichen Gründen haben wir die Seite „Gegenüberstellung Steps“ nicht implementiert und so bleiben hier die Tabs vorerst auch weg.

Auch die Seite „Stepdetailansicht“ wird neu in einem Tab auf der Seite „Stepansicht“ platziert. Der Tab befindet sich rechts von den bestehenden Tabs Screenshot und HTML. Damit der Benutzer erkennt, ob sich ein Sprung in den neuen Tab lohnt, wird im Tab die Änderungsrate dargestellt.

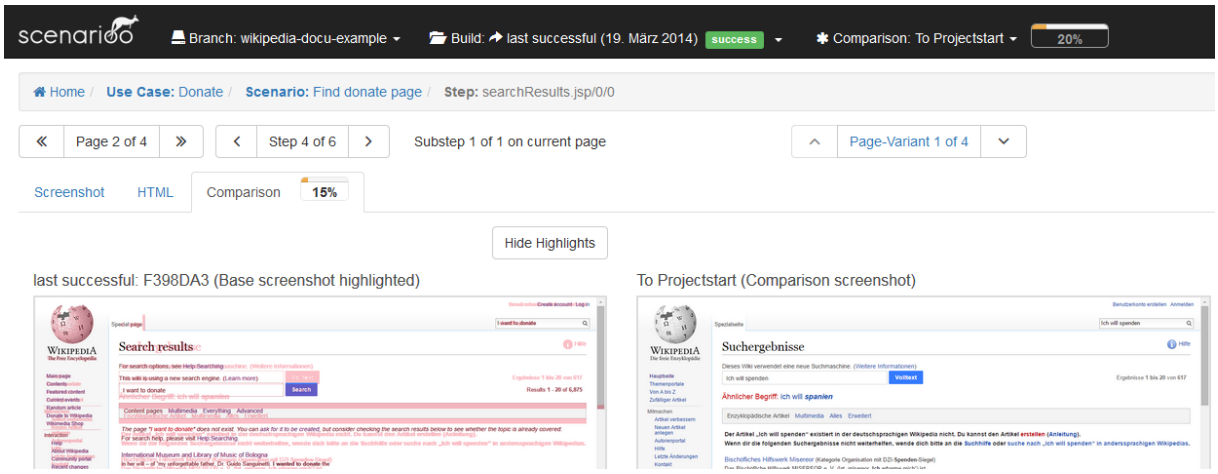


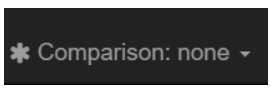
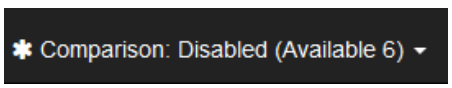
Abbildung 23: Stepdetailansicht im Wikipedia Beispiel

5.3.3 Übersicht Diff-Informationen

Die Seite „Übersicht Diff-Informationen“ wird aus verschiedenen Gründen so nicht umgesetzt. Bei den meisten Projekten wo Scenarioo eingesetzt wird wäre die Anzahl Zeilen viel zu hoch, sodass die Übersicht enorm darunter leiden würde. Ausserdem werden in einem ersten Schritt die Metadaten und Actions noch nicht verglichen.

5.4 Anpassungen nach Usability Tests

Bei den Usability Tests konnten wir diverse Verbesserungsmöglichkeiten zugunsten der Benutzerfreundlichkeit ausarbeiten. Über die Durchführung der Usability Tests haben wir in einem anderen Kapitel berichtet. In diesem Kapitel wird erläutert, welche Probleme erkannt wurden und was für Auswirkungen eine entsprechende Lösung auf das Design hat.

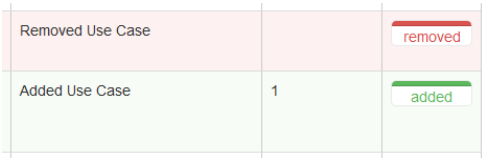
Erkenntnis	**Comparison Menu wird übersehen**
Beschreibung	Sämtliche Testbenutzer übersehen das Comparison Menu, wenn sie zum ersten Mal einen Vergleich starten wollen. Im Nachinterview betonen alle, es sei am richtigen Ort platziert, doch zu Beginn haben sie es schlichtweg nicht beachtet.
Lösungsvorschlag	Ist kein Vergleich ausgewählt soll es nicht „Comparison: none“ heissen, sondern „Comparison: Disabled (Available n). Ausserdem soll auf der Startseite ein zusätzliches Comparison Panel entstehen, welches alle möglichen Vergleiche auflistet. Als weitere Optimierung wird eine hellere Schriftfarbe verwendet.
Vergleich nach Umsetzung	Stand Usability Test 	Optimiert 

Erkenntnis	**Neue/Gelöschte Elemente werden nicht sofort erkannt**
Beschreibung	Den Testbenutzern ist nicht sofort klar, dass eine 100% Änderungsrate und ein grüner Balken ein neues Element ist. Gelöschte Elemente erkennen sie in kurzer Zeit aufgrund der grauen Schattierung.
Lösungsvorschlag	Anstelle von „100%“ die Texte „added“ oder „removed“ in entsprechender Farbe verwenden. Eventuell mit einem Icon versehen um sie noch deutlicher hervorzuheben. Gelöschte Elemente sollen einen schwachen roten Hintergrund erhalten. Hinzugefügte Elemente sollen einen schwachen grünen Hintergrund erhalten.

Vergleich nach Umsetzung

Stand Usability Test

Optimiert

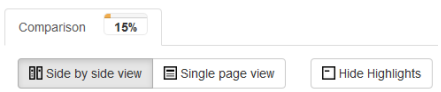


Erkenntnis	**Diff Screenshot Toggle Button wird nicht verwendet**
Beschreibung	Von keiner der Testpersonen wurde die Funktion genutzt, den Diff Screenshot auszublenden. Gemäss Interviewaussagen wurde das Ausblenden als unnötig befunden oder der Button wurde übersehen.
Lösungsvorschlag	Es soll eine Side-by-Side Ansicht sowie eine Single-Page-View entstehen um die Funktionalität zu erweitern. Ausserdem wird die Beschriftung des Buttons überarbeitet und mit einem Icon versehen. Die neue Buttonleiste wird neu oberhalb der Screenshot Beschriftung sein um sie klarer abzugrenzen.

Vergleich nach Umsetzung

Stand Usability Test

Optimiert

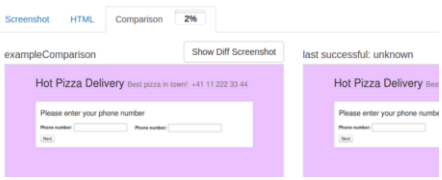


Erkenntnis	**Anordnung der Diff-Screenshots unklar**
Beschreibung	Teilweise waren die Testpersonen verunsichert weil die Reihenfolge der Diff Screenshots umgekehrt zu der Reihenfolge im Auswahlmenu der comparison Builds ist. Denn im Auswahlmenu liegt der Vergleich auf der rechten Seite, bei den Screenshots ist der Vergleich auf der linken Seite.

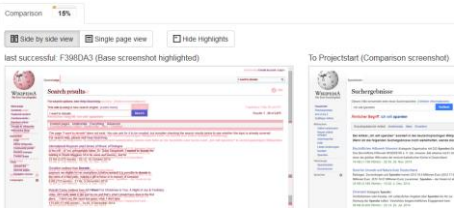
Lösungs-vorschlag	Diff Screenshots analog zum Auswahlmenu anordnen. Zusätzlich beschriften, welcher Screenshot zum Base Build gehört und welcher zum Comparison Build.

Vergleich nach Umsetzung

Stand Usability Test



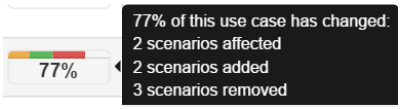
Optimiert



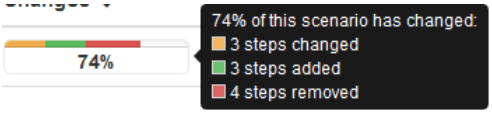
Erkenntnis	**Bedeutung der Farben im Diff Icon unklar**
Beschreibung	Den Testpersonen war nicht eindeutig klar, was die Farben in den Diff Icons bedeuten.
Lösungs-vorschlag	Die textuelle Beschreibung innerhalb der Tooltipps entsprechend einfärben. Eine Farblegende soll erstellt werden.

Vergleich nach Umsetzung

Stand Usability Test



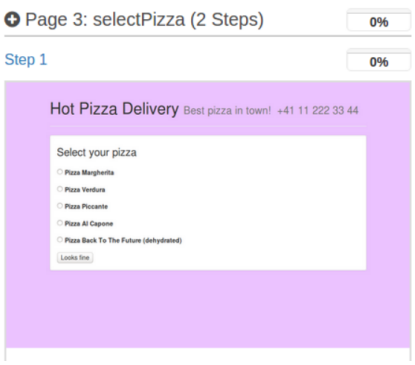
Optimiert



Erkenntnis	**Mehrspaltiges Layout in der Szenario-Übersicht unverständlich**
Beschreibung	Die Testpersonen haben nicht verstanden, dass sie Pages, die mehrere Steps enthalten, aufklappen können.
Lösungsvorschlag	Das zweispaltige Layout wird in ein einspaltiges Layout umgewandelt um die unterschiedlichen Seiten klarer voneinander abzugrenzen.

Vergleich nach Umsetzung

Stand Usability Test



Optimiert

Dieses Problem betrifft nicht direkt unsere Erweiterung und wird zu einem späteren Zeitpunkt von Zühlke selbst optimiert.

6 Analyse Domain Modell

Das nachfolgende Domain-Modell zeigt die Diff Viewer Erweiterung innerhalb vom Scenarioo Projekt. Blau eingefärbt sind jene Entities, welche bereits schon vorhanden sind und gelb sind jene, welche durch das Diff-Feature neu hinzukommen.

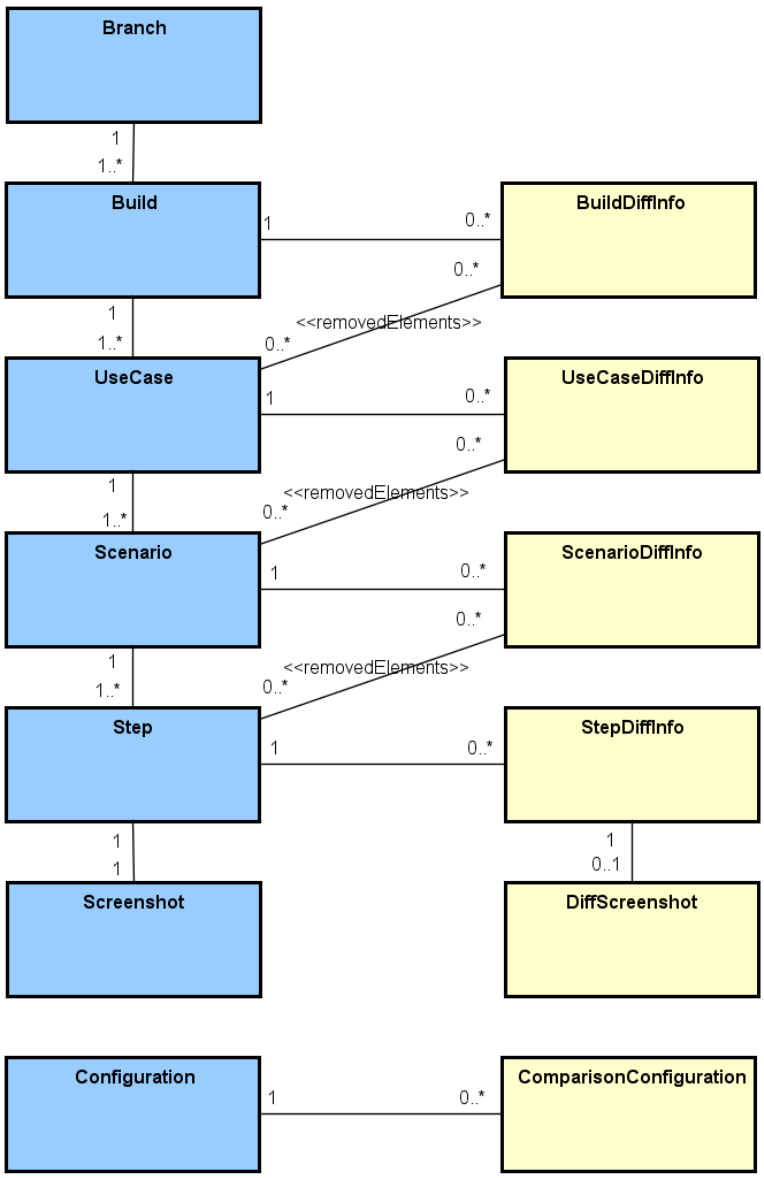


Abbildung 24: Domain Modell Scenarioo

7 Design

7.1 Architektur

In den nachfolgenden Kapiteln wird die bestehende Architektur von Scenarioo dokumentiert. Diese hat sich durch den Diff Viewer nicht verändert.

7.1.1 Ziele und Einschränkungen

Die in der nachfolgenden Systemübersicht aufgezeigte Architektur stand zu Projektbeginn bereits fest. Grundlegende Änderungen wie zum Beispiel die Einführung einer Datenbank wären zwar möglich, müssten aber gut begründet und abgesprochen sein.

Hinsichtlich der nicht-funktionalen Anforderung der Interoperabilität ist es unser Ziel, das neue Feature so gut es geht in die bestehende Architektur einzufügen. Folglich werden wir unsere Lösungsansätze in erster Linie mit den bestehenden Technologien entwerfen.

7.1.2 Systemübersicht

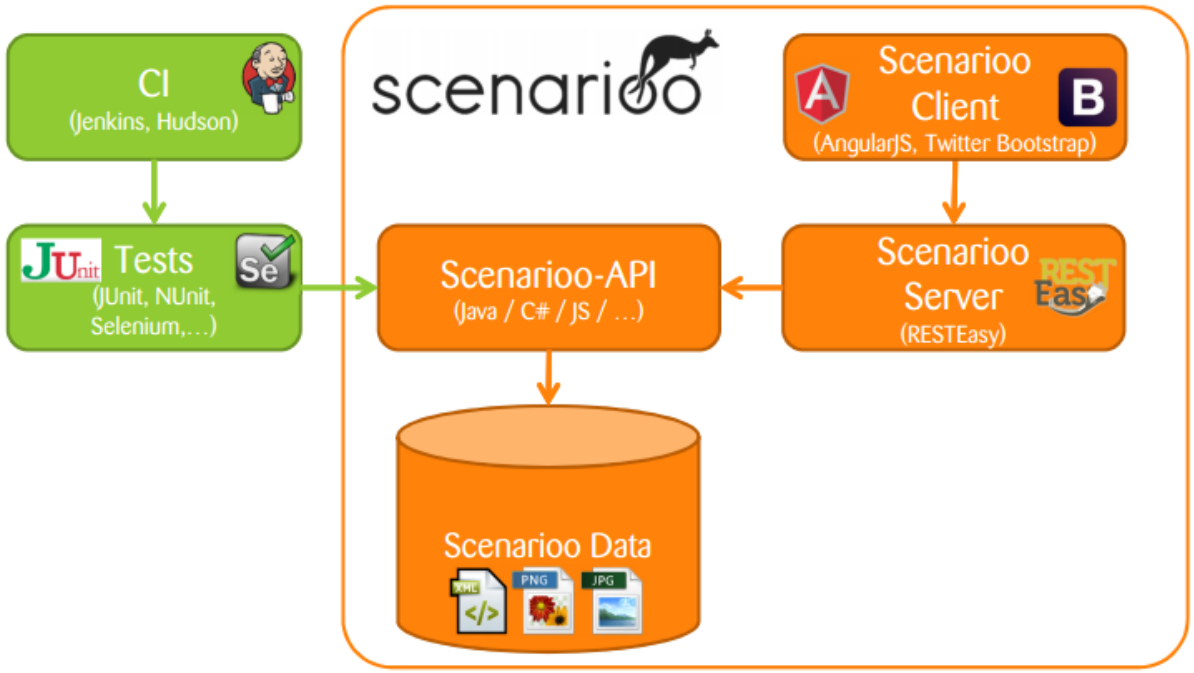


Abbildung 25: Scenarioo Systemübersicht (Rolf Bruderer, Zühlke 2014)

Die Scenarioo-Systemlandschaft besteht aus mehreren Applikationen mit unterschiedlichen Aufgaben.

Im Idealfall führt ein Server mittels Continuous Integration jede Nacht die Unit Tests aus. Die Testfälle generieren zusammen mit der entsprechenden Scenarioo-API die XML-Dateien mit den Informationen über Use Cases, Szenarien und co. Ebenso werden bei der Testausführung Screenshots von den einzelnen Schritten im Filesystem abgelegt.

Die so abgespeicherten Informationen können dann mit dem Scenarioo-Client eingesehen werden. Dieser holt sich die Informationen über eine REST-Schnittstelle vom Scenarioo-Server.

7.1.3 Prozesse und Threads

Der aktuelle Import von neuen oder bestehenden Build-Informationen läuft in einem separaten Thread ab.

Der Buildvergleich wird ebenfalls in einem separaten Thread ausgeführt, da dieser durch seine andauernde Laufzeit sonst blockierend wirken würde.

7.1.4 Deployment Diagramm vom Scenarioo-Viewer

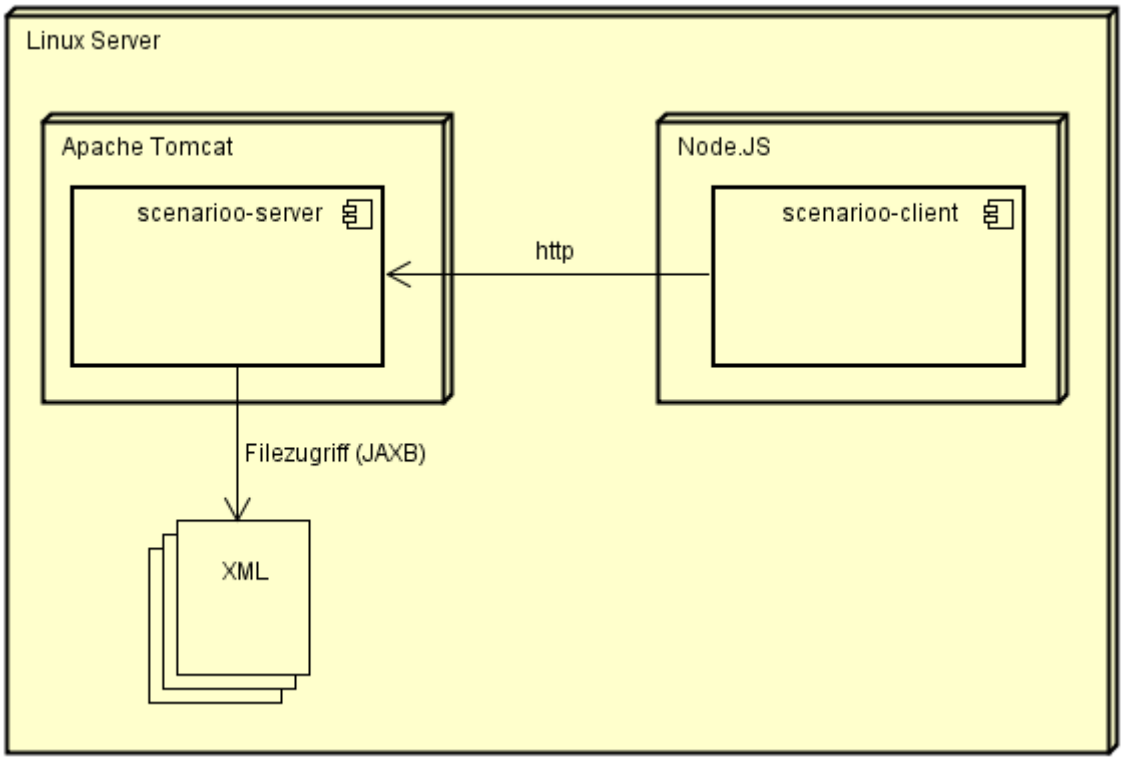


Abbildung 26: Deployment Diagramm Scenarioo vom Scenarioo-Viewer

Die Scenarioo-Viewer Applikation wird vorzugsweise auf einer Linux-Machine eingerichtet. Darauf sollte mindestens Java 6 installiert sein.

Die XML-Dateien mit den Build-Informationen werden direkt auf dem Filesystem abgelegt. Da die Scenarioo-Server Applikation auf diese Files zugreift, muss derjenige User, welcher die Scenarioo-Server Applikation ausführt, zwingend Lese- und Schreibrecht auf dem Verzeichnis mit den XML-Dateien haben. Das Mapping zwischen den XML-Dateien und den Java Objekten wird mittels JAXB gehandhabt.

Die Scenarioo-Server Applikation wird auf einem Apache Tomcat ausgeführt und die Scenarioo-Client Applikation wird von einem Node.JS Server ausgeliefert.

7.1.5 Component Diagramm

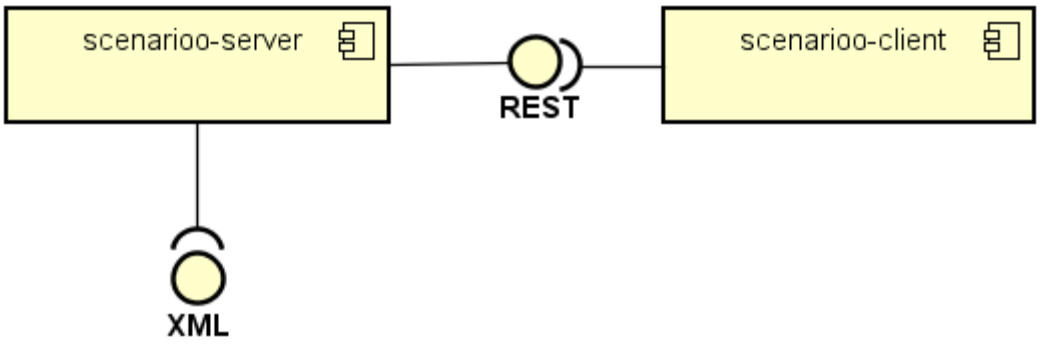


Abbildung 27: Component Diagramm Scenarioo-Viewer

Die beiden Komponenten der Scenarioo-Viewer Applikation kommunizieren über eine REST-Schnittstelle. Alle vom Client benötigten Informationen werden über diese Schnittstelle vom Server abgefragt.

Damit der Server dem Client diese Informationen überhaupt zur Verfügung stellen kann, ist er auf die XML-Schnittstelle auf dem Server angewiesen.

7.1.5.1 REST API

Der Scenarioo-Server enthält eine auf JAX-RS basierende REST API. Diese API Schnittstelle lassen wir unverändert und bauen parallel eine API für den Abruf von Diff Informationen auf. Eine ausführliche Dokumentation der möglichen Diff Viewer REST Aufrufe befindet sich im Scenarioo Wiki https://github.com/magitnu/scenarioo/wiki/Diff-Viewer-REST--API und im Anhang dieses Dokumentes.

7.1.5.2 XML

Die XML-Dateien müssen in einer bestimmten Ordnerstruktur abgelegt werden. Für das Erstellen dieser Struktur und der XML-Dateien lohnt es sich, eine Scenarioo-Writer Library in der entsprechenden Programmiersprache zu verwenden.

Eine detaillierte Dokumentation der Ordnerstruktur kann dem Scenarioo Wiki auf Github entnommen werden.

https://github.com/scenarioo/scenarioo/wiki/Scenarioo-Writer-Documentation-Format

7.1.6 Dokumentationsmodell

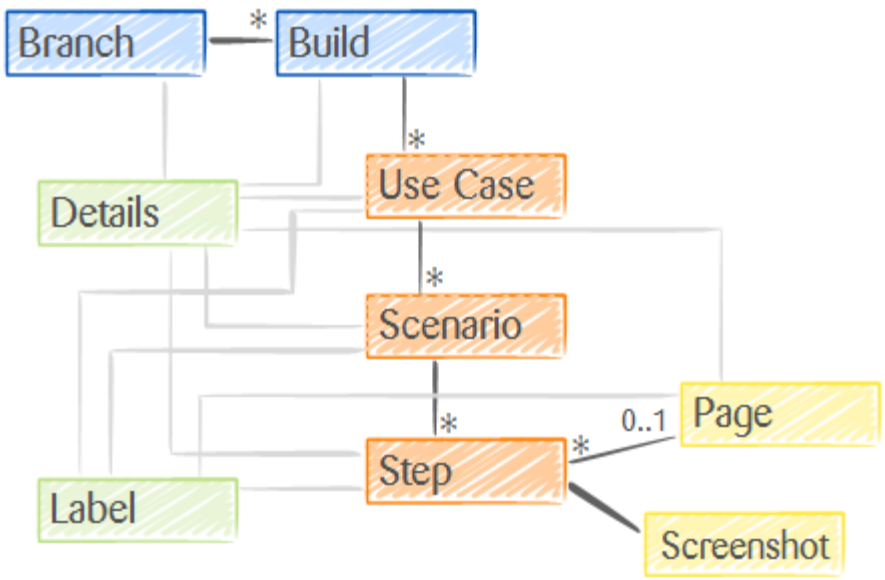


Abbildung 28: Dokumentationsmodell (Rolf Bruderer, Zühlke 2015)

7.1.6.1 Branch

Verschiedene Produktversionen, Releases oder gar Featurebranches können so separat dokumentiert werden.

7.1.6.2 Build

In jedem Branch können mehrere Dokumentationsversionen abgelegt werden (typischerweise jede Nacht). Jeder Build hat eine Version und einen Zeitstempel.

7.1.6.3 Use Case

Die getesteten und dokumentierten Use Cases.

7.1.6.4 Scenario

Die eigentlichen Testfälle zeigen pro Use Case die relevanten Benutzerszenarien.

7.1.6.5 Step

Dies sind die einzelnen Interaktions-Schritte eines Benutzerszenarios.

7.1.6.6 Screenshot

Jeder Step wird mit einem Screenshot dokumentiert.

7.1.6.7 Page

Typischerweise passieren die Interaktions-Schritte auf einer Page (das kann eine Webseite, eine UI-Maske, eine View, etc. sein). Mehrere Schritte auf der gleichen Page werden in Scenarioo zusammen gruppiert dargestellt.

7.1.6.8 Label

Ein Label ist ein angehängtes Schlagwort, damit können die Use Cases, Szenarien und sogar die einzelnen Schritte speziell markiert und kategorisiert werden.

7.1.6.9 Details

Auf jedem Schritt können beliebige Zusatzinformationen abgelegt werden, wie z.B. HTML-Code, Benutzer-Rolle, Test-Konfiguration, Umsystem-Aufrufe, Link zum Sourcecode, Link zu anderen Dokumentationen. Hierfür stehen generische Objekt-Strukturen zur Verfügung. Ebenso können solche Zusatzinformationen auch an anderen Objekten (z.B. am Use Case oder Szenario) in der Dokumentation angehängt werden.

7.2 Designentscheide von Zühlke

Nachfolgend werden jene Entscheide von Zühlke aufgelistet, welche einen Einfluss auf das Design des Diff Viewers haben.

Entscheid	Die Grundfunktionalität von Scenarioo darf durch fehlende Third Party Libraries, wie zum Beispiel ein Bildvergleich Tool, nicht beeinträchtigt werden.
Auswirkung	Im ScreenshotComparator musste ein Fallback eingebaut werden, für den Fall, dass das Bildvergleich Tool GraphicsMagick nicht installiert ist. Bei den End-to-End Tests können keine fixen Vergleichswerte getestet werden, da GraphicsMagick nicht zwingend installiert ist und somit keine Änderungen auf dem Bild erkennt werden können.

Entscheid	Die API der Scenarioo Writer Library darf nicht verändert werden.
Auswirkung	Die bestehenden Entities durften nicht angepasst werden. So musste auf ein zuerst angedachtes ComparableItem Interface verzichtet werden.

Entscheid	Ein eindeutiger Step Identifier darf in dieser Arbeit nicht eingeführt werden. Als Step Identifier dienen die Page und der entsprechende In-Page-Index.
Auswirkung	Ein Step kann innerhalb einer Page nicht eindeutig identifiziert werden. Dies führt dazu, dass falsche Steps miteinander verglichen werden können.

Entscheid	Diff Viewer XML-Daten müssen komplett getrennt von den restlichen Scenarioo Daten gespeichert werden
Auswirkung	Separate Ordnerstruktur mit Diff Viewer XML-Daten.

Entscheid	Auf Parallelisierung soll in dieser Arbeit verzichtet werden. Die Grundlagen dafür sollen aber geschafft werden.
Auswirkung	Vergleiche laufen in einem Thread, werden aber dennoch seriell ausgeführt.

Entscheid	Aufbau der REST URLs darf nicht verändert werden
Auswirkung	Lange URLs. Bevor eine Pfadvariable kommt, wird textuell beschrieben, um was für eine Pfadvariable es sich handelt.

Entscheid	Keine JavaDoc für jegliche public Methoden. JavaDoc nur dort schreiben, wo es einen Mehrwert bringt.
Auswirkung	Viele public Methoden erhalten keine JavaDoc.

Entscheid	Frontend Code muss an John Papa Styleguide angepasst werden
Auswirkung	Frontendkomponenten mussten teilweise an den neuen Styleguide angepasst werden.

7.3 Layer

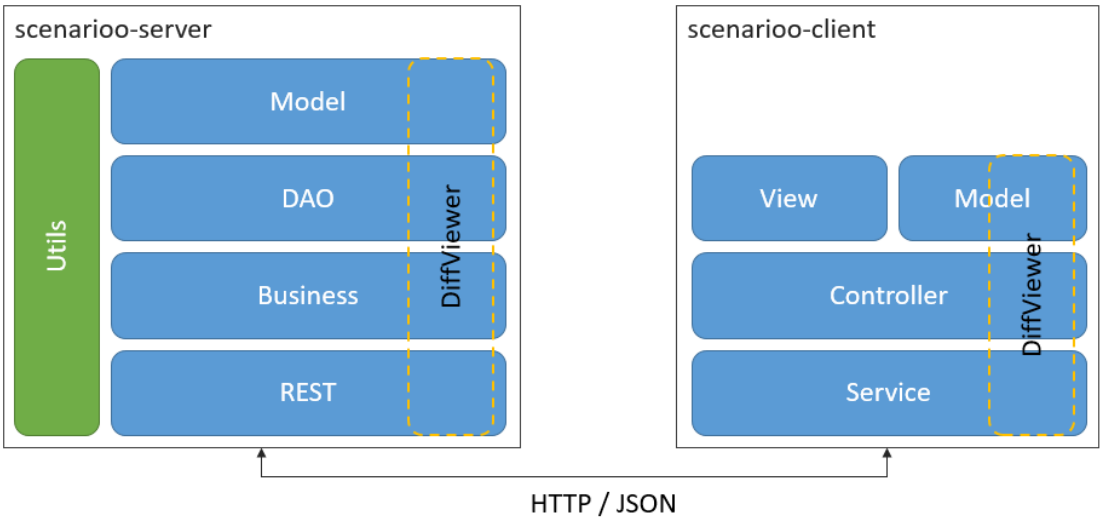


Abbildung 29: Layer von Scenarioo

An der grundlegenden Architektur von Scenarioo haben wir nichts geändert. Die Layer wurden beibehalten wie auf dem Diagramm ersichtlich.

7.4 Package- und Klassendiagramm

7.4.1 Frontend

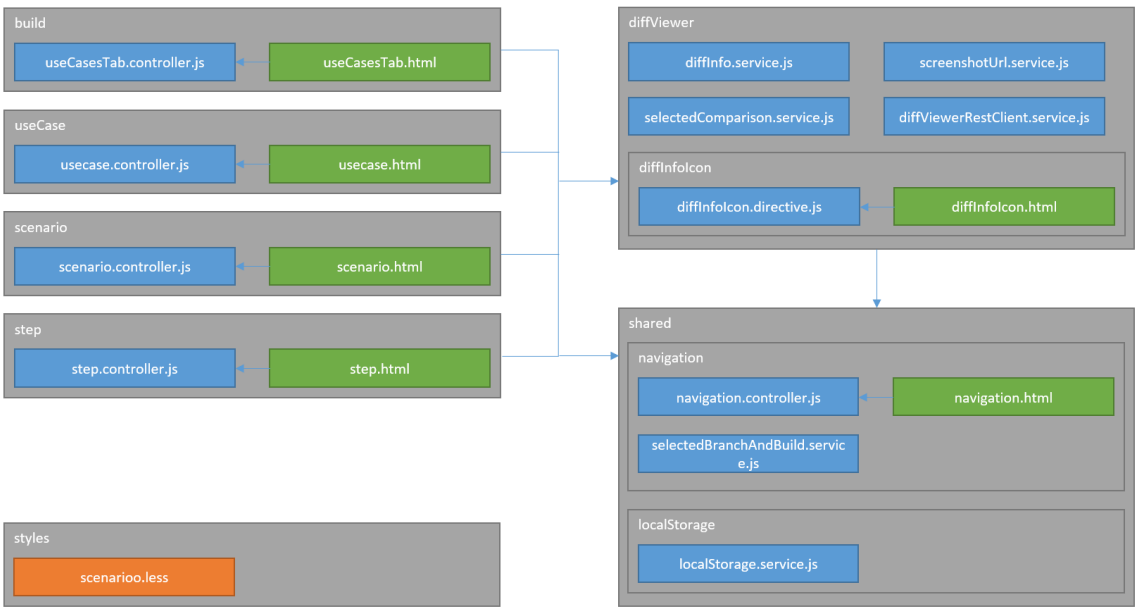


Abbildung 30: Die wichtigsten Frontend-Files für den Diff Viewer

Während unserer Arbeit hat Zühlke ein Refactoring am Frontend Code vorgenommen. Die Struktur richtet sich nun nach John Papas AngularJS Styleguide. Folgt man diesem Styleguide, so werden die Files nicht mehr nach Layern wie zum Beispiel Controller oder Services gruppiert, sondern nach logisch zusammengehörenden Komponenten.

Tabelle 7: Beschreibung Diff Viewer Frontend-Files

File	Beschreibung
useCasesTab.controller.js	Stellt die Use Case Diff-Informationen der View zur Verfügung.
useCasesTab.html	Stellt die Use Cases zusammen mit den Diff-Icons dar.
usecase.controller.js	Stellt die Szenario Diff-Informationen der View zur Verfügung.
usecase.html	Stellt die Szenarien zusammen mit den Diff-Icons dar.
scenario.controller.js	Stellt die Step Diff-Informationen der View zur Verfügung.
scenario.html	Stellt die Steps zusammen mit den Diff-Icons dar.
step.controller.js	Stellt die Step Diff-Information und die Screenshot-URLs der View zur Verfügung.
step.html	Neu gibt es einen zusätzlichen Comparison Tab auf dieser Seite. Dieser zeigt dem Benutzer den Comparison Screenshot und weitere Möglichkeiten zur Änderung der Darstellung.
diffInfo.service.js	Bereitet die Diff-Informationen so auf, dass sie von der View verwendet werden können.
screenshotUrl.service.js	Stellt Funktionalitäten zum Laden der Screenshots zur Verfügung.
selectedComparison.service.js	Weiss stets über den aktuell ausgewählten Comparison Build Bescheid.
diffViewerRestClient.service.js	Enthält alle REST Ressourcen, welche für den Diff Viewer relevant sind. Sie bilden die Schnittstelle zum Frontend.
diffInfoIcon.directive.js	Berechnet die Werte für das Diff-Icon und dessen Tooltip.
diffInfoIcon.html	Stellt das Diff-Icon dar.
navigation.controller.js	Stellt die verfügbaren Comparison Builds der View zur Verfügung.
navigation.html	Stellt nebst dem Branch und Build Menu auch das Comparison Menu dar

selectedBranchAndBuild.service.js	Weisst stets über den aktuell ausgewählten Build Bescheid.
localStorage.service.js	Speichert ausgewählte Werte in den locale Storage. Zum Beispiel wird der ausgewählte Comparison Build gespeichert.
Scenarioo.less	Enthält das Styling der Applikation

7.4.2 REST Ressourcen

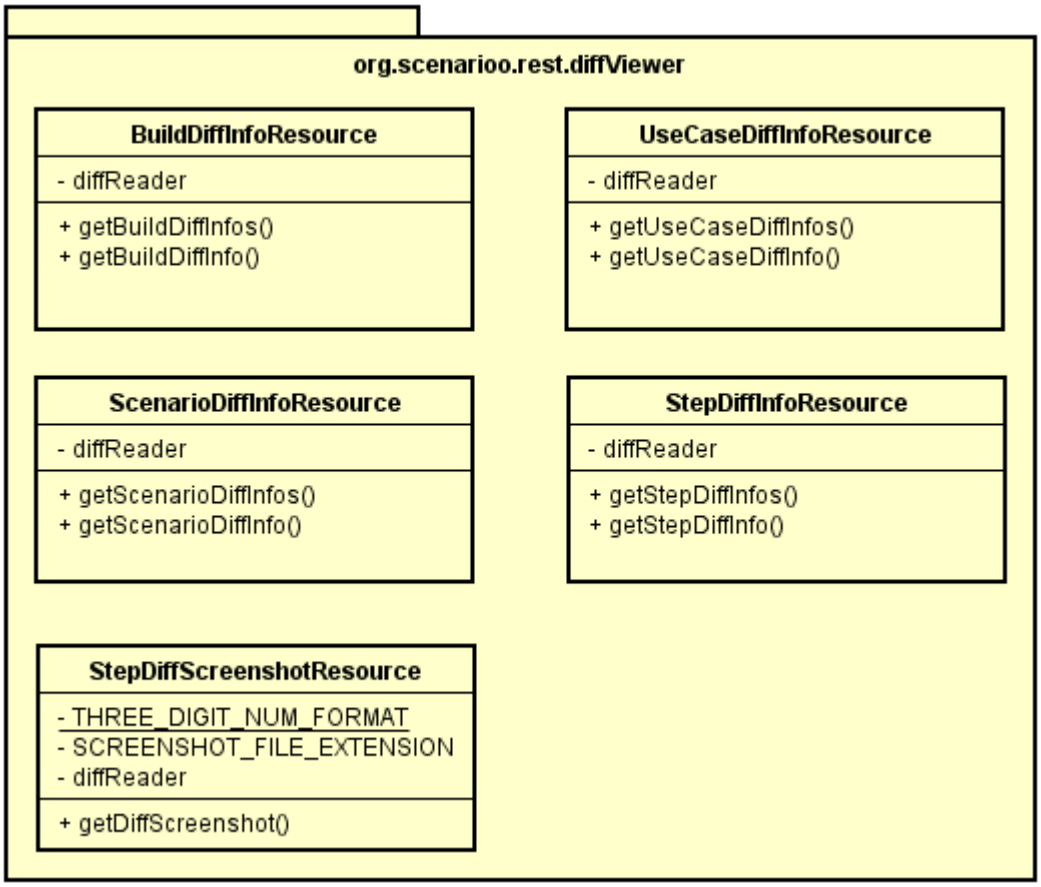


Abbildung 31: REST Resourcen für Diff Viewer

Damit die aus dem Vergleich gewonnenen Daten an den Client übergeben werden können, haben wir die REST Schnittstelle mit weiteren Ressourcen ergänzt. Für jedes Element existiert eine eigene Ressource, welche entweder eine Liste von Elementen oder ein einzelnes Element zurückgibt. Eine detaillierte Dokumentation der Schnittstelle befindet sich im Anhang.

7.4.3 Comparator

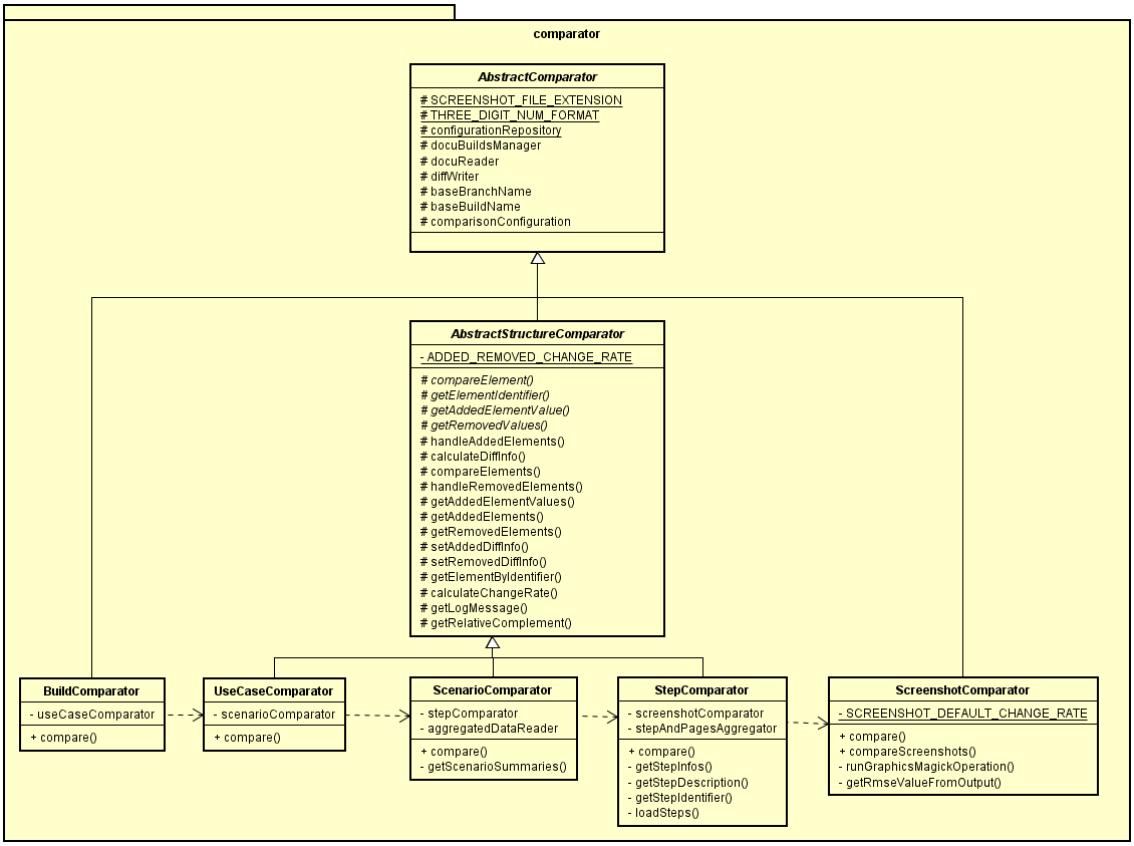


Abbildung 32: Klassendiagramm vom Comparator Package

Die Comparator-Klassen befinden sich im eigenen Package org.scenarioo.business.diffViewer.comparator innerhalb des scenarioo-server Projekts. Alle Comparator Klassen implementieren die abstrakte Klasse AbstractComparator und sind verantwortlich für den Vergleich der ihnen zugewiesenen Elemente (Use Case, Szenario, Step, …).

Tabelle 8: Beschreibung der Comparator Klassen

Klasse	Beschreibung
AbstractComparator	Abstrakte Klasse, welche gemeinsame Funktionalitäten aller Comparator implementiert.
AbstractStructure-Comparator	Abstrakte Klasse für alle Comparator, welche einen Strukturvergleich durchführen müssen.
ScreenshotComparator	Führt den Bildvergleich von zwei Screenshots aus.
StepComparator	Iteriert über alle Schritte in einem Szenario und vergleicht diese mit den entsprechenden Schritten vom Vergleichsbuild.

ScenarioComparator	Iteriert über alle Szenarien in einem Use Case und vergleicht diese mit den entsprechenden Szenarien vom Vergleichsbuild.
UseCaseComparator	Iteriert über alle Use Cases in einem Build und vergleicht diese mit den entsprechenden Use Cases vom Vergleichsbuild.
BuildComparator	Führt einen kompletten Buildvergleich aus.

7.4.3.1 Pattern Chain of Responsibility

Inspiriert wurden wir beim Design der Comparator Klassen vom Pattern Chain of Responsibility. Bei diesem sendet der Client eine Anfrage an den ersten konkreten Handler, welcher die Abstrakte Klasse oder das Interface implementiert und bekommt dann von irgendeinem konkreten Handler in der Kette die Antwort geliefert.

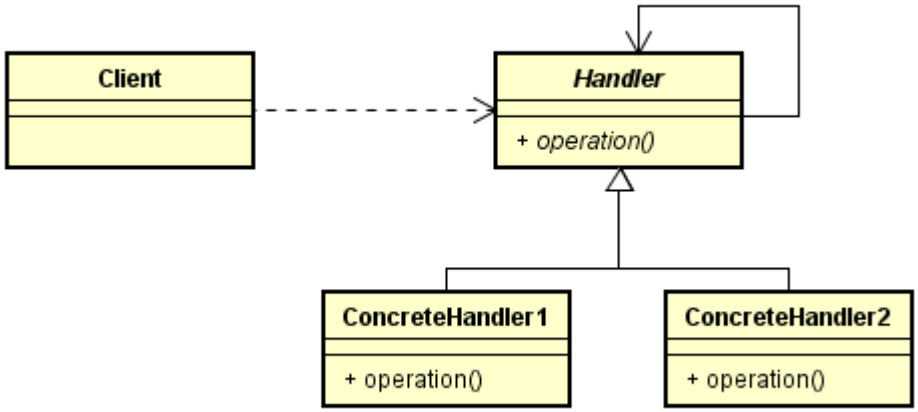


Abbildung 33: Pattern Chain of Responsibility

Unser Comparator Pattern unterscheidet sich allerdings leicht vom Chain of Responsibility. Jeder Comparator ist zwar nur für seinen Bereich zuständig und delegiert die weitere Aufgabe dann an den nächsten Comparator in der Kette, aber das Resultat der Compare Funktion ist je nach Comparator ein anderes. So retourniert ein StepComparator zum Beispiel ein ScenarioDiffInfo Objekt und der UseCaseComparator ein BuildDiffInfo Objekt.

Dennoch behalten wir auch in unserer Struktur den Vorteil der losen Kopplung. Ebenso ist es mit dieser Struktur möglich, nur einen Teil der Vergleichskette anzustossen. Dies erlaubt es, sehr spezifische Vergleiche durchzuführen. Sollte man an einem Vergleich von nur einem einzigen Szenario und den darunterliegenden Schritten interessiert sein, so ist dies mit diesem Design ohne Probleme möglich.

7.4.4 DAO

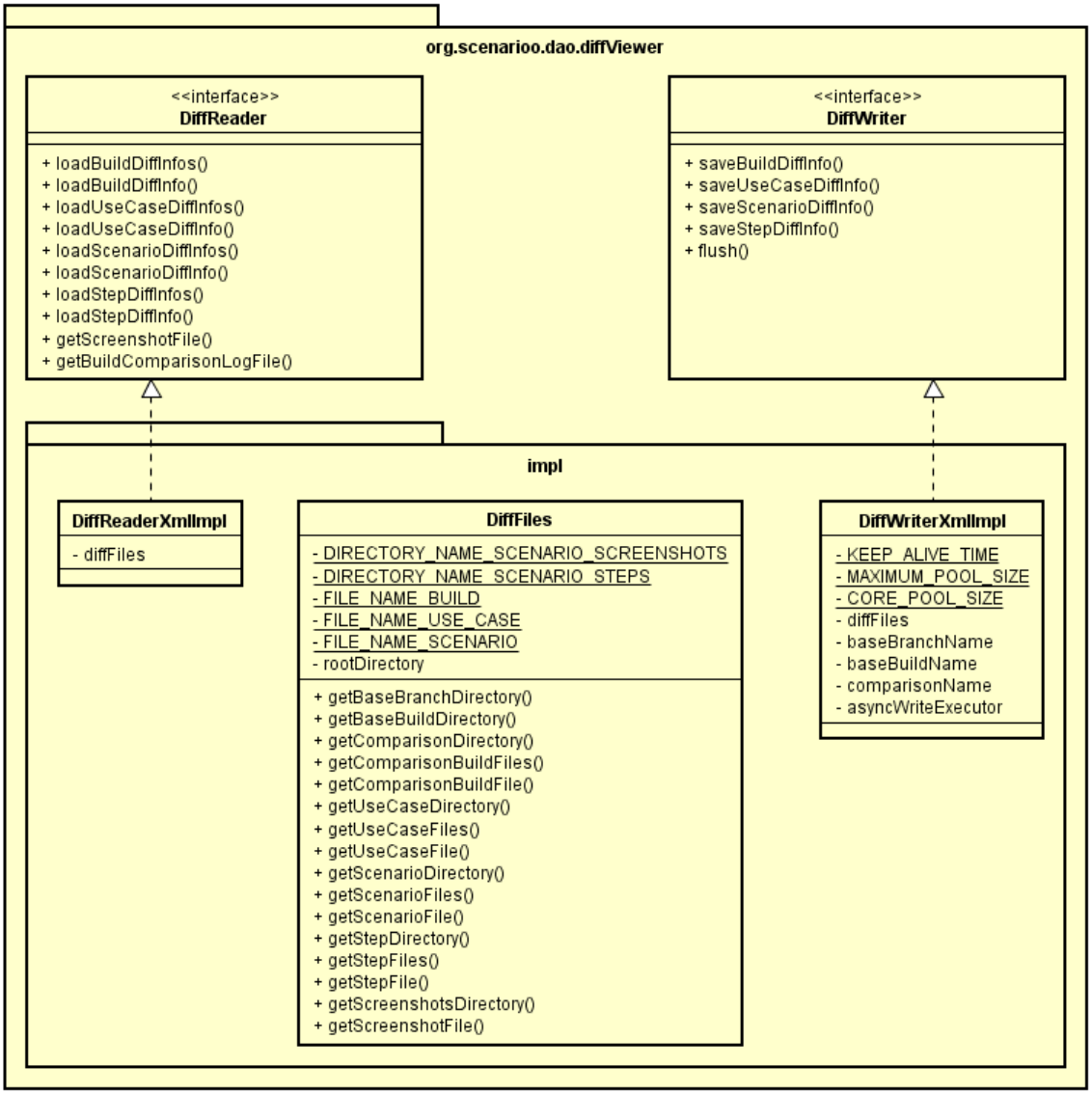


Abbildung 34: Klassendiagramm vom DAO Package

Bei den DAOs haben wir darauf geachtet, dass wir gegen ein Interface implementieren. Sollte in Zukunft eine andere Technologie zur Datenhaltung gewählt werden, so kann einfach die Implementierung ausgetauscht werden und das Interface bleibt bestehen.

Abbildung 35: Beschreibung der DAO Klassen

Klasse	Beschreibung
DiffReader	Interface zum Lesen der Diff-Informationen
DiffWriter	Interface zum Schreiben der Diff-Informationen
DiffReaderXmlImpl	XML Implementation vom DiffReader
DiffWriterXmlImpl	XML Implementation vom DiffWriter

DiffFiles	Repräsentiert die Dateistruktur und hilft den Implementierungsklassen beim Lesen und Schreiben.

7.4.5 Entities

Die aus dem Vergleich gewonnenen Diff-Informationen werden über die annotierten Entity-Klassen ins XML-Format abgespeichert.

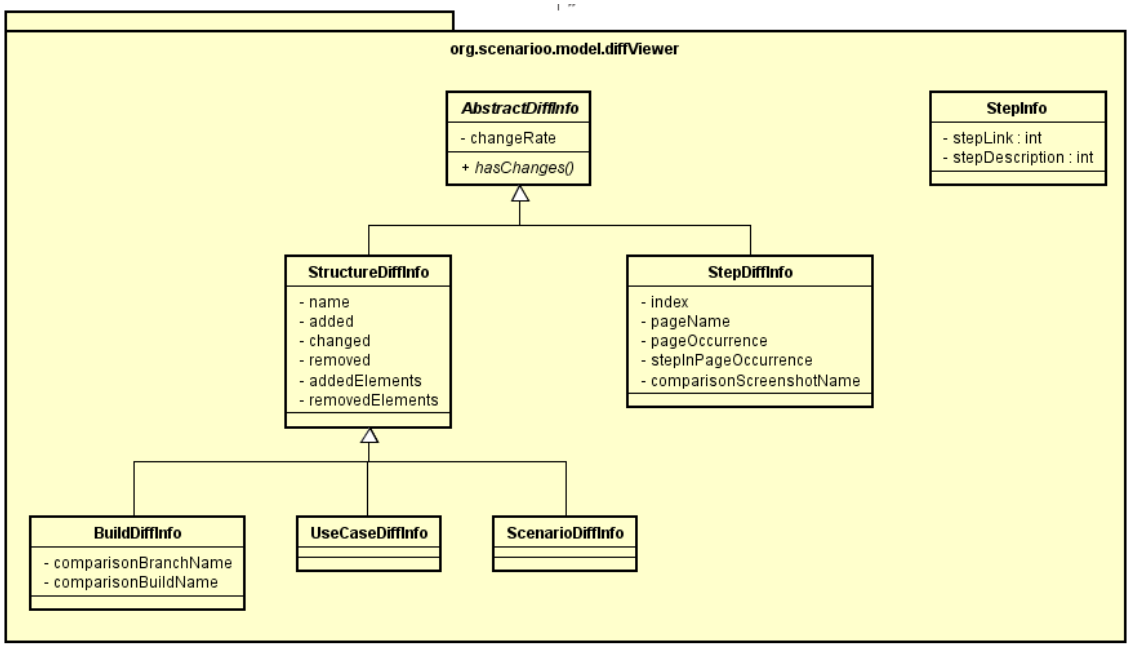


Abbildung 36: Neue Entity-Klassen

Tabelle 9: Beschreibung der neuen Entity-Klassen

Klasse	Beschreibung
AbstractDiffInfo	Abstrakte Klasse, welche gemeinsame Funktionalitäten implementiert und die abstrakte Methode hasChanges definiert.
StructureDiffInfo	Enthält allgemeine Diff-Informationen wie zum Beispiel ob ein Use Case hinzugekommen, entfernt oder modifiziert wurde.
BuildDiffInfo	Erbt von StructureDiffInfo. Enthält build-spezifische Informationen
UseCaseDiffInfo	Erbt von StructureDiffInfo. Enthält usecase-spezifische Informationen
ScenarioDiffInfo	Erbt von StructureDiffInfo. Enthält scenario-spezifische Informationen
StepDiffInfo	Enthält die für einen Step relevanten Diff-Informationen.
StepInfo	Entity Klasse für gelöschte Steps. Enthält alle wichtigen Informationen um einen gelöschten Step im Frontend darzustellen.

7.4.6 Gesamtüberblick

Ein Gesamtüberblick über alle Packages und Klassen kann dem Anhang entnommen werden.

7.5 Sequenzdiagramm

7.5.1 Strukturvergleich

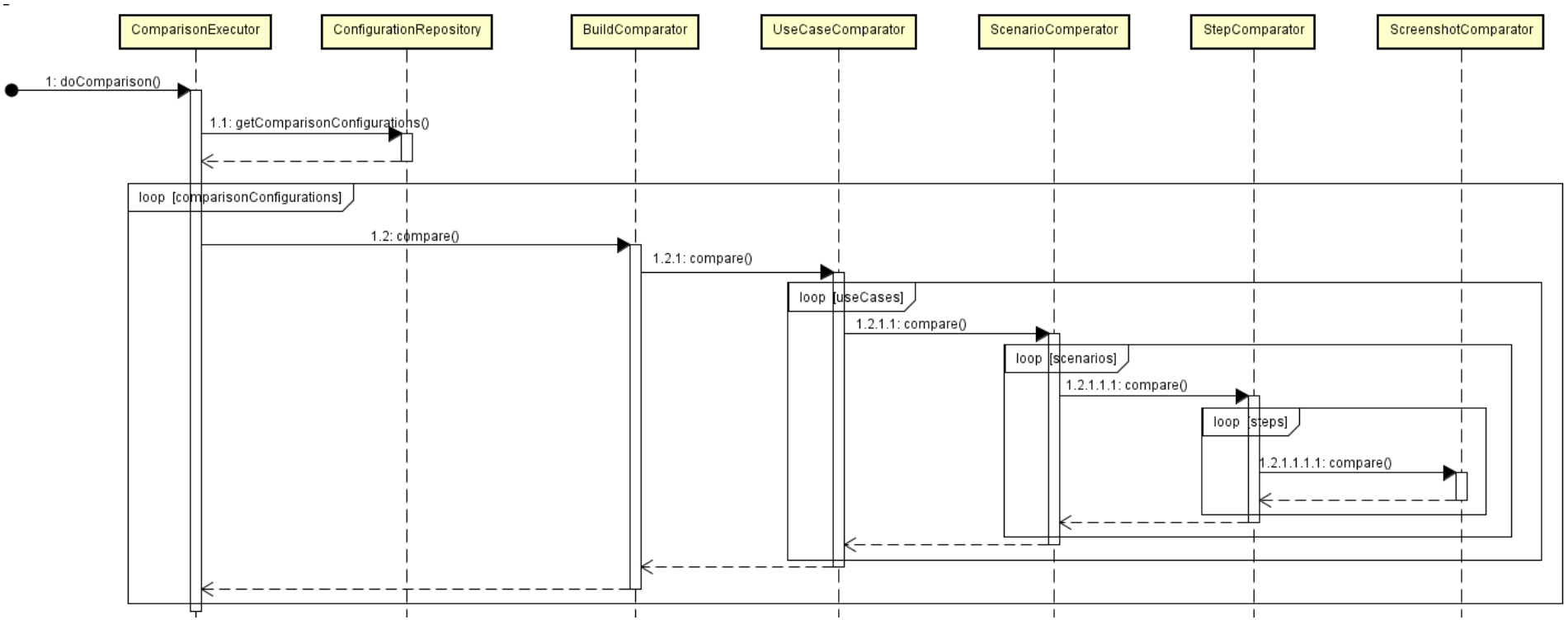


Abbildung 37: Sequenzdiagramm des Gesamtablaufs eines Vergleiches

Der Buildvergleich wird standardmässig beim Import angestossen. Der ComparisonExecutor liest die zu vergleichenden Builds aus und startet dann den Vergleichsprozess, indem er die compare Methode vom BuildComparator aufruft. Danach wird die gesamte Struktur durchlaufen und mit dem jeweiligen Comparator werden die entsprechenden Diff-Informationen gesammelt und im XML-Format abgespeichert.

Wie genau der Vergleich funktioniert, kann dem folgenden Kapitel Comparator entnommen werden.

7.5.2 Comparator

Das folgende Sequenzdiagramm zeigt anhand vom UseCaseComparator beispielhaft, wie ein Strukturvergleich funktioniert. Diese Logik ist auch für den ScenarioComparator und StepComparator gültig.

Eine Beschreibung der einzelnen Schritte befindet sich unterhalb des Sequenzdiagramms.

7.5.2.1 UseCaseComparator Überblick

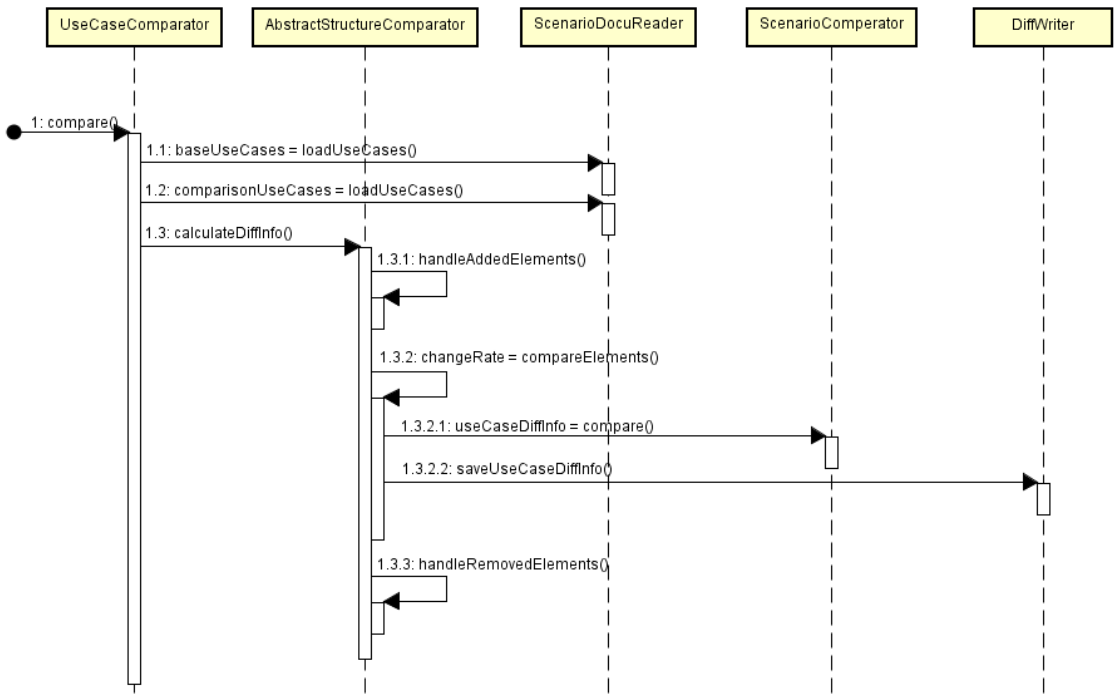


Abbildung 38: Sequenzdiagramm eines Strukturvergleichs

1.1 Lade alle Use Cases vom Base Build
1.2 Lade alle Use Cases vom Comparison Build
1.3 Berechne die Diff Information anhand der geladenen Use Cases
1.3.1 handleAddedElements berechnet die neuen Use Cases im Base Build. Diese Funktion wird im nächsten Diagramm detaillierter beschrieben.
1.3.2 compareElements iteriert über alle Use Cases und ruft den darunterliegenden Comparator auf

1.3.2.1 Der nächste Comparator (ScenarioComparator) wird aufgerufen. Dieser liefert die Diff Information vom Vergleich der Szenarien, welcher in diesem Use Case enthalten sind.
1.3.2.2 Die vom ScenarioComparator gelieferte Diff Information wird ins XML-Format abgespeichert
1.3.3 handleRemovedElements berechnet die gelöschten Use Cases im Base Build. Diese Funktion wird weiter unten detaillierter beschrieben.

7.5.2.2 handleAddedElements

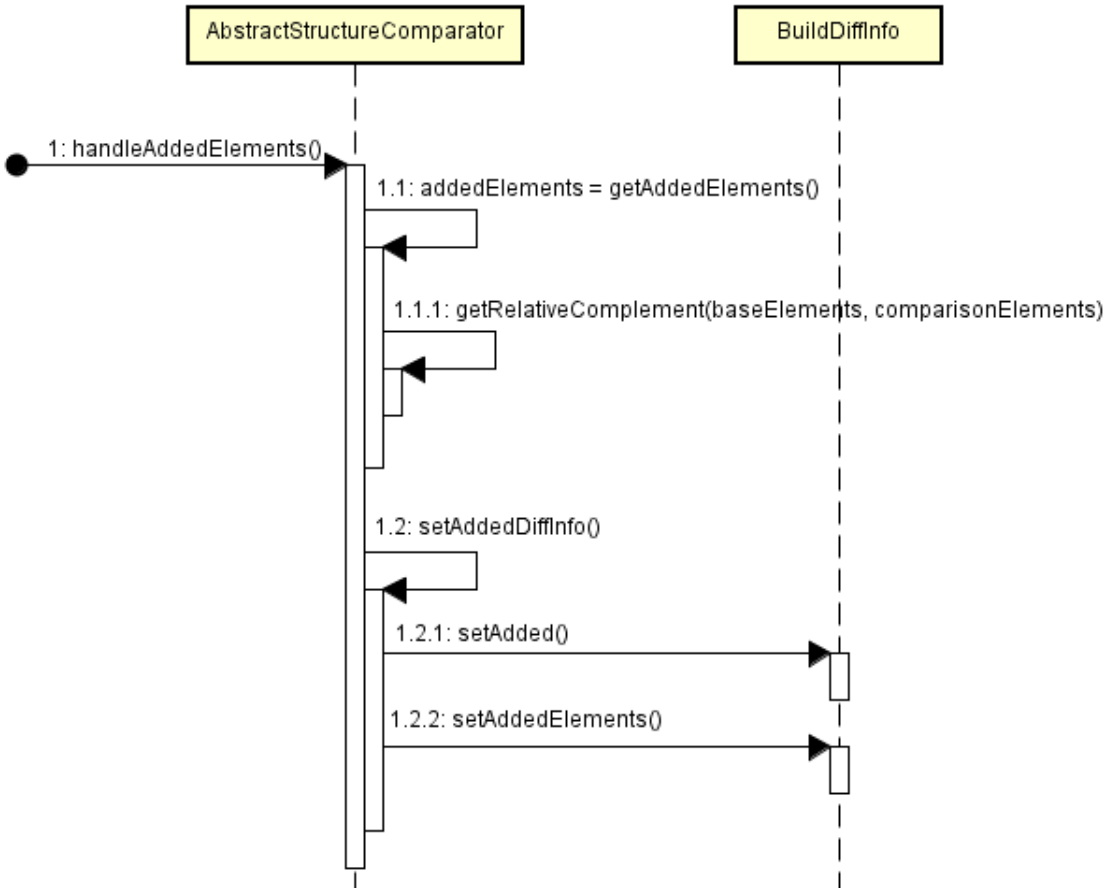


Abbildung 39: Sequenzdiagramm der Methode handleAddedElements

1.1 Finde alle neuen Elemente
1.1.1 Die Differenzmenge von allen Base Use Cases ohne Comparison Use Cases entspricht den neuen Use Cases
1.2 Die gewonnene Information wird auf das entsprechende Diff Info Objekt gesetzt
1.2.1 Setze die Anzahl neuer Use Cases
1.2.2 Setze die Werte der neuen Use Cases. In diesem Fall sind dies die Namen der neuen Use Cases.

7.5.2.3 CompareElements

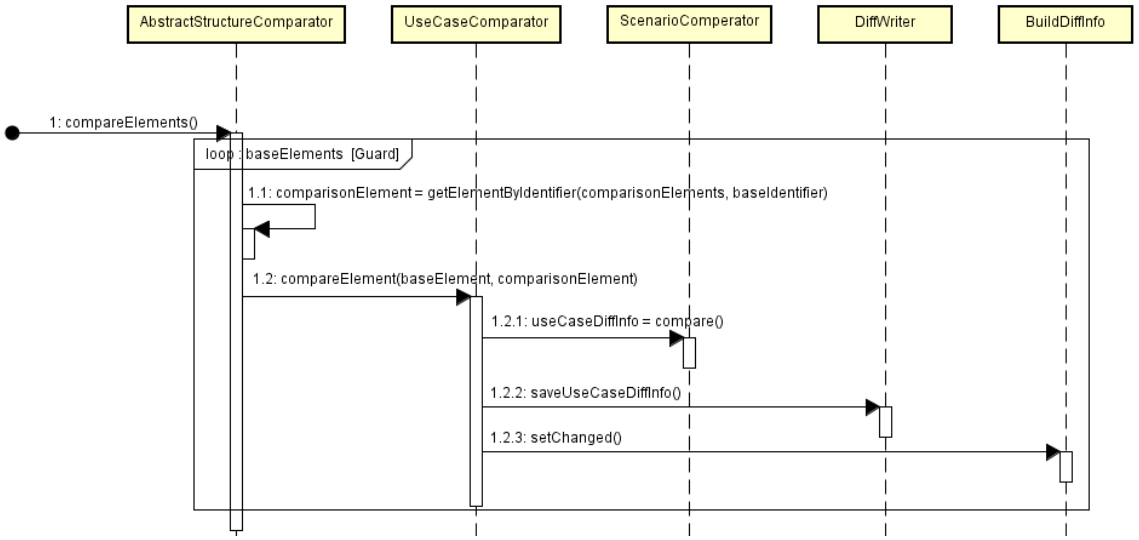


Abbildung 40: Sequenzdiagramm der Methode compareElements

1.1 Anhand vom Base Identifier wird der comparison Use Case ausgelesen
1.2 Sofern dieser nicht null ist, wird der Vergleich der beiden Elemente ausgeführt
1.2.1 Der nächste Comparator (ScenarioComparator) wird aufgerufen. Dieser liefert die Diff Information vom Vergleich der Szenarien, welcher in diesem Use Case enthalten sind.
1.2.2 Die vom ScenarioComparator gelieferte Diff Information wird ins XML-Format abgespeichert
1.2.3 Sollte der Vergleich der Szenarien Änderungen enthalten, so wird die Anzahl der geänderten Use Cases auf dem BuildDiffInfo um eins erhöht.

7.5.2.4 handleRemovedElements

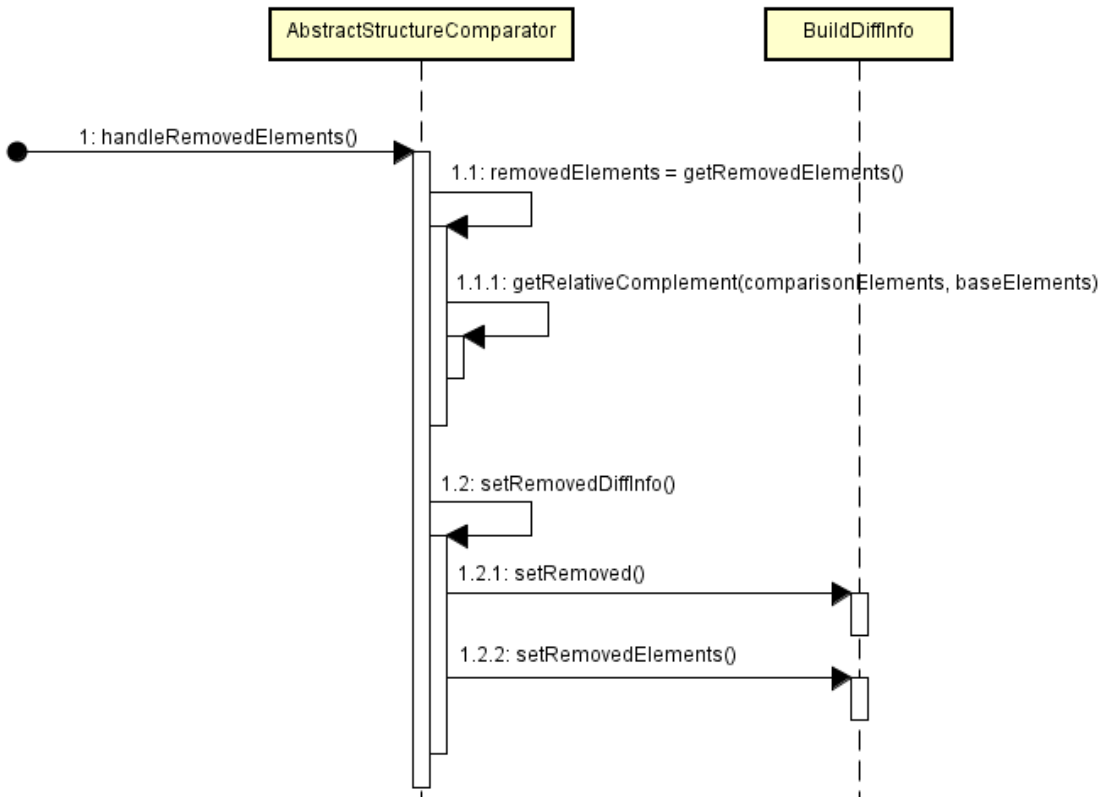


Abbildung 41: Sequenzdiagramm der Methode handleRemovedEelements

1.1 Finde alle gelöschten Elemente
1.1.1 Die Differenzmenge von allen Comparison Use Cases ohne Base Use Cases entspricht den gelöschten Use Cases
1.2 Die gewonnene Information wird auf das entsprechende Diff Info Objekt gesetzt
1.2.1 Setze die Anzahl gelöschter Use Cases
1.2.2 Setze die Werte der gelöschten Use Cases. In diesem Fall sind dies die Use Cases selber.

8 Implementation

8.1 Vergleich anstossen

Da wir beim Diff Viewer auf die Dokumentationsdaten der einzelnen Builds angewiesen sind, können Vergleiche erst ausgeführt werden, nachdem alle Builds komplett importiert sind. Noch nicht importierte Builds werden beim Starten der Applikation abgehandelt. Weiter ist es über die Weboberfläche möglich, einzelne Builds neu zu importieren.

Beide Aktionen lösen auch einen Vergleich für die entsprechenden Builds aus.

8.1.1 ComparisonExecutor

Die Klasse ComparisonExecutor im Packager org.scenarioo.business.diffViewer ist zuständig dafür, dass ein Base Build mit den korrekten Comparison Builds verglichen wird. Pro Vergleich wird ein separater Thread gestartet.

Damit die Vergleiche erst ausgeführt werden, nachdem alle benötigten Builds importiert sind, werden die Vergleich-Threads in denselben ThreadPool wie die Import-Threads geworfen.

Dieser ThreadPool kann zeitgleich nur einen Thread bearbeiten und so werden alle Threads wie in einer Queue nacheinander abgehandelt.

8.1.2 ThreadLogAppender

Die zu einem Vergleich gehörenden Log-Messages sollen in einem einzelnen Logfile gespeichert werden. Genau dasselbe Verhalten ist bereits bei den Imports vorhanden. Anstatt die bestehende Funktionalität zu kopieren, hat man diese so überarbeitet, dass sie wiederverwendbar wurde.

So verwenden nun der BuildImporter und der ComparisonExecutor den ThreadLogAppendner. Die einzelnen Threads werden beim ThreadLogAppender registriert und so werden alle Log-Messages vom entsprechenden Thread in das angegebene Logfile geschrieben.

Das separieren der Log-Messages bietet die Möglichkeit, die Log-Messages jederzeit im Frontend anzeigen zu können.

8.2 Änderungsgrad

Damit ein Benutzer erkennen kann wie sehr sich ein Use Case, ein Scenario, ein Step oder ein Screenshot verändert hat, wird ein sogenannter Änderungsgrad eingeführt. Dieser soll in Prozent angeben, wie sehr sich das entsprechende Element verändert hat. Dadurch kann der Benutzer leicht erkennen, wo sich die stärksten Änderungen befinden. Er hat zudem die Möglichkeit, Tabellen mit den verschiedenen Elementen nach dem Änderungsgrad zu sortieren.

8.2.1 Screenshots

Um den Änderungsgrad der Screenshots zu berechnen, können wir auf die Funktionalität von GraphicsMagick zurückgreifen. GraphicsMagick bietet unterschiedliche Metriken und somit unterschiedliche statistische Methoden an um den Unterschied zwischen zwei Bildern zu berechnen. Im Kapitel Bildvergleich wird ausführlich erklärt, welche Metrik wir verwenden und wie der Bildunterschied in Prozent berechnet wird.

8.2.2 Strukturunterschiede

Strukturunterschiede werden als 100% Änderung berechnet. Mit Strukturunterschiede sind folgende Änderungen gemeint:

- Step hinzugefügt oder gelöscht
- Scenario hinzugefügt oder gelöscht
- Page hinzugefügt oder gelöscht

8.2.3 Berechnung vom Änderungsgrad

Es wird aus allen unterliegenden Änderungswerten das arithmetische Mittel ausgerechnet. Es gibt keine spezielle Gewichtung für eine Art von Änderungswerten wie beispielsweise die von Screenshots.

8.3 Screenshot-Vergleich

Beim Screenshot Comparator wird GraphicsMagick verwendet um Screenshots zu vergleichen. Im Kapitel Evaluation wird ausführlich erklärt, weshalb genau dieses Bildverarbeitungsprogramm eingesetzt wird. Angesteuert wird das Konsolenprogramm über das pure-java interface IM4Java. In diesem Kapitel wird erklärt, wie wir den Screenshot-Vergleich für Scenarioo nutzen.

8.3.1 Metrik für Bildvergleich

GraphicsMagick bietet unterschiedliche Metriken und somit unterschiedliche statistische Methoden an, um den Unterschied zwischen zwei Bildern zu berechnen. Es wird zwischen Metriken unterschieden, die entweder auf Maximal- oder Durchschnittswerte ausgelegt sind. In einem ersten Schritt werden wir die bereitgestellten Metriken genauer untersuchen, um sie anschliessend mit Beispielbildern zu testen.

8.3.2 Untersuchung der Metriken

Grundsätzlich vergleicht GraphicsMagick die zwei Bilder Pixel für Pixel. Die Differenz jedes Pixels wird von jeder Metrik unterschiedlich gewichtet. Bei jedem Pixel wird zudem pro RGB Farbe eine eigene Statistik geführt. Der maximale Wert eines Pixels variiert je nach Farbtiefe des Bildes. Beim gebräuchlichsten RGB-Farbraum mit 8 Bit pro Kanal ist der maximale RGB Wert 256 (MaxRGB). (Farbtiefe (Computergrafik) 2016)

Nachfolgend wird erklärt, wie Pixelunterschiede von jeder Metrik unterschiedlich in die Statistik einberechnet werden. Die Diagramme können gemäss Abbildungsverweis auf wolframalpha.com nachgebildet werden.

Der GraphcisMagick Sourcecode ist vereinfacht dargestellt. Der Original Sourcecode befindet sich unter http://www.graphicsmagick.org/Hg.html.

8.3.2.1 Average Error Metriken

Average Error Metriken berechnen den durchschnittlichen Fehler über alle Pixel.

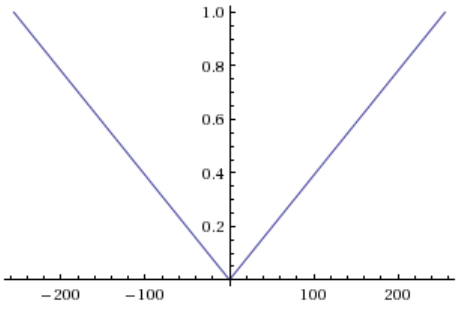
Mean Absolute Error (MAE)

Bei MAE wird der Kanalfehlerabstand linear berechnet. Wie der Name verrät wird dazu der Absolute Wert des Fehlers berechnet. Dadurch werden die individuellen Fehler gleich gewichtet.

Tabelle 10: Mean Absolute Error Details

Formel	MAE = $\frac{1}{n}\sum_{i=1}^{n} \lvert e_i \rvert$

Implementierung GraphicsMagick	`stats.red += fabs(pixelA.red - pixelB.red) / MaxRGB;`
	`statistics -> stats.red /= number_pixels;`
	Es wird der absolute Pixelunterschied durch den maximalen RGB Wert geteilt und aufsummiert. Anschliessend wird durch die Anzahl Pixel dividiert.

Diagramm	Dieses Diagramm zeigt die möglichen MAE Werte eines einzelnen Pixels im 8 Bit RGB Farbraum.  Abbildung 42: Diagramm MAE Pixel: | x | / 256 x=-256 to 256

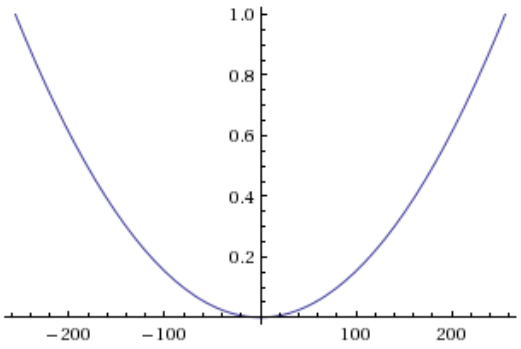
Mean Squared Error (MSE)

MSE berechnet die mittlere quadratische Abweichung der Fehlerdistanz. Das bedeutet eine höhere Gewichtung von grösseren Fehlern.

Tabelle 11: Mean Squared Error Details

Formel	MSE = $\frac{1}{n}\sum_{i=1}^{n} {e_i}^2$

Implementierung GraphicsMagick	`difference =( pixelA.red - pixelB.red)/ MaxRGB;`
	`stats.red += difference * difference;`
	`statistics-> stats.red /= number_pixels;`
	Es wird der Pixelunterschied durch den maximalen RGB Wert geteilt und quadriert. Anschliessend wird durch die Anzahl Pixel dividiert.

Diagramm	Dieses Diagramm zeigt die möglichen MSE Werte eines einzelnen Pixels im 8 Bit RGB Farbraum.  Abbildung 43: Diagramm MSE Pixel: (x / 256) ^2 x=-256 to 256

Root Mean Squared Error (RMSE)

RMSE berechnet die Wurzel der mittleren quadratischen Abweichung. Das Ziehen der Wurzel bewirkt, wie im Diagramm ersichtlich, dass höhere Werte als bei MSE erreicht werden.

Tabelle 12: Root Mean Squared Error Details

Formel	RMSE = $\sqrt{\frac{1}{n}\sum_{i=1}^{n} {e_i}^2}$

Implementierung GraphicsMagick	`difference =(pixelA.red - pixelB.red)/ MaxRGB;`
	`stats.red += difference * difference;`
	`statistics-> stats.red /= number_pixels;`
	`statistics->red=sqrt(statistics->red);`
	Es wird der Pixelunterschied durch den maximalen RGB Wert geteilt und quadriert. Anschliessend wird durch die Anzahl Pixel dividiert. Vom Endergebnis wird die Wurzel gezogen.

Diagramm	Dieses Diagramm zeigt die möglichen MSE Werte eines einzelnen Pixels im 8 Bit RGB Farbraum.

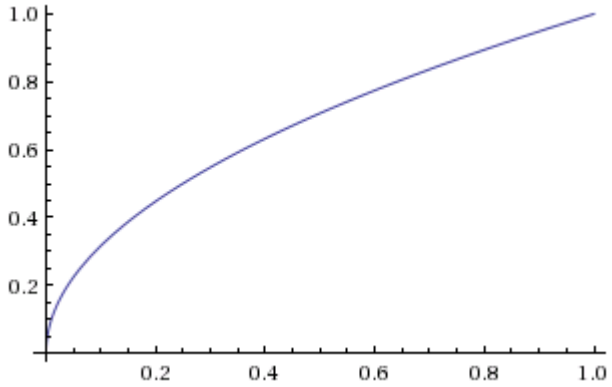


Abbildung 44: Diagramm RMSE Pixel: sqrt(x) x= 0 to 1

8.3.2.2 Maximum Error Metriken

Maximum Error Metriken sind auf den maximalen Fehler eines einzelnen Pixels ausgelegt.

Peak Absolute Error (PAE)

Der Pixelunterschied bei Peak Absolute Error Metrik wird gleich berechnet wie bei MAE. Allerdings wird nicht der Durchschnitt aller Abweichungen ausgegeben, sondern die maximal aufgetretene Abweichung.

Tabelle 13: Peak Absolute Error Details

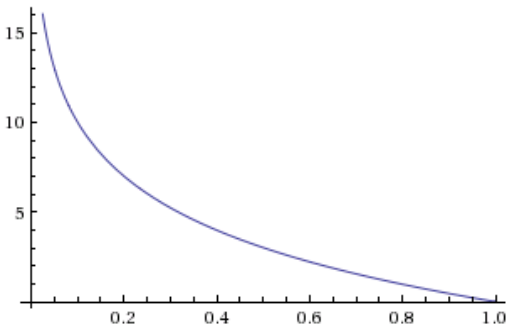
Implementierung GraphicsMagick	`difference= fabs(pixelA.red - pixelB.red) / MaxRGB;`
	
	`if (difference > stats.red)`
	`  stats.red=difference;`

Peak Signal to Noise Ratio (PSNR)

Peak Signal to Noise Ratio beschreibt das Signal-Rausch-Verhältnis. Der Pixelunterschied wird gleich berechnet wie bei MSE. Ausser eben das Verhältnis welches am Schluss einberechnet wird.

Tabelle 14: Peak Signal to Noise Ratio Details

Formel	PSNR = $20\ log_{10}\left(\frac{MAX}{\sqrt{MSE}}\right)$
Implementierung GraphicsMagick	`... siehe MSE` `statistics->red=(20.0 * log10(1.0/sqrt(statistics->red)));`

Diagramm	 Abbildung 45: Diagramm PSNR: 20 * log10(1.0 / sqrt(x)) x=0 to 1

8.3.2.3 Test der Metriken

Insgesamt haben wir vier unterschiedliche Testfälle angelegt. Die Testbilder wurden so ausgelegt, dass von ganz kleinen Änderungen bis zur totalen Veränderung alles dabei ist.

Tabelle 15: Testfälle für Metriken

Test	Beschreibung
Test 1	Der Seite wurde ein blauer Button mit der Grösse von 65px auf 240x hinzugefügt
Test 2	Das Originalbild wird mit dem Testbild aus dem Kapitel Evaluation Bildvergleich verglichen
Test 3	Oberhalb der Seite wurde ein gelber Werbebanner mit der Höhe von 100px hinzugefügt
Test 4	Zwei Screenshots aus einem produktivem Scenarioo Projekt werden verglichen. Auf dem einen Screenshot sind weitere Formularfelder eingefügt worden.
Test 5	Ein komplett Schwarzes Bild wird mit einem komplett weissen Bild verglichen.

Tabelle 16: Maximum Error Metriken

Metrik	Beschreibung	Test 1	Test 2	Test3	Test 4	Test 5
PAE	Peak Absolute Error	100%	100%	100%	100%	100%
PSNR	Peak Signal to Noise Ratio	28%	18%	16%	19.57%	1%

Tabelle 17: Test mit Average Error Metriken

Metrik	Beschreibung	Test 1	Test 2	Test3	Test 4	Test 5

MAE	Mean Absolute Error	0.17%	1.99%	3.28%	1.99%	75.00%
MSE	Mean squared error	0.16%	1.68%	2.36%	1.10%	75.00%
RMSE	Root Mean Squared Error	4.01%	12.97%	15.35%	10.50%	86.60%
RMAE	Root Mean Absolute Error	4.15%	14.11%	17.81%	14.11%	86.60%

8.3.3 Bestimmen einer Metrik

Durch die Tests haben wir festgestellt, dass einzig die Metrik RMSE einen zufriedenstellenden Wert berechnet. Die Maximum Error Metriken stehen sowieso ausser Frage, da wir einen Mittelwert der Pixelfehler wollen. Bei den MAE und MSE Metriken ist schlichtweg die Prozentzahl zu klein. Bei der Berechnung vom Änderungsgrad (in einem eigenen Kapitel beschrieben) würden solch tiefe Zahlen zu wenig ins Gewicht gefallen.

Der Grund dafür liegt darin, dass bei RMSE der Prozentwert durch die Wurzel erhöht wird. Das gleiche könnten wir auch bei MAE erreichen, indem wir am Schluss vom Resultat die Wurzel ziehen. Diesen Versuch haben wir durchgeführt und erreichten damit auch Werte in dieser Grössenordnung. Nun gilt es zu entscheiden ob RMSE oder die eigens kreierte RMAE Metrik die geeignetere Wahl für einen Screenshot-Vergleich ist. In der Wissenschaft teilen sich die Meinungen, welche Metrik verwendet werden sollte:

- Diese Arbeit spricht für für MAE, da im Gegensatz zu RMSE die Fehlerberechnung natürlich ist:
  Advantages of the mean absolute error (MAE) over the root mean square error (RMSE) in assessing average model performance (Cort J. Willmott* 2079)

- Diese Arbeit hingegen spricht für RMSE, besonders wenn die Fehler Normalverteilt sind:
  Root mean square error (RMSE) or mean absolute error (MAE)? – Arguments against avoiding RMSE in the literature (T. Chai 2014)

Für den Diff Viewer braucht die Prozentangabe nicht besonders exakt zu sein. Es ist viel wichtiger, dass der Benutzer abschätzen kann, wie stark sich eine Seite geändert hat. Daher ist es von Vorteil, wenn auch kleinere Änderungen stark gewichtet werden. Das ist besonders bei der Metrik RMAE der Fall. Aus diesem Grund werden wir diese Metrik für den Screenshot-Vergleich verwenden.

8.3.4 Aussehen vom Diff Screenshot

Die Unterschiede zwischen zwei Screenshots sollen auf einem neu erstellten Screenshot dargestellt werden. Auch hier wird grundsätzlich jeder Pixel zwischen den Bildern verglichen. Weist ein Pixel einen Unterschied auf, kann dieser auf dem Diff Screenshot unterschiedlich dargestellt werden. GraphicsMagick bietet dafür folgende Möglichkeiten:

Original: 

Abgeändert: 

Tabelle 18: GraphicsMagick Pixel Annotation Style

Style	Beschreibung	Beispiel
Assign (Zuordnen)	Ersetzt den Pixel mit einer vordefinierten Farbe	
Threshold (Schwelle)	Ersetzt den Pixel mit einem weissen oder schwarzen Pixel, je nachdem wie intensiv die Veränderung ist.	
Tint (Farbton)	Tönt den Pixel mit einer vordefinierten Farbe	
XOR (entweder oder)	Färbt den Pixel mit einer XOR Operation zur vordefinierten Farbe ein	

Für die Diff Viewer Funktionalität wird der Tint Style verwendet. So kann der Benutzer, dank der leichten Transparenz, erkennen, was sich an der ursprünglichen Stelle verändert hat.

8.4 Struktur der Diff Daten

Wie bereits in der Evaluation erwähnt, haben wir uns dazu entschieden, eine separate Dateistruktur für die Diff Daten aufzubauen. Im scenarioo-application-data Ordner gibt es neu den Unterordner diffViewer, in welchem alle Diff Daten abgespeichert werden. Die nachfolgende Grafik zeigt auf, wie die Gesamtstruktur der Diff Daten aufgebaut ist.

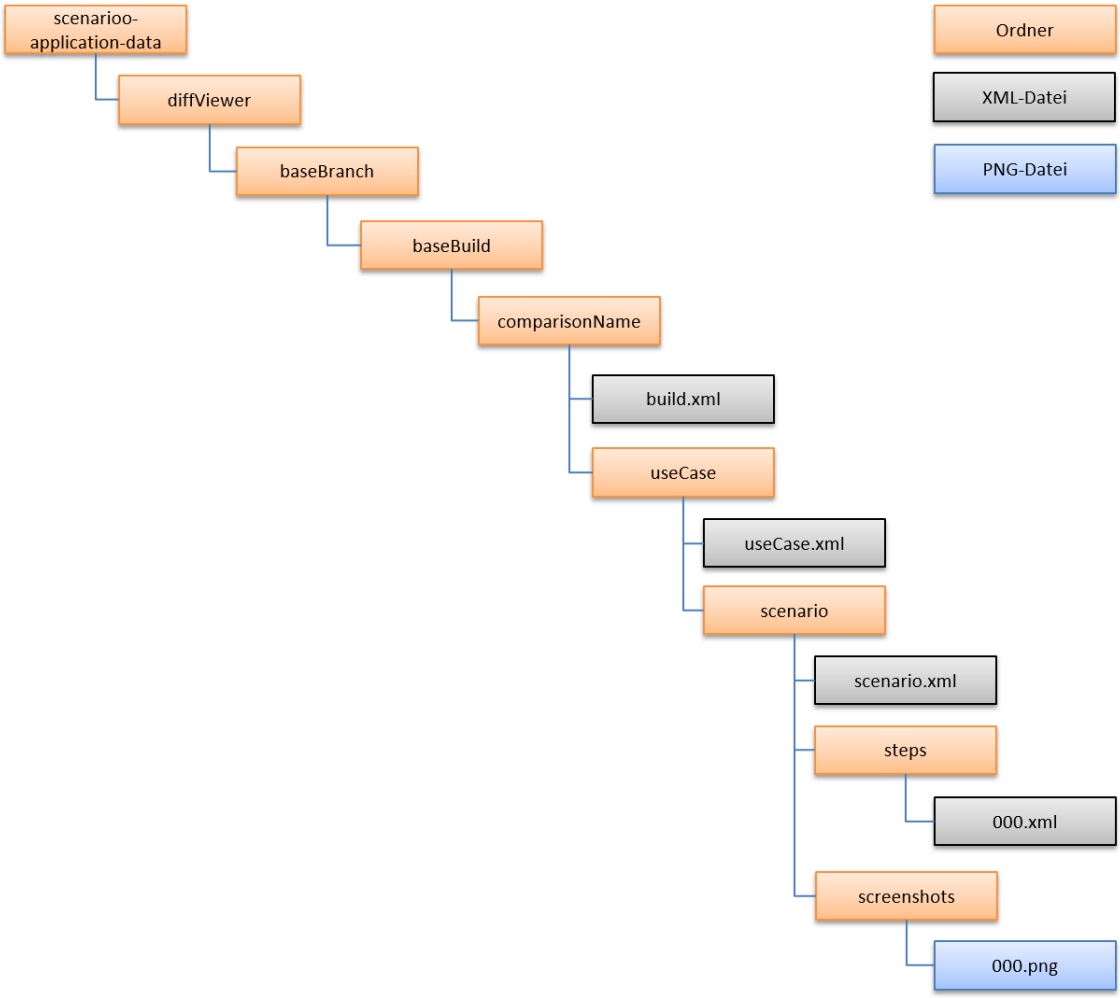


Abbildung 46: Gesamtstruktur der Diff Daten

8.5 Testverfahren

Das Resultat dieser Arbeit wird in die produktive Version von Scenarioo eingespielt. Deshalb ist es enorm wichtig, dass keine unerwarteten Fehler auftauchen und das System stabil läuft. Aus diesem Grund legen wir viel Wert darauf, unseren Code ausgiebig zu testen. Dazu können wir Frameworks vom bestehenden Projekt verwenden. Nachfolgend wird erklärt, wie mit Hilfe dieser Frameworks getestet wird.

8.5.1 Backend

8.5.1.1 Junit

Junit ist ein beliebtes Framework um Unit-Tests zu schreiben. In einem Junit-Test werden Annahmen getroffen, die erfüllt werden müssen. Gelingt der Test, wird er in grün dargestellt. Hat er jedoch einen Fehler, wird er rot markiert. Junit-Tests sind dank der Annotation @Test sofort erkennbar. (JUnit - About 2016)

Im Backend werden jene Methoden mit Junit geprüft, die eine selbst implementierte Logik beinhalten. Einfache getter und setter werden beispielsweise nicht geprüft.

Beispiel eines Scenarioo Junit-Tests:

```
@Test
public void testWriteAndReadStepDiffInfo() {
   final StepDiffInfo stepDiffInfo = getStepDiffInfo(STEP_INDEX);

   writer.saveStepDiffInfo(USE_CASE_NAME, SCENARIO_NAME, stepDiffInfo);
   writer.flush();

   final StepDiffInfo actualStepDiffInfo= reader.loadStepDiffInfo(
                              BASE_BRANCH_NAME, BASE_BUILD_NAME,
                              COMPARISON_NAME, USE_CASE_NAME,
                              SCENARIO_NAME, STEP_INDEX);

   assertStepDiffInfo(actualStepDiffInfo, STEP_INDEX);
}

private void assertStepDiffInfo(StepDiffInfo actualStepDiffInfo,
                              int expectedIndex) {
   assertEquals(CHANGE_RATE, actualStepDiffInfo.getChangeRate(), 0.0);
   assertEquals(expectedIndex, actualStepDiffInfo.getIndex());
   assertEquals(PAGE_NAME, actualStepDiffInfo.getPageName());
   assertEquals(PAGE_OCCURENCE,actualStepDiffInfo.getPageOccurrence());
   assertEquals(STEP_IN_PAGE_OCCURENCE,actualStepDiffInfo.
                              getStepInPageOccurrence());
}
```

8.5.1.2 Mockito

Mockito ist ein Framework zum Erstellen von Mock-Objekten in Unit-Tests. Damit können einzelne Objekte durch Mock-Objekte ersetzt werden. Somit müssen keine echten Testdaten vorhanden sein, sondern das Verhalten kann für das Mock-Objekt vorgegeben werden. Ein Mock-Objekt ist demzufolge ein Platzhalter für echte Objekte und liefert zum Testfall passende Werte zurück. (Mockito 2015)

Mockito wurde bisher nicht in Scenarioo eingesetzt. In Absprache mit Zühlke führten wir dieses neue Framework ein, um lesbare Tests zu schreiben, besonders in Situationen wo sonst keine Tests möglich gewesen wären. Ein Beispiel dafür ist der Screen-shot Vergleich. Um diesen Vergleich testen zu können, wäre die Installation von GraphicsMagick zwingend notwendig. Mit Mockito ist es nun möglich, die Funktionsweise von GraphicsMagick zu mocken und die Tests ohne dessen Installation durchzuführen. Ausserdem erleichtert es die Tests beim Strukturvergleich. Dank Mockito müssen keine XML-Dateien angelegt werden und auch die unterliegenden Comparators müssen nicht ausgeführt werden. All diese Abhängigkeiten können durch Mocks ersetzt werden.

Beispiel eines Scenarioo Junit Tests mit Mockito:

```
@Test
public void testParseGmConsoleOutput() {
   final File mockScreenshot = new File("mockScreenshot.png");

   try {
      doNothing().when(gmConsole).run(any(IMOperation.class));
   } catch (final Exception e) {
      e.getStackTrace();
   }

   when(gmConsoleOutputConsumer.getOutput()).thenReturn(
                                    OUTPUT_CONSUMER_MOCK);
   double difference = screenshotComparator.compareScreenshots(
                                  screenshot, mockScreenshot);
   assertEquals("Difference of screenshots", SCREENSHOT_DIFFERENCE,
                                    difference, DOUBLE_TOLERANCE);
}
```

8.5.2 Frontend Tests mit Jasmine

Jasmine ist ein verhaltensbasiertes Framework, um JavaScript Code zu testen. Jasmine wird für Unit-Tests verwendet und kann zudem Mocks von Objekten anlegen. (Jasmine Introduction kein Datum)

Bei Scenarioo wird mit Jasmine überprüft, ob die im scenarioo-docu-generation-example generierten Inhalte korrekt geladen und angezeigt werden.

Beispiel eines Scenarioo Unit-Tests mit Jasmine:

```
describe('scenario is found', function() {

   beforeEach(function() {
      $routeParams.stepInPageOccurrence = 1;
      $controller('StepController', {
                              $scope: $scope,
                              $routeParams: $routeParams,
                              $location: $location,
                              $q: $q,
                              $window: $window,
                              ConfigService: ConfigService,
                              ScenarioResource: ScenarioResource,
                              StepResource: StepResource,
                              HostnameAndPort: HostnameAndPort,
                              BranchAndBuildService:BranchAndBuildService,
                              DiffInfoService: DiffInfoService,
                              ApplicationInfoPopupService: {},
```

```
SharePagePopupService: {}
                        });

     spyOn(RelatedIssueResource, 'query').
                             and.callFake(queryRelatedIssuesFake());
     spyOn(BranchesResource, 'query').and.callFake(getEmptyData());
     spyOn(BuildDiffInfoResource, 'get').and.callFake(getEmptyData());
     spyOn(StepDiffInfoResource, 'get').and.callFake(getEmptyData());
   });

   it('loads the step data', function () {
     loadPageContent();
     expect($scope.step).toEqualData(TestData.STEP.step);
   });
});
```

8.5.3 End-to-end Tests

End-to-end Tests in Scenarioo verfolgen zwei Ziele. Erstens wird, wie der Name schon sagt, die gesamte Funktionalität von einem Ende zum anderen Ende geprüft. Dadurch wird sowohl der Code vom scenarioo-server im Backend, als auch der Code vom scenarioo-client im Frontend getestet. In einem solchen Test wird der gesamte Vorgang vom Auslesen der XML-Dateien bis hin zur Darstellung im Browser durchgeführt.

Das zweite Ziel, welches verfolgt wird, ist eine automatische Dokumentation der Scenarioo Funktionalitäten. Idealerweise werden die end-to-end Tests im Scenarioo Format abgelegt. Dadurch kann das Resultat der end-to-end Tests in Scenarioo importiert und dargestellt werden. Auf der Demo Seite von Scenarioo kann jederzeit die sogenannte „self-docu“ vom aktuellsten Master Branch aufgerufen werden: http://demo.scenarioo.org/scenarioo-master/#/?branch=scenarioo-self-docu&build=last%20successful&tab=usecases

8.5.3.1 Protractor

Protractor ist ein end-to-end test framework für AngularJS Applikationen. Protractor ruft für die Tests einen Browser auf und verhält sich wie ein User, der eine Webseite bedient. Protractor basiert auf WebDriverJS, welches Browserspezifische Treiber lädt, um wie ein Benutzer agieren zu können. Protractor ermöglicht es, AngularJS spezifische Elemente ohne zusätzliches Setup zu verwenden, indem es dessen Locator Strategies implementiert. (Protractor Home 2016)

Beispiel eines Scenarioo end-to-end Tests mit Protractor:

```
scenario('Screen without annotations')
   .description('No screen annotations are shown if…')
   .it(function () {
     stepPage.goToPage('/step/Find Page/results/startSearch.jsp/0/0');
     stepPage.assertScreenshotIsShown();
     stepPage.assertNoScreenAnnotationsArePresent();
     step('No screen annotations are shown');
   });

StepPage.prototype.assertScreenshotIsShown = function() {
   expect(element.all(by.className('sc-screenshot')).count()).toBe(1);
   expect(element(by.className('sc-screenshot')).isDisplayed()).
                           toBeTruthy();
};
```

8.5.4 Kompatibilität Tests

Scenarioo wird mit unterschiedlichen Browsern verwendet und Entwickler arbeiten mit unterschiedlichen Betriebssystemen. Um die Kompatibilität mit diesen Systemen weiterhin zu gewährleisten, haben wir mit unterschiedlichen Entwicklungsumgebungen gearbeitet:

Entwicklungsumgebung Pascal:

- Betriebssystem: Windows 7, Windows 10
- Browser: Firefox

Entwicklungsumgebung Manuel:

- Betriebssystem: Ubuntu 14.04
- Browser: Chrome

Zudem wurde das Diff Viewer Feature mit dem Internet Explorer 11 getestet, weil erfahrungsgemäss die Webseiten im Internet Explorer leicht unterschiedlich dargestellt werden.

8.6 Abgrenzung Code

Da wir eine bestehende Applikation erweitern, ist es nicht einfach festzustellen, welcher Code von uns stammt. Dieses Kapitel soll helfen, den von uns geschriebenen Code zu identifizieren.

8.6.1 Github Übersicht

Der Code von Scenarioo wird auf GitHub verwaltet. Dazu nachfolgend die wichtigsten Eckdaten.

8.6.1.1 Benutzer

Mit folgenden GitHub Benutzern wurde der von uns geschriebene Code eingecheckt:

- Manuel Scheuber: author = magitnu
- Pascal Forster: author = dogen91

8.6.1.2 Issues

Zu jeder User-Story wurde ein GitHub issue erstellt und jeder Commit enthält eine entsprechende Referenz darauf. Die Issues vom Diff Viewer sind unter folgender Adresse auffindbar:

https://github.com/scenarioo/scenarioo/milestones/3.x%20Diff%20Viewer%20Release

8.6.1.3 Repositories

- Offizielles Scenarioo Repository mit Issues und einem Wiki : https://github.com/scenarioo/scenarioo
- Unser Fork vom Scenarioo Repository, welcher regelmässig mit dem original Repository abgeglichen wird: https://github.com/magitnu/scenarioo

8.6.1.4 Wiki

Bis zum produktiven Einsatz des Diff Viewers werden dessen Wiki Einträge nur auf unserem Fork verfügbar sein. Es handelt sich dabei um folgende Einträge:

- https://github.com/magitnu/scenarioo/wiki/Diff-Viewer-Installation-and-Configuration-Guide
- https://github.com/magitnu/scenarioo/wiki/Diff-Viewer-End-User-Guide

8.6.2 Scenarioo-server

Auf dem Server lässt sich der von uns geschriebene Code relativ leicht identifizieren. Denn für einen Grossteil vom Diff Viewer Code haben wir eigene Packages angelegt. Diese Packages lauten wie folgt:

- src/main/java
  - org.scenarioo.business.diffViewer
  - org.scenarioo.business.diffViewer.comparator
  - org.scenarioo.dao.diffViewer
  - org.scenarioo.model.diffViewer
  - org.scenarioo.rest.diffViewer
- src/test/java
  - org.scenarioo.business.diffViewer.comparator
  - org.scenarioo.dao.diffViewer

Weitere Klassen bei denen wir Änderungen vorgenommen haben:

- src/main/java
  - org.scenarioo.utils
    - NumberFormatCreator
    - ThreadLogAppender
  - org.scenarioo.business.aggregator
    - StepsAndPagesAggregator

8.6.3 Scenarioo-client

Auf dem Client wird die Identifizierung unseres Codes schon schwieriger. Das liegt daran, dass wir unseren Code in die bestehenden Views und Controllers integriert haben.

Es gäbe die Möglichkeit, alle von uns committeten Dateien anzeigen zu lassen. Doch gerade bei Pascal Forster wird aufgrund eines umfangreichen Merges eine riesige Liste ausgegeben. Die Syntax für den Befehl in der Git Bash lautet:

```
git log --no-merges --stat --committer="<email>" --name-only --
pretty=format:"" | sort –u
email entweder paesc91@hotmail.com oder manuel.scheuber@gmail.com
```

Grob zusammengefasst stammt folgender Code von uns:

- app/diffViewer: Der gesamte Inhalt
- app/images/diffViewer: Der gesamte Inhalt
- Bei den Views: Code innerhalb von <sc-diff…> Direktiven oder Elemente die mit ng-if="comparisonInfo.isDefined" ergänzt sind.
- Bei den Controllern: Funktionen, die eine …DiffInfoResource verwenden oder Variablen die das Word comparison beinhalten.
- Auch an anderen Orten wurden Anpassungen vorgenommen. Beispielsweise bei den Tests oder im Stylesheet. Doch mit obigen vier Punkten sollte ein Grossteil unseres Codes auffindbar sein.

8.6.4 Scenarioo-docu-generation-example

Im scenarioo-docu-generation-example werden die Testdaten für die end-to-end Tests generiert. Diese Testdaten haben wir mit einem zweiten Branch ergänzt, um alle möglichen Strukturunterschiede testen zu können.

8.7 Code Reviews

8.7.1 Code Reviews Zühlke

Die folgenden Anpassungen haben wir gemäss dem Feedback von Zühlke zu unserem Code vorgenommen.

Naming

Beschreibung:	Während dem Review haben wir das Naming diverser Klassen oder Methoden angepasst.
Refactoring	Glossar erstellt und betroffene Codestellen angepasst. Link zum Glossar: https://github.com/magitnu/scenarioo/blob/develop/documentation/diff-viewer/glossary.md

REST URL Pfade

Beschreibung:	Um kürzere Pfade bei unserer REST API zu erreichen, haben wir auf die Beschreibung der Parameter verzichtet. Gemäss Zühlke sollen wir ihr Paradigma mit der Beschreibung aus Konsistenzgründen beibehalten, da ansonsten API Benutzer über die unterschiedlichen Ansätze verwirrt sein könnten.
Betroffener Code:	/scenarioo-server/src/main/java/org/scenarioo/rest/diffViewer/ …DiffInfoResource.java
Refactoring	Pfad vor der Anpassung: @Path("/rest/diffViewer/{baseBranchName}/{baseBuildName}") Pfad nach der Anpassung: @Path("/rest/diffViewer/baseBranchName/{baseBranchName}/baseBuildName/{baseBuildName}")

Fehlerbehandlung Backend

Beschreibung:	Beim Aufruf von nicht vorhandenen Ressourcen haben wir jeweils eine Warnung mit log4j ausgegeben. Dies ist unter anderem auch der Fall bei einem neu hinzugefügten Scenarioo-Element. Denn zu diesem neuen Element existieren keine DiffInfos, welche geladen werden könnten. Wird im Frontend beispielsweise ein neuer Step verglichen, können keine Diff Informationen dazu abgerufen werden, weil diese nicht existieren. Gemäss Zühlke soll die dabei entstandene Exception allerdings nicht behandelt, sondern weitergegeben werden.
Betroffener Code:	/scenarioo-server/src/main/java/org/scenarioo/rest/diffViewer/ …DiffInfoResource.java
Refactoring	Log4j Warnung entfernt, damit direkt eine RessourceNotFoundException geworfen wird. Dadurch erscheint im FrontEnd ein Console Error wenn eine Ressource nicht vorhanden ist.

Java Doc

Beschreibung:	Wir haben bei allen Public Methoden Javadoc-Kommentare verfasst um die Methode und deren Parameter zu beschreiben. Gemäss Zühlke soll im Scenarioo Projekt allerdings nur ein Kommentar geschrieben werden, wenn er zusätzliche Informationen enthält. Bei Methoden sollte der Methodenname schon für sich selbst sprechen. Ein Kommentar ist beispielsweise dann nötig, wenn die Methode nicht sofort erkennbare Nebeneffekte ausführt.
Betroffener Code:	Sämtliche Javadoc-Kommentare ohne Zusatzinformationen.
Refactoring	Überflüssige Javadoc-Kommentare entfernen.

John Papa Styleguide

Beschreibung:	Während unserer Arbeit hat Zühlke den scenarioo-client Code gemäss dem John Papa Styleguide (https://github.com/johnpapa/angular-style-guide) überarbeitet.
Betroffener Code:	Ein Grossteil des Codes vom scenarioo-client
Refactoring	Den neuen Stand vom scenarioo-develop Branch mit unserem Fork mergen. Unseren Code gemäss den von Zühlke angewendeten John Papa Regeln angepasst.

8.7.2 Code Review Michael Gfeller

Die folgenden Anpassungen haben wir gemäss dem ersten Feedback von Michael Gfeller zu unserem Code vorgenommen.

Long Method vermeiden

Beschreibung:	Wo immer möglich sollten die Methoden kurz und knackig sein.
Refactoring	Besonders die compare Methode im den Comparators war zu lang. Wir haben diese in mehrere kleine Methoden auf gesplittet.

Duplicate Code vermeiden

Beschreibung:	Ähnlicher oder sogar gleicher Code soll ausgelagert werden.
Refactoring	In den Comparators gab es einige Codestellen, welche zwar von der Logik her gleich waren, aber sich dennoch leicht unterschieden. Wir haben diese Stellen mit Generics in eine neue abstrakte Klasse ausgelagert und konnten so einen Grossteil der Comparator Klasse auslagern.

8.8 Code Guidelines

Im Scenarioo Wiki sind die Coding Guidlines sowie alle DIE-Settings genauestens dokumentiert:

Scnearioo Coding Guidlines:
https://github.com/scenarioo/scenarioo/wiki/Coding-guidelines

Eclipse Settings:
https://github.com/scenarioo/scenarioo/wiki/Eclipse-IDE-Settings

Webstorm Settings:
https://github.com/scenarioo/scenarioo/wiki/WebStorm-IDE-Settings

8.9 Technologie Stack

8.9.1 Backend

Nachfolgende Technologien werden im scenarioo-server Projekt eingesetzt.

8.9.1.1 Tomcat

Apache Tomcat ist ein Open Source Webserver für Java Servlet, JavaServer Pages, Java Expression Language und Java WebSocket Technologies. Die scenarioo-server Anwendung wird darauf ausgeführt.

http://tomcat.apache.org/

8.9.1.2 Mockito

Mockito ist ein Framework zum Erstellen von Mock-Objekten in Unit-Tests. Mit Mockito können einzelne Objekte durch Mock-Objekte ersetzt werden und somit isoliert von ihrer Umgebung getestet werden. Bei Scenarioo wird Mockito in Diff Viewer Tests im Bereich von Screenshot- und Strukturvergleichen verwendet.

http://mockito.org/

8.9.1.3 Im4java

Im4java ist ein Java Interface zur ImageMagick oder GraphicsMagick commandline. Bei Scenarioo werden die Befehle für den Screenshot-Vergleich darüber abgesetzt.

http://im4java.sourceforge.net/

8.9.1.4 Log4j

Log4j ist ein Framework zum Loggen von Anwendungsmeldungen in Java. Sämtliche scenarioo-server Infos, Warnungen und Fehler werden darüber ausgegeben und gegebenenfalls gespeichert.

http://logging.apache.org/log4j/2.x/

8.9.1.5 JAX-RS

Java API for RESTful Services (JAX-RS) ist eine API um REST Schnittstellen für Java Webservices anzulegen.

https://jax-rs-spec.java.net/

8.9.2 Frontend

Nachfolgende Technologien werden im scenarioo-client Projekt eingesetzt.

8.9.2.1 Node.js

Node.js ist eine event-basierte, serverseitige Plattform basierend auf Google Chromes V8 JavaScript engine. Node.js wird meistens, auch beim scenarioo-client, als Webserver verwendet.

https://nodejs.org/en/

8.9.2.2 Jasmine

Jasmine ist ein verhaltensbasiertes Framework um JavaScript Code zu testen. Jasmine wird für Unit Tests verwendet und kann zudem Mocks von Objekten anlegen. (Jasmine Introduction kein Datum)

Bei Scenarioo wird mit Jasmine überprüft, ob die im scenarioo-docu-generation-example generierten Inhalte korrekt geladen und angezeigt werden.

http://jasmine.github.io/

8.9.2.3 AngularJS

AngularJS ist ein clientseitiges JavaScript-Webframework zur Erstellung von Single-page-Webanwendungen nach einem Model-View-ViewModel-Muster. (AngularJS 2016)

https://angularjs.org/

8.9.2.4 Protractor

Protractor ist ein end-to-end test framework für AngularJS Applikationen. Protractor ruft für die Tests einen Browser auf und verhält sich wie ein User, der eine Webseite bedient. Die End-to-end Tests sowie die self-docu von Scenarioo benutzen dieses Framework.

http://www.protractortest.org/

8.9.2.5 Less

Less ist eine Stylesheet-Sprache mit dem Ziel, das Schreiben von CSS effizienter zu gestalten. (Less (Stylesheet-Sprache) 2016)

http://lesscss.org/

9 Usability-Test

Benutzer des Diff Viewers sollen auf einen Blick erkennen können, was sich zwischen zwei Builds geändert hat. Ein effizienter Vergleich zweier Builds kann allerdings nur gewährleistet werden, wenn der Diff Viewer für jedermann verständlich und leicht bedienbar ist. Aus diesem Grund führen wir einen Usability Test durch, um zu prüfen, wie wir den Diff Viewer weiter optimieren können, um dieses Ziel zu erreichen.

9.1 Testkonzept

9.1.1 Fragestellungen

Wir gestalten den Usability Test so, dass wir nach der Durchführung die nachfolgenden Fragen beantworten können.

9.1.1.1 Allgemeine Fragen

- Wie gut ist die User Experience insgesamt?
- Wie nützlich ist die Anwendung?
- Welches sind die realen Bedürfnisse und wie verhalten sich die User wirklich?
- Sind die einzelnen Bereiche gut auffindbar?
- Wissen die Testpersonen immer, wo sie sich befinden?
- Sind Abläufe, Texte und Begriffe verständlich?

(Peuker 2015)

9.1.1.2 Spezifische Fragen

- Wird den Testpersonen klar, wie sie vorgehen müssen, um einen Build zu vergleichen?
- Wird den Testpersonen klar, wie sie den Vergleichsmodus beenden können?
- Wird den Testpersonen klar, dass sie nach dem Änderungsgrad sortieren können?
- Können die Testpersonen feststellen wo die grössten Unterschiede zwischen zwei Builds sind?
- Können die Testpersonen feststellen wo die grössten Unterschiede zwischen zwei Screenshots sind?
- Verstehen die Testpersonen, was die Farben bei den DiffIcons bedeuten?
- Verstehen die Testpersonen, wann ein Use Case/Scenario/Step neu oder gelöscht ist?

9.1.2 Testbereiche

Es werden ausschliesslich die Funktionen des Diff Viewers getestet.

9.1.3 Testpersonen

Die Firma Zühlke rekrutierte vier Personen für den Usability-Test. Gemäss einer Studie der Nielsen Norman Group sollten wir mit dieser Anzahl ungefähr 75% der Usability-Probleme aufdecken können. (Nielsen 2000) Jeder dieser vier Personen nutzt Scenarioo in seinem Arbeitsalltag. Dadurch können wir sichergehen, dass die grundlegende Bedienung von Scenarioo keine zusätzlichen Unklarheiten hervorruft.

Die Personen waren sowohl Nutzer von grösseren Endkunden als auch Mitarbeiter von Zühlke, welche Scenarioo in ihren Projekten einsetzen.

9.1.4 Softwarestand

Für den Usability-Test steht der Diff Viewer im Status „Feature Complete“ zur Verfügung. Das bedeutet, der Softwarestand enthält alle User Stories der höchsten Priorität.

9.1.5 Testdaten

Die Testdaten bestehen aus produktiven Daten eines grossen Unternehmens, welches Scenarioo schon über mehrere Jahre hinweg im Einsatz hat. Der Datensatz umfasst über 220 Szenarios, welche entsprechend viele Steps aufweisen. Es ist ein grosser Vorteil, dass wir diesen Datensatz nutzen können. Erstens fühlt es sich für die Testpersonen authentischer an, wenn sie mit realen Daten arbeiten können. Zweitens sind darin alle möglichen Unterschiede zwischen zwei Builds enthalten.

9.1.6 Test Setup:

Für die Usability-Tests können wir einen eigens dafür eingerichteten Testraum der Firma Zühlke nutzen. Dieser Raum verfügt über einen Testcomputer mit dem Usability Tool Morae Recorder. Für den Testleiter steht im Beobachtungsraum ein zweiter Computer mit dem Morae Observer bereit sowie einem Beamer für die Beobachter. Morae macht dabei ein Livestreaming von Bildschirm und Kamerabild via Netzwerk auf den Observer.

9.1.7 Test Aufgaben:

Der Testbenutzer soll sich in die Rolle eines Solution Designers hineinversetzen. Er soll besonders nennenswerte Änderungen seit dem letzten Release protokollieren. Dazu wurden ihm mehrere Teilaufgaben gestellt. Die Aufgaben sind so ausgelegt, dass wir während dem Bobachten die oben genannten Fragestellungen beantworten können. Im Anhang befindet sich die komplette Aufgabenstellung, welche an die Testpersonen verteilt wurde.

9.2 Testdurchführung

Um beurteilen zu können, wie gut sich die Testperson mit Scenarioo auskennt, wurde ein Vorinterview durchgeführt. In diesem Vorinterview wurden auch demographische Daten sowie der Verwendungszweck von Scenarioo aufgenommen.

Nachfolgend wurde der Usability-Test in der oben genannten Form durchgeführt. Die Testpersonen wurden gebeten, ihre Vorgehensweise zu kommentieren sowie Gedanken laut auszusprechen. Das hat sehr gut funktioniert. Es war spannend zu hören, welche Gedanken sich die Personen gemacht haben. Beispielsweise wurde oft kommentiert, wieso gewisse Erwartungen nicht eingetroffen sind.

Das Setup von Morae hat – nach Startschwierigkeiten im Netzwerkbereich – hervorragend funktioniert. Die Testpersonen fühlten sich dank der Raumtrennung unbeobachtet und liessen sich nicht aus der Ruhe bringen, wenn etwas nicht auf Anhieb funktionierte. Ausserdem waren die Aufzeichnungen nützlich für die spätere Auswertung.

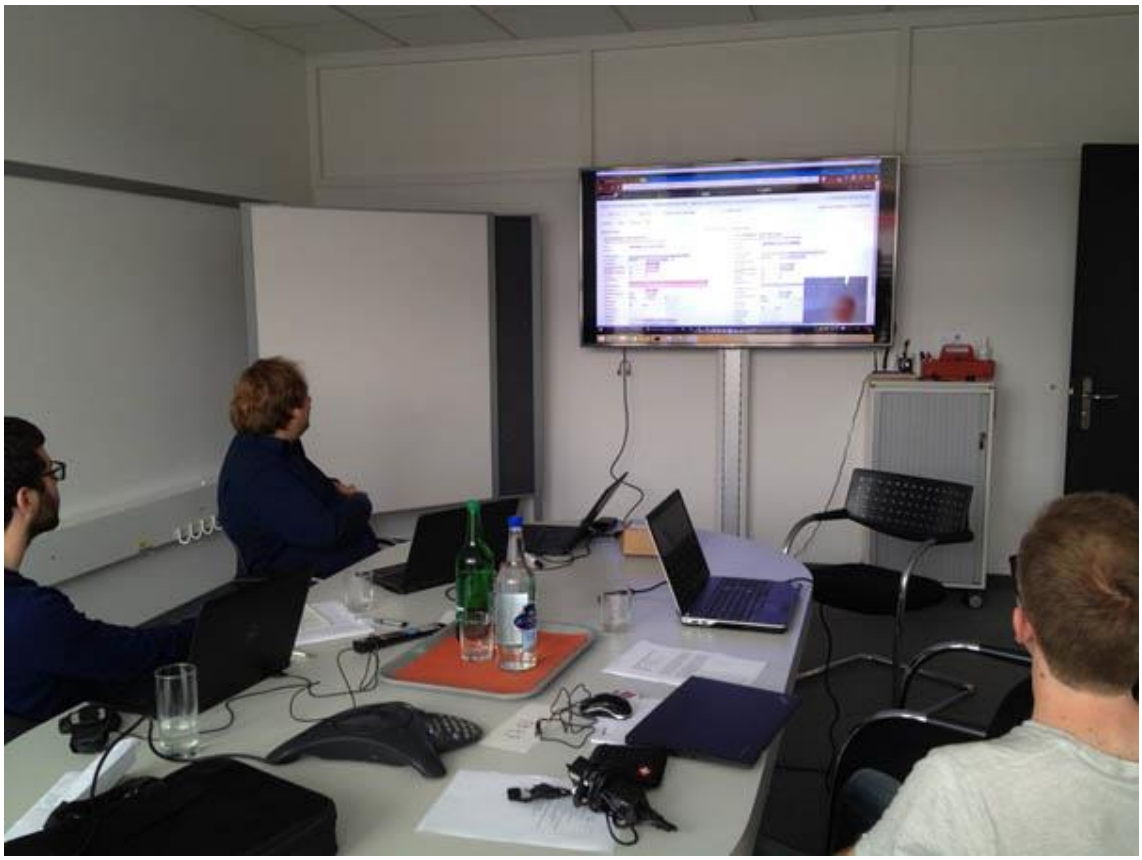


Abbildung 47: Beobachtungsraum während dem Usability-Test

Als Abschluss wurde ein Nachinterview mit der Testperson durchgeführt. Es wurden konkrete Probleme besprochen, aber auch Fragen zur allgemeinen User Experience gestellt.

Der Fragebogen vom Vor- und Nachinterview befindet sich im Anhang.

9.3 Testergebnisse

Der Usability Test war sehr aufschlussreich. Die grösste Überraschung für uns war, dass das Auswahlmenu für die comparisons von allen Testpersonen zuerst übersehen wurde. Zusammengefasst gibt es einige Stellen, an denen wir die Usability mit wenig Aufwand erheblich verbessern können. Diese Erkenntnisse sind im Kapitel Designkonzept im Abschnitt Anpassungen nach Usability-Tests dokumentiert.

10 Projektmanagement

10.1 Rollen und Verantwortlichkeiten

10.1.1 Prof. Dr. Markus Stolze

Prof. Dr. Markus Stolze, Institutspartner IFS und Studiengangleiter Informatik sowie Dozent an der HSR, hat in der Arbeit die Funktion als Betreuer inne.

10.1.2 Daniel Suter

Daniel Suter ist Expert Software Engineer bei der Firma Zühlke und Mitentwickler des Open Source Projektes Scenarioo.

Er ist Hauptansprechpartner auf Seite vom Industriepartner und übernimmt im Rahmen der Arbeit die Verantwortung für Software Architektur, technische Umsetzung und Qualitätskontrolle.

10.1.3 Rolf Bruderer

Rolf Bruderer ist Lead Software Architekt bei der Firma Zühlke und Leiter des Scenarioo-Entwicklungsteams.

Er übernimmt im Projekt die Rolle des Product Owners, als solcher ist er verantwortlich für Produkt-Design-Entscheide und Abklärungen bezüglich Anforderungen auf Seiten des Industriepartners. Ausserdem unterstützt er Daniel Suter bei der Betreuung der Arbeit, auch als Stellvertreter in technischen Fragen.

10.1.4Manuel Scheuber

Manuel Scheuber ist Teilzeitstudent an der HSR und arbeitet nebenbei als System Administrator bei der Firma Columba Informatik AG. Er ist Teil des Entwicklerteams und übernimmt zusätzlich die Rolle des Scrum Masters.

10.1.5Pascal Forster

Pascal Forster arbeitet nebst seinem Teilzeitstudium an der HSR drei Tage in der Woche als Software Entwickler bei der Firma Namics AG. Er ist Teil des Entwicklerteams.

11 Prozessmodell

Der Projektantrag definiert als Rahmenbedingung, dass die Projektarbeit iterativ absolviert werden soll. Wir erfüllen diese Rahmenbedingung, indem wir die Arbeit nach dem Prozessmodell Scrum umsetzen.

Scrum ist ein agiles, iteratives Modell, welches durch seine Flexibilität überzeugt. Ziel unserer Arbeit ist es, eine produktiv einsetzbare Erweiterung vom Open Source Projekt Scenarioo im Sinne eines Minimum Viable Product zu erarbeiten.

Das von uns gewählte Prozessmodell erlaubt es uns, den Funktionsumfang der Erweiterung während dem Projektverlauf stets neu zu priorisieren und zu definieren.

Scrum basiert auf den vier agilen Grundsätzen (Agilemanifesto kein Datum):

- Individuals and interactions over processes and tools
- Working software over comprehensive documentation
- Customer collaboration over contract negotiation
- Responding to change over following a plan

Da die Bachelorarbeit zu einem grossen Teil auch aus Dokumentation besteht, wird der zweite Punkt vernachlässigt.

11.1.1 Rollen

Tabelle 19: Scrum Rollen

Rolle	Beschreibung und Aufgaben
Product Owner	• Ansprechperson auf Kundenseite • Product Backlog pflegen • Anforderungen priorisieren
Scrum Master	• Scrum Prozess überwachen • Informationsaustausch zwischen Product Owner und Team koordinieren • Daily Meetings moderieren
Team	• Selbstorganisierte Gruppe • Schätzung der Anforderungen • Umsetzung der Anforderungen

(Rollen Team 2015)

11.1.2 Artefakte

Tabelle 20: Scrum Artefakte

Artefakt	Beschreibung
Product Backlog	Im Product Backlog befinden sich alle noch nicht umgesetzten User Stories. Vor Beginn eines neuen Sprints werden die offenen User Stories vom Product Owner neu priorisiert und der Aufwand vom Team neu geschätzt.

Sprint Backlog	Im Sprint Backlog befinden sich die in einem Sprint umzusetzenden User Stories. Hierbei ist es wichtig, dass das Team die Menge der Aufgaben bestimmen kann.
Burndown Charts	Pro Sprint wird ein Burndown-Chart erstellt. Dieses vergleicht die Ist- und Sollstunden und ermöglicht es, Zeitprobleme zu erkennen.

(Artefakte Reporting kein Datum)

11.1.3 Sprints

In Scrum werden die Iterationen Sprint genannt und dauern in der Regel 2 bis 4 Wochen. Pro Sprint wird ein Sprint Backlog erstellt. Am Ende eines Sprints sollte ein fertiges Artefakt resultieren. Dies kann ein testbarer Softwarestand oder auch ein Teil der Dokumentation sein.

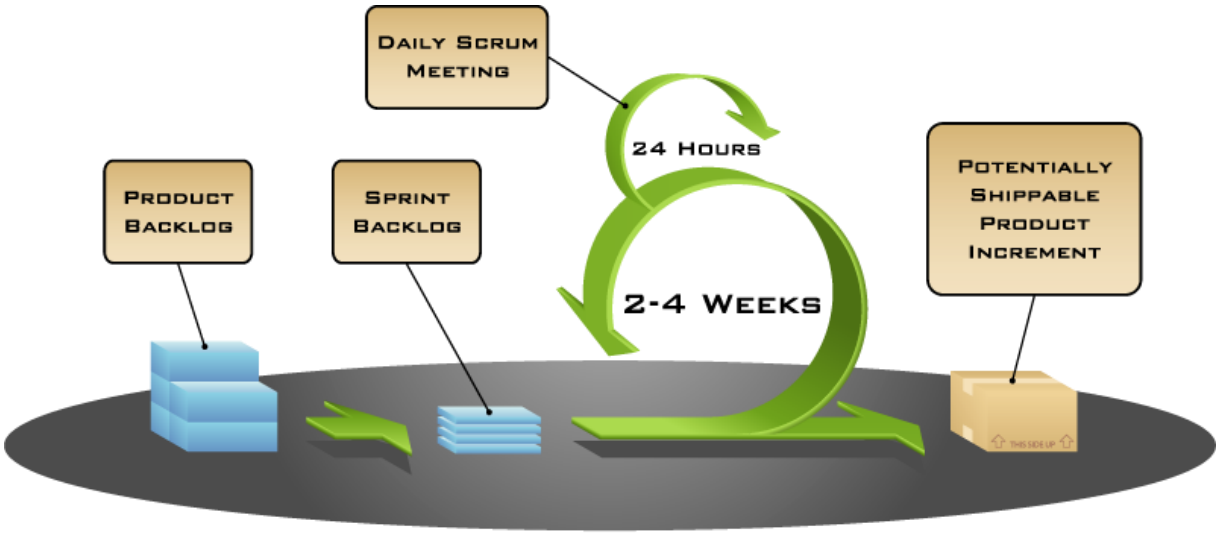


Abbildung 48: Übersicht über die Sprints

(Scrum Übersicht 2016)

11.2 Aufwandschätzung, Zeitplan, Projektplan

11.2.1 Aufwandschätzung

Gemäss Modulbeschreibung wird pro Gruppenmitglied ein Gesamtaufwand von ungefähr 370 Stunden für die Bachelorarbeit gefordert.

Wir planen die ersten 13 Wochen mit einem Aufwand von 19 Stunden, gefolgt von drei Wochen mit 26 Stunden Aufwand pro Woche. Zum Abschluss der Arbeit planen wir noch eine Woche mit 45 Stunden Aufwand.

Tabelle 21: Übersichtstabelle der Aufwandschätzung

Projektdauer	17 Wochen
Anzahl Projektmitglieder	2
Aufwand pro Projektmitglied	370 Stunden (13 * 19h + 3 * 26h + 1 * 45h)
Gesamtaufwand	740 Stunden
Projektstart	22. Februar 2016
Projektende	17. Juni 2016

11.2.2 Zeitplan

Die gesamte Zeitplanung wird in der Projektmanagementsoftware Jira vorgenommen. Aufgrund des hohen Detailgrades umfasst der Zeitplan mehrere Seiten. Daher haben wir beschlossen, ihn nicht direkt in diesem Kapitel einzubauen, sondern in den Anhang einzufügen. Eine grobe Übersicht der Zeitplanung ist im nachfolgenden Kapitel enthalten. Dort ist beschrieben wie lange die jeweiligen Sprints dauern.

11.2.3 Sprints

Ein Sprint dauert in der Regel drei Wochen. Die Ausnahmen bilden die erste und die letzte Projektwoche, welche jeweils nur eine Woche dauern. In der nachfolgenden Tabelle werden die groben Ziele der einzelnen Sprints und ihre Dauer beschrieben. Zusätzlich wird der für den Sprint geplante Gesamtaufwand in Arbeitstagen angegeben. Ein Arbeitstag dauert 8 Stunden.

Tabelle 22: Grobe Sprintplanung

Sprint	Beschreibung	Periode	Soll Aufwand
Sprint 0	• Scenarioo verstehen • Kick-Off Meeting	22.02.16 – 28.02.16	4.75 AT
Sprint 1	• Projektplanung • Risikomanagement • Anforderungen erarbeiten • LowFi Prototyp • Entwicklungsumgebung	29.02.16 – 20.03.16	14.25 AT
Sprint 2	• Evaluation • Konzept (Analyse, Design) • Evaluiertes Framework prototyp-mässig testen	21.03.16 – 10.04.16	14.25 AT
Sprint 3	• Implementierung	11.04.16 – 01.05.16	14.25 AT
Sprint 4	• Implementierung • Usability Test	02.05.16 – 22.05.16	14.25 AT
Sprint 5	• Bugfixing • UI-Optimierung	23.05.16 – 12.06.16	19.50 AT

Sprint 6	• Projekt fertigstellen • Abgabe	13.06.16 – 17.06.16	11.25 AT

11.3 Meilensteine

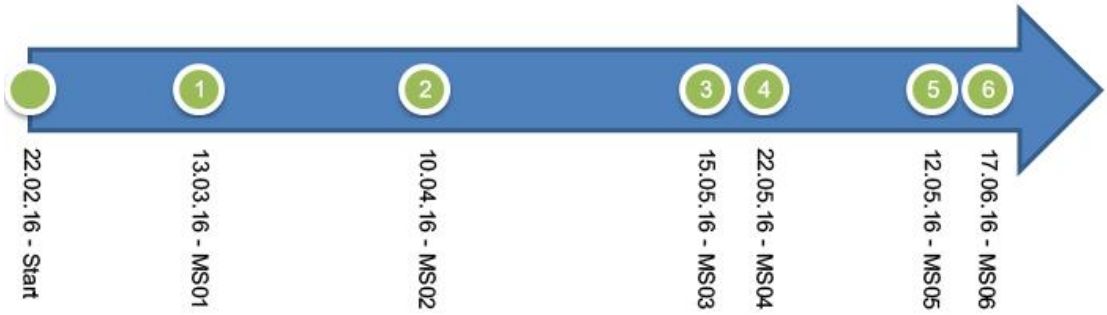


Abbildung 49: Zeitliche Übersicht über die Meilensteine

11.3.1 MS01 Projektplanung

Termin	13. März 2016
Beschreibung	Projektplanung existiert mit Meilensteinen
Work Products	Projektplanung Meilensteine Risikoabschätzung

11.3.2 MS02 Konzept

Termin	10. April 2016
Beschreibung	Es existiert eine klare Vorstellung, wie die gewünschte Funktionalität umgesetzt werden kann
Work Products	Benutzungs- und Architekturkonzept Frameworks sind evaluiert und technisch geprüft

11.3.3 MS03 Feature Complete

Termin	15. Mai 2016
Beschreibung	Applikation enthält alle geplanten oder primären Funktionen, so dass diese grundsätzlich im Usability-Test benutzt werden können. Kann aber noch nicht verhindernde Bugs enthalten.
Work Products	Applikation mit umgesetzten Features

11.3.4 MS04 Usability Test

Termin	22. Mai 2016

Beschreibung	Usability-Test wurde durchgeführt, die Resultate ausgewertet und priorisiert
Work Products	Testprotokolle
	Überarbeitete Anforderungen

11.3.5 MS05 Code Complete

Termin	12. Juni 2016
Beschreibung	Höchst priorisierte Bugs und Findings aus Usability-Tests behoben.
Work Products	Release Candidate 1.

11.3.6 MS06 Abgabe

Termin	17. Juni 2016
Beschreibung	BA mit allen geforderten Dokumenten und Artefakten abgegeben.
Work Products	Fertige BA
	Falls zeitlich möglich, Github-Fork zurück gemerged.

11.3.7 Einhaltung der Termine

Die Termine von den Meilensteinen wurden eingehalten. Dies wird auch im Abnahmeprotokoll durch Zühlke bestätigt. Einzig über den Meilenstein Code Complete lässt sich streiten. Denn in der letzten Woche haben wir noch kleinere Refactorings vorgenommen, die uns vorher nicht bekannt waren. Gründe dafür waren das Code Review von Michael Gfeller und ein kleiner Änderungswünsch von Zühlke.

11.4 Sprints

11.4.1 Sprint 0

11.4.1.1 Summary

Periode	22.02.16 – 28.02.16
Stunden Soll	38 Stunden
Stunden Plan	20 Stunden
Stunden Ist	22 Stunden

11.4.1.2 Burndown Chart

-

11.4.1.3 Ziele

- Scenarioo verstehen
- Kick-Off Meeting

11.4.1.4 Abgeschlossen

Key	Summary
SCENARIOO-64	Einarbeitung Open Source Projekt Scenarioo
SCENARIOO-65	Sitzungen Sprint 0

11.4.1.5 Probleme

Da wir zu Beginn des Projektes noch nicht wussten, welches Projektmanagement Tool wir benötigen, konnten wir in der ersten Woche noch keine Jira-Tasks erfassen. Dies wurde aber nachgetragen und stellt somit kein Problem mehr dar für die Zukunft.

11.4.2Sprint 1

11.4.2.1 Summary

Periode	29.02.16 – 20.03.16
Stunden Soll	114 Stunden
Stunden Plan	101 Stunden
Stunden Ist	101.5 Stunden

11.4.2.2 Burndown Chart

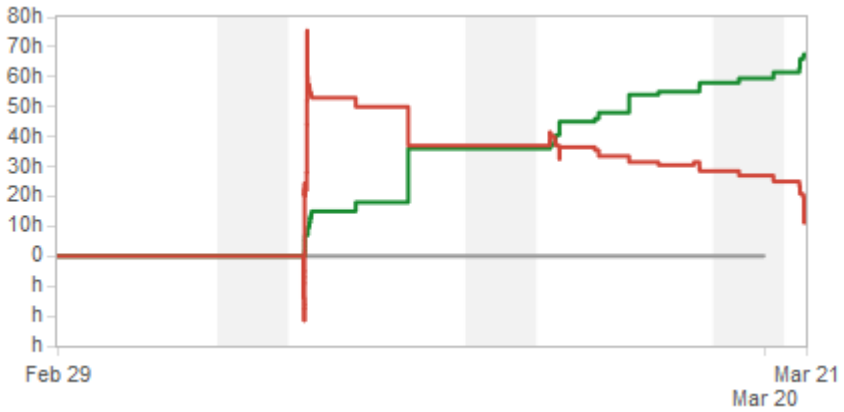


Abbildung 50: Burndown Chart Sprint 1

11.4.2.3 Ziele

- Projektplanung
- Risikomanagement
- Anforderungen erarbeiten
- LowFi Prototyp

11.4.2.4 Abgeschlossen

Key	Summary
SCENARIOO-5	Einführung Technischer Bericht
SCENARIOO-6	Stand der Technik
SCENARIOO-11	Anforderungsspezifikation
SCENARIOO-15	Projektmanagement
SCENARIOO-57	Projektstruktur und Dokumentenlayout
SCENARIOO-59	Definition of Done
SCENARIOO-60	Einverständniserklärung schreiben
SCENARIOO-61	Jira vorbereiten
SCENARIOO-62	Interview vorbereiten
SCENARIOO-66	Sitzungen Sprint 1
SCENARIOO-70	Low-Fi Prototyp
SCENARIOO-71	Meilenstein- und Sprintplanung anpassen
SCENARIOO-72	Entwicklungsumgebung einrichten

11.4.2.5 Probleme

In diesem Sprint haben wir die ersten Erfahrungen mit dem Agile Board von Jira gesammelt. Dies führte dazu, dass wir mehr Zeit als erwartet mit dem Sprint Management verbraucht haben.

Ebenso mussten wir feststellen, dass Jira in der Standardversion nicht geeignet ist für eine saubere Zeiterfassung und für das generieren von Reports. Deshalb führen wir nebst dem Jira noch ein Google Doc für die Zeiterfassung.

11.4.3 Sprint 2

11.4.3.1 Summary

Periode	21.03.16 – 10.04.16
Stunden Soll	114 Stunden
Stunden Plan	107 Stunden
Stunden Ist	115.25

11.4.3.2 Burndown Chart

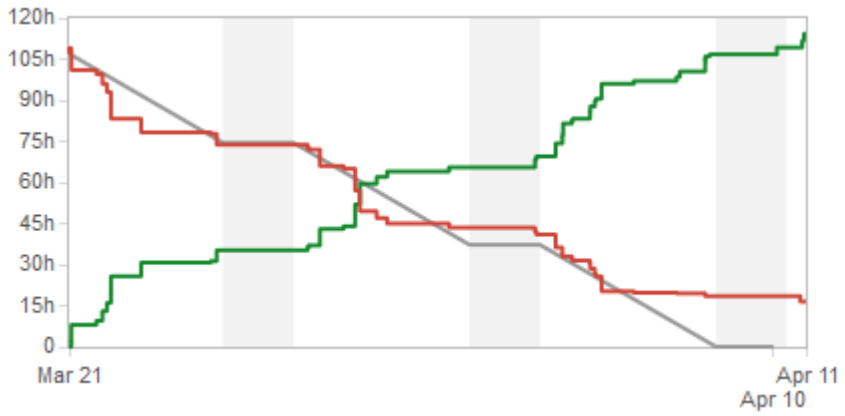


Abbildung 51: Burndown Chart Sprint 2

11.4.3.3 Ziele

- Benutzungs- und Architekturkonzept
- Frameworks sind evaluiert
- Konzept technisch geprüft

11.4.3.4 Abgeschlossen

Key	Summary
SCENARIOO-84	Aufbau Testumgebung mit richtigem Buildvorgang
SCENARIOO-83	Buildvergleich Konzept erarbeiten
SCENARIOO-82	Framework: Technisch prüfen
SCENARIOO-81	User Stories Übersichtsgrafik
SCENARIOO-80	Besprechung User Stories, UI-Prototyp
SCENARIOO-78	Proof of Concept
SCENARIOO-77	Scenarioo Aufbau anschauen
SCENARIOO-75	Sitzungen Sprint 2
SCENARIOO-74	Detaillierte Risikobewertung
SCENARIOO-73	Abschluss Sprint 1
SCENARIOO-35	Nicht-Funktionale Anforderungen
SCENARIOO-13	Design
SCENARIOO-12	Analyse
SCENARIOO-7	Bewertung (Evaluation)

11.4.3.5 Probleme

In diesem Sprint traten keine nennenswerten Probleme auf.

11.4.4Sprint 3

11.4.4.1 Summary

Periode	11.04.16 – 01.05.16
Stunden Soll	114 Stunden
Stunden Plan	110 Stunden
Stunden Ist	131.25 Stunden

11.4.4.2 Burndown Chart

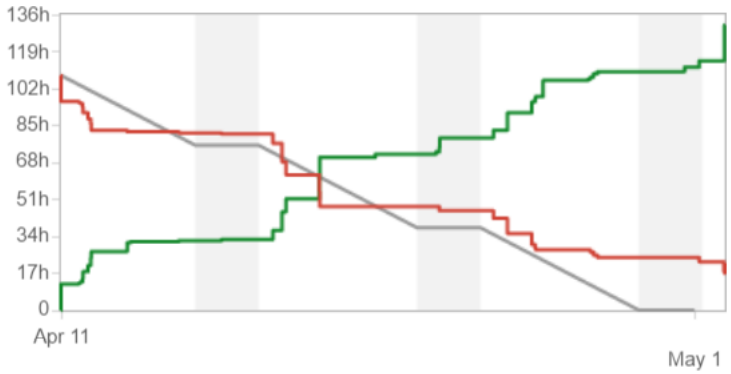


Abbildung 52: Burndown Chart Sprint 3

11.4.4.3 Ziele

- Beginn der Implementation

11.4.4.4 Abgeschlossen

Key	Summary
SCENARIOO-4	Aufgabenstellung einfügen
SCENARIOO-86	Java 6 abklären
SCENARIOO-88	User Stories auf Englisch in Github erfassen
SCENARIOO-96	Bildvergleichtools mit verschobenem Bild testen
SCENARIOO-98	User Stories schätzen

11.4.4.5 Probleme

In diesem Sprint konnten wir relativ wenig Tasks abschliessen. Dies lag daran, dass wir uns auf die Implementierung der Tasks fokussiert haben und die Reviews auf den nächsten Sprint geschoben haben. Dies haben wir aus dem Grund so gemacht, da wir bis zum Usability Test Feature Complete sein mussten.

11.4.5Sprint 4

11.4.5.1 Summary

Periode	02.05.16 – 22.05.16
Stunden Soll	114 Stunden
Stunden Plan	132 Stunden
Stunden Ist	121 Stunden

11.4.5.2 Burndown Chart

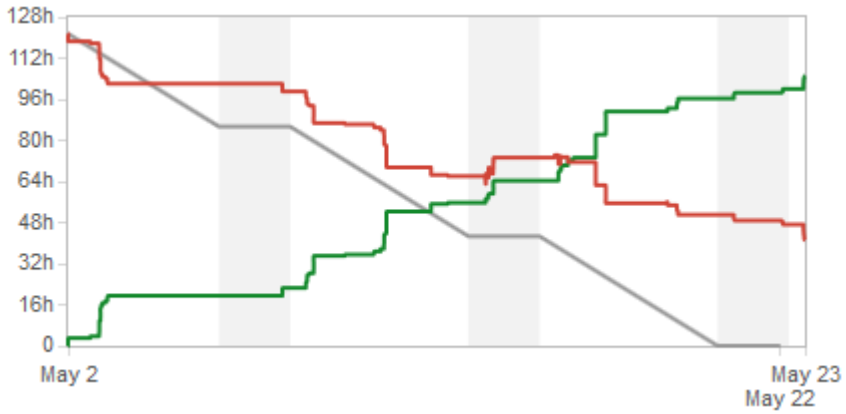


Abbildung 53: Burndown Chart Sprint 4

11.4.5.3 Ziele

- Feature Complete
- Usability Test durchgeführt und ausgewertet

11.4.5.4 Abgeschlossen

Key	Summary
SCENARIOO-134	Hauptrepo in Fork mergen
SCENARIOO-133	Scenario DiffIcon falsch wenn gelöscht/geändert
SCENARIOO-131	Step Tab leer
SCENARIOO-127	Neue Steps: Icons falsch bei Szenarioübersicht
SCENARIOO-124	Tab ohne Inhalt bei 'none' Comparison
SCENARIOO-123	Änderungsrate von Szenario stimmt nicht mit Steps überein
SCENARIOO-122	Änderungsbalken stimmt auf Use Case Übersicht nicht
SCENARIOO-120	Zwischenpräsentationen durchführen
SCENARIOO-119	Usability Test durchführen

SCENARIOO-118	Usability Test vorbereiten
SCENARIOO-117	Zwischenpräsentation vorbereiten
SCENARIOO-116	Sitzungen Sprint 4
SCENARIOO-111	Windowskompatibilität prüfen
SCENARIOO-110	User Story 3.2: Screenshot-Vergleich
SCENARIOO-109	User Story 3.1: Step Änderungsübersicht
SCENARIOO-108	Auswertung Usability Tests
SCENARIOO-107	User Story 2.8: Nach Änderungsgrad sortieren
SCENARIOO-105	User Story 2.6: Szenarioübersicht: Step Änderungen erkennen
SCENARIOO-104	User Story 2.5: Szenarioübersicht: Page Änderungen erkennen
SCENARIOO-103	User Story 2.4: Szenario Änderungsdetails anzeigen
SCENARIOO-102	User Story 2.3: Szenario Änderungen erkennen
SCENARIOO-101	User Story 2.2: Use Case Änderungsdetails anzeigen
SCENARIOO-100	User Story 2.1: Use Case Änderungen erkennen
SCENARIOO-99	Strukturvergleich implementieren
SCENARIOO-93	User Story 1.5: Vergleichs-Modus beenden
SCENARIOO-92	User Story 1.3: Comparable-Build auswählen
SCENARIOO-91	User Story 1.2: Latest Build als Vergleich-Build auswählen
SCENARIOO-90	User Story 1.1: Vergleich-Builds konfigurieren
SCENARIOO-87	Änderungsgrad definieren

11.4.5.5 Probleme

Keine grösseren Probleme in diesem Sprint.

11.4.6 Sprint 5

11.4.6.1 Summary

Periode	23.05.16 – 12.06.16
Stunden Soll	156 Stunden
Stunden Plan	160 Stunden
Stunden Ist	130 Stunden

11.4.6.2 Burndown Chart

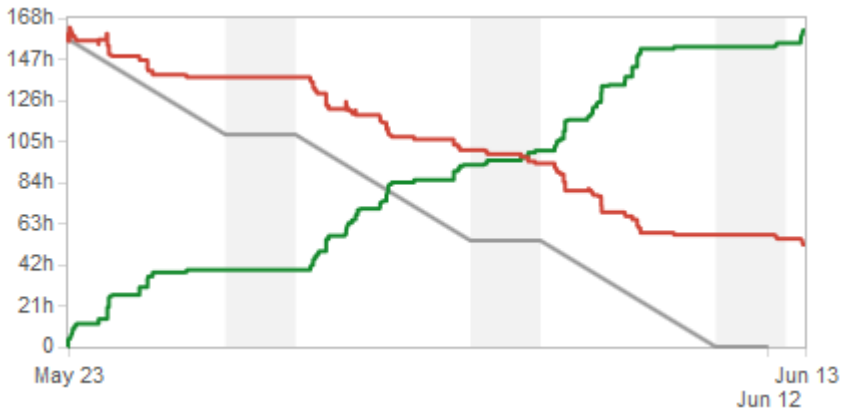


Abbildung 54: Burndown Chart Sprint 5

11.4.6.3 Ziele

- Code Complete

11.4.6.4 Abgeschlossen

Key	Summary
SCENARIOO-189	Änderungen Step Ansicht gemäss Input Rolf
SCENARIOO-187	Show/Hide Highligts Button Step Ansicht: Highlights klein schreiben
SCENARIOO-175	DiffScreenshots werden unter Windows nicht angelegt.
SCENARIOO-174	Bei neuen Steps können keine Diff Screenshots angezeigt werden
SCENARIOO-173	BuildDiffInfo erweitern
SCENARIOO-168	Comparison Alias umbenennen
SCENARIOO-167	Stepindex von gelöschten Steps
SCENARIOO-166	E2E Tests für Use Case und Szenarioübersicht
SCENARIOO-165	Sprintplanung 5
SCENARIOO-164	Sitzungen Sprint 5
SCENARIOO-163	E2E Testdaten erstellen
SCENARIOO-161	Code Review mit Michael Gfeller organisieren
SCENARIOO-159	Tooltips mit Farblegende versehen
SCENARIOO-158	Diff Screenshot Anordnung

SCENARIOO-156	DiffIcon added/removed überarbeiten
SCENARIOO-154	Comparison Menu: none --> available n
SCENARIOO-153	Vergleichs-Modus beenden: Review und Frontend Tests
SCENARIOO-151	Szenarioübersicht: Page Änderungen erkennen: Review und Frontend Tests
SCENARIOO-150	Abstract Online eintragen
SCENARIOO-149	Nach Änderungsgrad sortieren: Review und Frontend Tests
SCENARIOO-148	Dokumentations-Struktur mit Herr Stolze besprechen.
SCENARIOO-145	Abschluss Sprint 4 Dokumentation
SCENARIOO-144	Strukturvergleich implementieren: Refactoring und Review
SCENARIOO-143	IEEE Referenz in Doku einbauen
SCENARIOO-142	Gelöschter Comparison Alias
SCENARIOO-141	Comparable-Build auswählen: Review und Frontend Tests
SCENARIOO-140	DiffIcon wird im Internet Explorer nicht auf der gleichen Zeile angezeigt
SCENARIOO-138	Scenarioo-Client: Step Controller Refactoring
SCENARIOO-137	Scenarioo-Client: Diff Komponenten nach John Papas Style-guide anpassen
SCENARIOO-136	Exceptions bei REST Diff Resourcen nicht abfangen
SCENARIOO-135	Überflüssige Kommentare entfernen
SCENARIOO-132	Deleted Tooltip hinter Metadaten
SCENARIOO-129	Vor dem Buildvergleich das Konfigfile neu laden
SCENARIOO-128	Last Successful und Most Recent
SCENARIOO-126	Step Ansicht neue Steps kennzeichnen
SCENARIOO-125	Falscher comparisonScreenshot wird angezeigt
SCENARIOO-121	REST URLs anpassen
SCENARIOO-97	Datenstruktur detailliert spezifizieren
SCENARIOO-79	UI Prototyp dokumentieren
SCENARIOO-56	Accessibility
SCENARIOO-44	Testverfahren beschreiben

SCENARIOO-17	Softwaredokumentation
SCENARIOO-3	Management Summary
SCENARIOO-2	Abstract

11.4.6.5 Probleme

Es sind keine Probleme während diesem Sprint aufgetreten.

11.4.7 Sprint 6

11.4.7.1 Summary

Periode	13.06.16 – 17.06.16
Stunden Soll	90 Stunden
Stunden Plan	124 Stunden
Stunden Ist	

11.4.7.2 Burndown Chart

Burndown Chart konnte nicht generiert werden, da der Sprint noch nicht abgeschlossen ist.

11.4.7.3 Ziele

- Fertige Bachelorarbeit

11.4.7.4 Abgeschlossen

Konnten nicht eingefügt werden, da der Sprint noch nicht abgeschlossen ist

11.4.7.5 Probleme

Die Abnahmephase war ein bisschen hektisch, da wir keinen klaren Termin mit Zühlke für die Abnahme vereinbart haben.

11.5 Infrastruktur

Einen eigenen Server haben wir für diese Bachelorarbeit nicht betrieben. Der Buildserver wurde uns von Zühlke bereitgestellt. Die Jira Projekt Management Software wurde bei Namics gehostet.

11.6 Entwicklungswerkzeuge

Zweck	Tool	Version
IDE Backend	Eclipse	Kepler Service Release 2
IDE Frontend	Webstorm	2016.1.3
Versionierungssystem	Git (Github)	2.5.3
Projektmanagementtool	Jira	7.1.4
Bildbearbeitung	Adobe Photoshop	13.0
Dateiablage	Dropbox	4.4.29
Build Management Automatisierung	Gradle	3.7.3
Buildtool Frontend	Gulp	3.9.1
Package Manager	npm	2.15.1

11.7 Qualitätskontrollen

Um die Qualität der abgeschlossenen Aufgaben zu gewährleisten wird jeweils ein Review durch die andere Person durchgeführt. Das Feedback erfolgt jeweils mündlich. Ohne Feedback darf ein Task nicht als geschlossen markiert werden. Ausserdem wird die IST und SOLL Zeit des Tasks verglichen. Weichen diese Zeiten zu sehr voneinander ab, wird intern diskutiert woran das liegt. So können wir bei späteren Aufgaben genauer schätzen und erreichen so eine kontinuierliche Verbesserung unserer Prozesse.

11.7.1 Definition of Done

Code:

- Code eingecheckt?
- Anforderungen erfüllt?
- Implementierung getestet?
- Unit-Test geschrieben?
- Sämtliche Tests sind Fehlerfrei?
- Build läuft durch?
- Review durchgeführt?
- Review dokumentiert?
- Coding Guidelines eingehalten?
- Code Patterns sinnvoll eingesetzt und dokumentiert?
- Code stimmt mit Architektur überein?
- Dokumentation ergänzt?

Dokumentation:

- Formatierung geprüft?
- Review durchgeführt?
- Review dokumentiert?
- Teildokument in Gesamtdokument eingefügt?
- Versionsnummer in Gesamtdokument erhöht?
- Fachbegriffe in Glossar erläutert?
- Abkürzungen ins Abkürzungsverzeichnis eingetragen?

11.8 Programmierrichtlinien

Die Programmierrichtlinien sind vom Open Source Projekt Scenarioo vorgeschrieben und auf dem entsprechenden GitHub Repository dokumentiert.

https://github.com/scenarioo/scenarioo/wiki/Coding-guidelines

11.9 Risiken

Barry Boehm hat in seiner Arbeit „A Prioritized Top-Ten List of Software Risk Items“ zehn Risiken aufgelistet, aufgrund derer Softwareprojekte grösstenteils scheitern. Jene Risiken auf dieser Liste, welche ein grosses Schadenspotential oder eine hohe Eintrittswahrscheinlichkeit für unsere Arbeit darstellen, haben wir genauer analysiert und beschrieben. Ergänzend haben wir weitere Risiken aufgelistet, welche wir aufgrund der Aufgabenstellung für diese Bachelorarbeit als besonders kritisch beurteilen.

(Boehm 2007)

11.9.1 Auflistung Risiken

In der nachfolgenden Tabelle haben wir die möglichen Risiken dieser Arbeit aufgelistet. Die Tabelle enthält präventive Massnahmen sowie die Massnahmen, welche wir vornehmen, falls ein Risiko eintritt. Dabei gewichten wir die Risiken nach folgender Skala:

E = Eintrittswahrscheinlichkeit
- (1) Unwahrscheinlich
- (2) Sehr selten
- (3) Selten
- (4) Möglich
- (5) Häufig

S = Schadenspotential
- (1) Unbedeutend
- (2) Gering
- (3) Spürbar
- (4) Kritisch
- (5) Projektbedrohend

Tabelle 23: Mögliche Risiken

ID	Risiko	E	S	Prävention	Massnahmen bei Eintritt
1	Unrealistischer Terminplan	4	4	Zeit zu Beginn der Arbeit gut nutzen. Klare Meilensteine definieren. Risiken fortlaufend kontrollieren und in Terminplan einrechnen.	Falls möglich den Terminplan überarbeiten. Ansonsten muss besprochen werden auf welche Features im Minimum Viable Product verzichtet werden kann.
2	Falsche Funktionalität	2	4	Detaillierte User Stories erstellen und fortlaufend mit den Projektpartnern kommunizieren. Endbenutzer von Beginn an miteinbeziehen.	Abklären welche Funktionalität tatsächlich gewünscht wird und diese so gut wie möglich in den Projektplan integrieren.
3	Unzureichende Usability	4	3	Prototypen erstellen und mit verschiedenen Benutzergruppen testen.	Problemstellen ausfindig machen und in Zusammenarbeit mit den Endbenutzern die Usability verbessern.
4	Fehlender Funktionsumfang bei Third Party Libraries	3	5	Bei der Evaluation prüfen ob die geforderten Funktionen mit der gewählten Third Party Library umgesetzt werden können.	Prüfen ob die Third Party Library in angemessener Zeit ersetzt werden kann.
5	Performanceprobleme	4	4	Bei zeitintensiven Algorithmen berechnen wie lange die Durchführung bei grösseren Datenmengen dauert.	Algorithmen falls möglich umschreiben. Andernfalls die benutzte Technologie durch eine effizientere ersetzen.
6	Einschränkung durch bestehende Systemlandschaft	2	5	Systemlandschaft frühzeitig analysieren und besonders kritische Komponenten ausfindig machen.	Einschränkende Komponenten in Zusammenarbeit mit Zühlke abändern oder ersetzen.

11.9.2Darstellung Risikomatrix

Die nachfolgende Risikomatrix bietet einen Überblick über die im Kapitel „Auflistung Risiken“ aufgelisteten Risiken. Es wird ersichtlich, dass vor allem die Risiken

- 1 - Unrealistischer Terminplan
- 4 - Fehlender Funktionsumfang bei Third Party Libraries
- 5 - Performanceprobleme

besondere Massnahmen zur Risikominimierung erfordern.

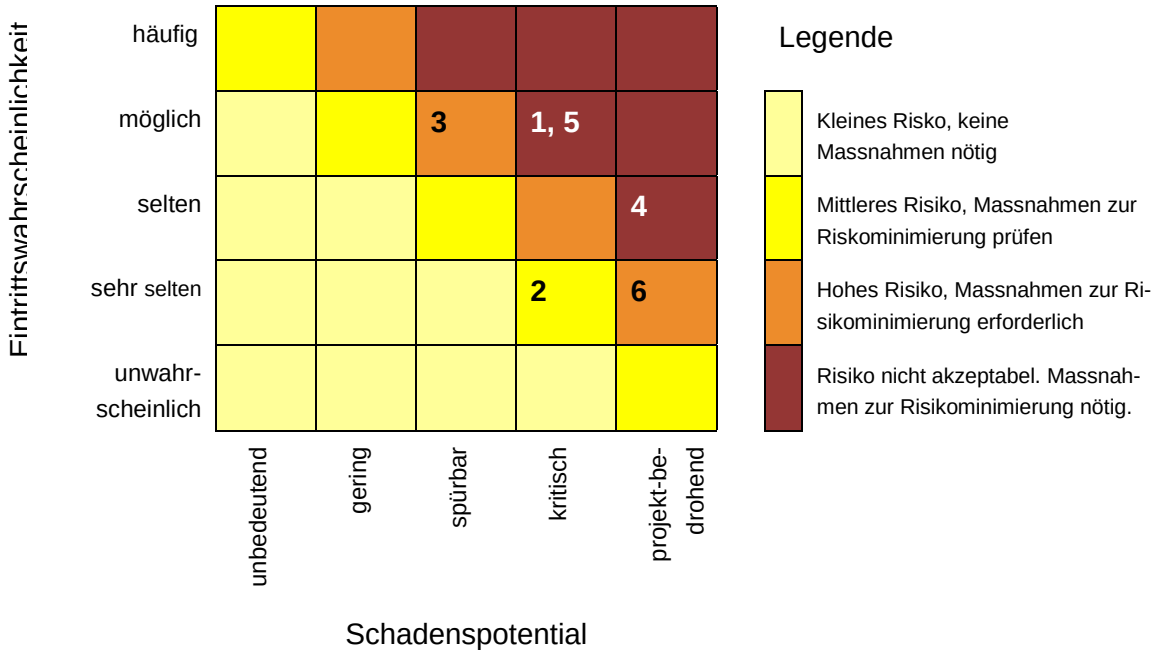


Abbildung 55: Risikomatrix

11.9.3Kritische Risiken

11.9.3.1 Risiko 1: Unrealistischer Terminplan

Eintrittswahrscheinlichkeit: Möglich **Auswirkung:** Kritisch

Beschreibung:

Im Rahmen dieser Arbeit wird das bestehende Produkt Scenarioo mit zusätzlicher Funktionalität erweitert. Es ist dabei schwierig abzuschätzen, wie viel Zeit wir für die Einarbeitung in den vorhandenen Quellcode brauchen und was für Auswirkungen selbst kleinste Änderungen haben.

Zudem müssen die genauen Anforderungen sowie das Benutzungs- und Architekturkonzept von uns erarbeitet werden. Das hat zur Folge, dass wir zu Beginn des Projektes die Entwicklungszeit nur grob schätzen können.

Ein weiterer Punkt ist die uns fehlende Erfahrung im Bereich der Bildanalyse. Hier können wir uns nur auf die Aussage von Rolf Bruderer (Scenarioo Entwickler) stützen, dass

die Umsetzung der Bildanalyse dank vorhandenen Frameworks nicht ein riesiger Aufwand sein sollte. Rolf Bruderer konnte in diesem Bereich schon Erfahrungen bei einer ähnlichen Arbeit sammeln.

Prävention:

- Frühzeitig definieren welche Funktionen das Minimum Viable Product enthalten soll
- Meilensteine sinnvoll setzen und fortlaufend kontrollieren
- Pufferzeiten einrechnen
- Projekt Monitoring mit regelmässigem Ist/Soll Vergleich

11.9.3.2 Risiko 4: Fehlender Funktionsumfang bei Third Party Libraries

Eintrittswahrscheinlichkeit: Selten **Auswirkung:** Projektbedrohend

Beschreibung:

Für den Vergleich von Printscreens und XML-Dateien werden wir mit grosser Wahrscheinlichkeit Third Party Libraries einsetzen. Sollte der Funktionsumfang dieser Libraries nicht genügen, um die geforderten Anforderungen umzusetzen, kann dies verheerende Auswirkungen haben. Sollte erst gegen Ende des Projektes klar werden, dass zwingend benötigte Funktionen mit dieser Library nicht umgesetzt werden können, wäre es zu spät, diese auszutauschen und es wäre daher nicht möglich eine lauffähige Version abzuliefern.

Prävention:

- Third Party Libraries in Zusammenarbeit mit Zühlke auswählen. Zühlke besitzt schon Erfahrungen mit Libraries dieser Art
- Benötigte Funktionen auflisten und zu Beginn testen, ob diese umgesetzt werden können
- Recherche im Internet, ob es bekannte Probleme mit dieser Third Party Library gibt

11.9.3.3 Risiko 5: Performanceprobleme

Eintrittswahrscheinlichkeit: Möglich **Auswirkung:** Kritisch

Beschreibung:

Bei Scenarioo wird im Normalfall jede Nacht ein neuer Build angelegt. Bei diesem Build werden alle erfassten Use Cases durchgetestet und dabei von jeder aufgerufenen Webseite ein Printscreen inklusive Metadaten abgespeichert. Es gibt Kunden, welche mehrere tausend Use Case Steps pro Build durchlaufen. Entsprechend lange dauert nur schon das generieren und abspeichern der Printscreens und XML-Dateien. Es kann sich dabei um mehrere Stunden handeln.

Sind alle Testergebnisse abgespeichert, muss unser Algorithmus alle Ergebnisse mit dem letzten Build vergleichen. Bei einer solchen Menge an Informationen müssen diese Algorithmen entsprechend performant sein.

Prävention:

- Klare Zieldefinition wie lange ein Build maximal dauern darf
- Bei Auswahl einer Third Party Library auf dessen Performance achten
- Bei zeitintensiven Algorithmen berechnen, wie lange die Durchführung bei grösseren Datenmengen dauert

11.9.4 Wöchentliche Risikoevaluation

11.9.4.1 Durchführung

Jede Woche führen wir eine Evaluation der Risiken durch. Dabei überprüfen wir folgende Punkte:

- Sind mögliche Massnahmen zur Risikominimierung getroffen?
- Kann die Bewertung bestehender Risiken angepasst werden?
- Sind neue Risiken aufgetaucht?

Wird ein Meilenstein oder Sprint abgeschlossen, dessen Resultate besonders viele Risiken beeinflusst, werden die Auswirkungen in diesem Kapitel dokumentieren.

11.9.4.2 Resultate

Die nachfolgenden Diagramme zeigen den Verlauf der wöchentlichen Risikoabschätzung. Um eine bessere Übersicht zu gewährleisten wurden die Risiken auf zwei Diagramme aufgeteilt.

Der Risikowert setzt sich wie folgt zusammen:

Risikowert = Eintrittswahrscheinlichkeit x Schadenspotential

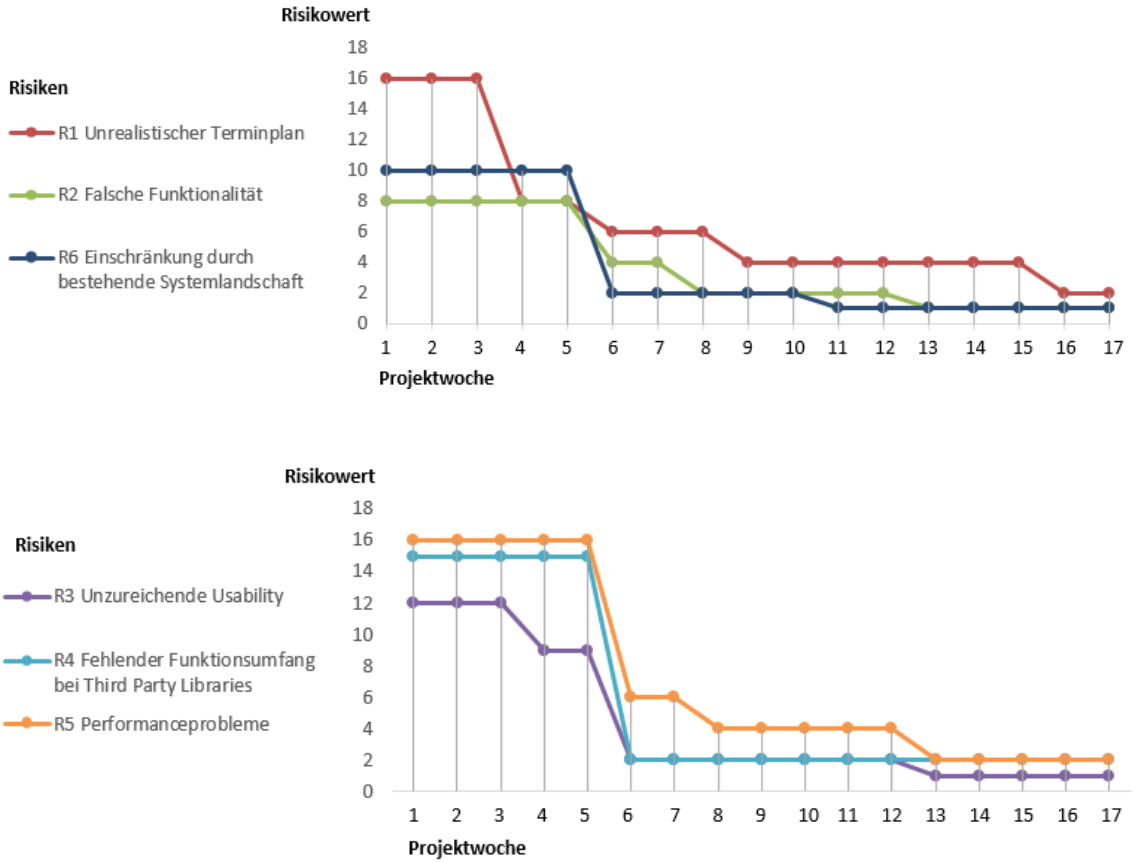


Abbildung 56: Diagramm wöchentlicher Risikoabschätzung

Nachfolgend ist beschrieben, weshalb die Risikowerte in bestimmten Wochen am meisten Sinken:

Tabelle 24: Wochen mit sinkenden Risiken

Woche	Phase	Grund
4	Abschluss Sprint 1, Meilenstein „Projektplanung“	Die Benutzerinterviews und der daraus entstandene Low-Fi Prototyp gaben uns einen Einblick, welche Anforderungen wie umgesetzt werden sollen.
6/7	Abschluss Sprint 2, Meilenstein „Konzept“	Die gefällten Architekturentscheide wurden mit einem Proof of Concept Prototyp getestet.
8/9	Sprint 3: Implementierung der User Stories	Für diesen Sprint wurden die User Stories in der Scenarioo Community diskutiert und priorisiert. Ausserdem wurden weitere Messungen zum Screenshot-Vergleich durchgeführt.
13	Sprint 4: Usability Test	Für den Usability Test konnten wir den Diff Viewer mit umfangreichen Kundendaten testen. Ausserdem konnten wir durch die Tests die Benutzerfreundlichkeit enorm erhöhen.

11.9.4.3 Risikoevaluation am Ende von Sprint 2

Im Sprint 2 haben wir diverse Architekturentscheide gefällt, durch welche wir die Risiken neu bewerten können. Ausserdem haben wir mit einem Proof of Concept Prototyp getestet, wie wir unseren Lösungsansatz in die bestehende Systemlandschaft einbinden können. Wir möchten hier die bisher ergriffenen Massnahmen zur Risikominimierung zusammenfassen.

Tabelle 25: Getroffene Massnahmen zur Risikoverminderung

ID	Risiko	Getroffene Massnahmen
1	Unrealistischer Terminplan	• Klare Meilensteine definiert • Dashboard zur fortlaufenden Zeitüberwachung eingerichtet • Priorisierung der User Stories mit dem Industriepartner • Genügend Zeit zur Fehlerbehebung und für den Abschluss der Dokumentation eingeplant
2	Falsche Funktionalität	• Benutzerinterviews mit unterschiedlichen Benutzergruppen durchgeführt • Design Prototyp sowie User Stories erstellt und diese in mehreren Sitzungen fortlaufend verbessert • Öffentliche Diskussion auf GitHub über die User Stories sowie den Design Prototyp
3	Unzureichende Usability	• Design Prototyp erstellt und diesen von UX-Spezialisten sowie Designern der Firma Zühlke prüfen lassen • Planung eines Usability Tests mit diversen Scenarioo Nutzern.
4	Fehlender Funktionsumfang bei Third Party Libraries	• Eingehende Prüfung verschiedenster Frameworks • Genauestens darauf geachtet, ob die vordefinierten Soll und Kann Kriterien vom Framework erfüllt werden
5	Performance-probleme	• Leistungstests mit Daten aus der Scenarioo Demoumgebung durchgeführt • Wahl der schnellsten Bildvergleichskomponente mit Möglichkeit zur Parallelisierung
6	Einschränkung durch bestehende Systemlandschaft	• Mehrere Konzepte erstellt und verglichen, wie die Diff-Informationen in der bestehenden XML-Struktur persistiert werden können • Die bestehende Architektur analysiert und ein Konzept geschrieben, wie unsere Lösung integriert werden könnte
-	Neue Risiken	• Wir haben uns, wie jede Woche, überlegt ob neue Risiken entstanden sind. Durch die getroffenen Architekturentscheidungen sind keine neuen Risiken aufgetaucht. Im Gegenteil. Die Wahrscheinlichkeit, dass ein bestehendes Risiko eintrifft, konnte enorm reduziert werden.

11.9.4.4 Risikoevaluation Projektende

Keines der möglichen Risiken ist eingetroffen. Die Hauptgründe dafür sehen wir wie folgt:

- Gründliche Evaluation von Third Party Libraries
- Ausführliches Benutzungs- und Architekturkonzept
- Fortwährende Kommunikation mit Zühlke
- Einbezug von Scenarioo-Usern
- Umfassende Tests im gesamten Projekt

12 Projekt Monitoring

12.1 Soll-Ist-Zeit-Vergleich

12.1.1 Wochenübersicht

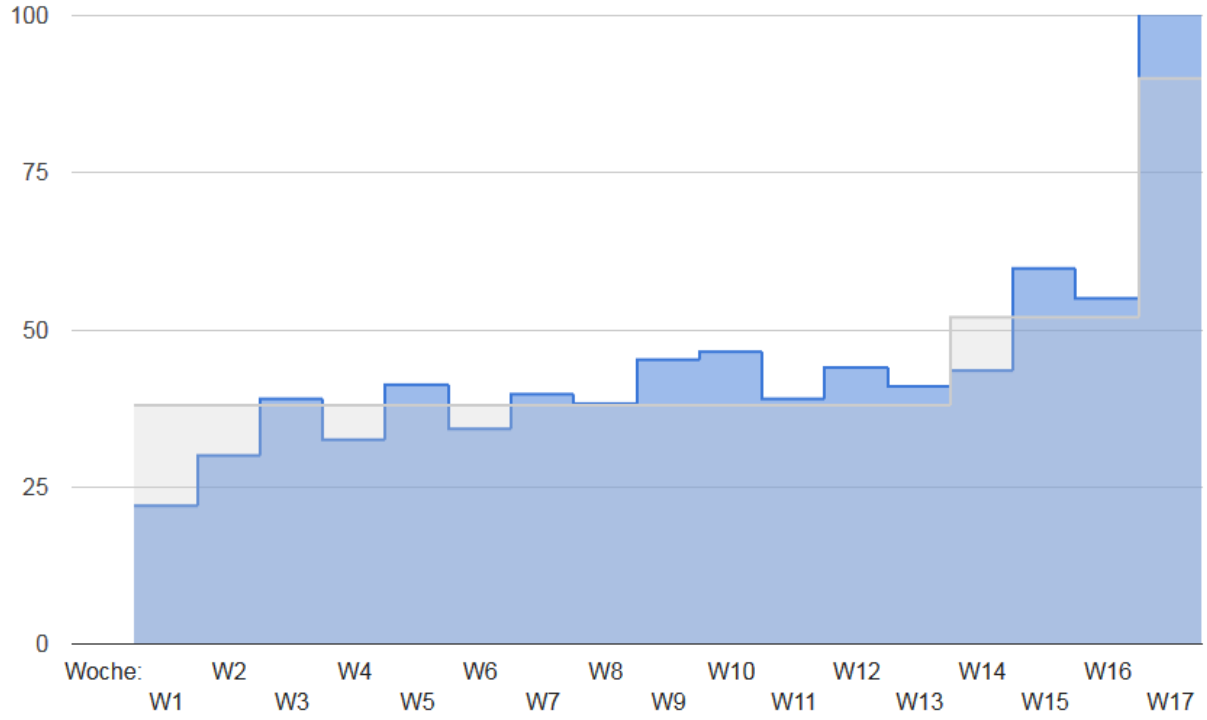


Abbildung 57: Wochenübersicht der Stunden (Soll = Hellblau, Ist = Dunkelblau)

Wie so oft hielt sich der Aufwand zu Beginn vom Projekt in Grenzen. Dies hat damit zu tun, dass wir zuerst diverse Abklärungen treffen mussten und somit nicht von 0 auf 100 loslegen konnten.

12.1.2 Stundenanteile

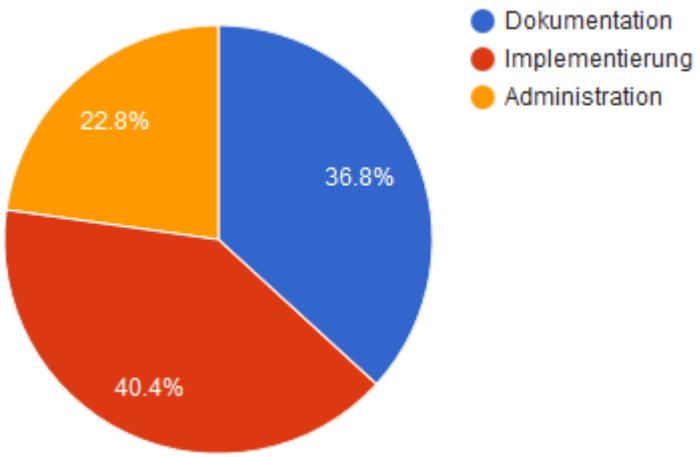


Abbildung 58: Stundenanteile

Der grösste Aufwand im Projekt betraf die Implementierung. Dicht gefolgt vom Dokumentationsaufwand. Der administrative Aufwand wäre wahrscheinlich noch etwas höher, wenn wir alle Gespräche und Diskussionen während dem Mittagessen und im Zug auch einberechnet hätten.

12.1.3 Gesamtübersicht

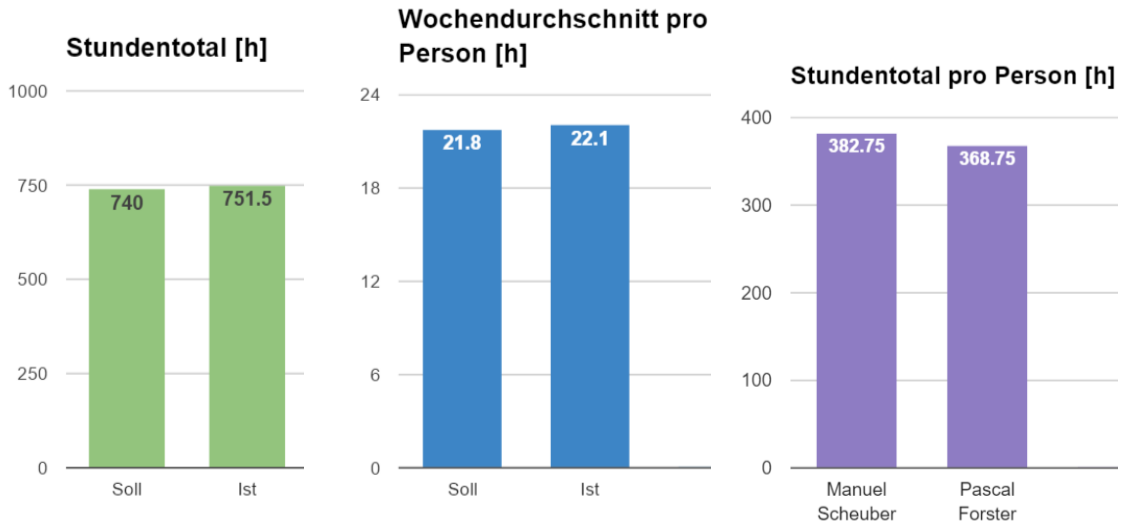


Abbildung 59: Gesamtübersicht der Stunden

Wir haben das geforderte Soll von 740 Stunden Aufwand erfüllt. Eine detaillierte Auflistung der Aufwände auf die einzelnen Jira Tasks kann dem Anhang entnommen werden.

12.1.4 Aufgewendete Stunden nach Tasks

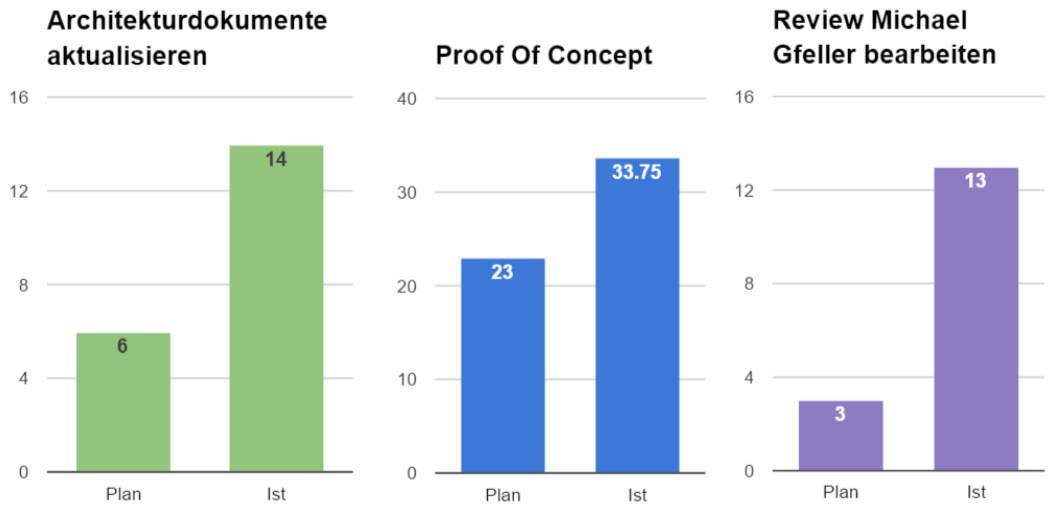


Abbildung 60: Aufgewendete Stunden nach Tasks

Bei diesen drei Tasks haben wir uns am schlimmsten verschätzt bei der Planung.

Das aktualisieren der Architekturdokumente haben wir ganz am Schluss gemacht. Wir mussten schlussendlich die Diagramme nicht nur aktualisieren, sondern auch noch neue Diagramme entwerfen. Dies kostete zusätzlich Zeit.

Der Proof Of Concept zog sich deshalb in die Länge, weil es während dieser Phase eine Änderung in der geplanten Umsetzung gab. Dieser Entscheid musste dann auch noch in den Proof Of Concept Prototyp eingebaut werden.

In der zweitletzten Woche haben wir mit Michael Gfeller noch einen Code Review gemacht. Das Feedback nahmen wir ernst und haben uns dementsprechend mühe bei der Verbesserung vom Code gegeben. In der Schätzung gingen wir davon aus, dass wir nur einen kleinen Teil vom Code einem Refactoring unterziehen. Nach jedem Refactoring sah man nochmal etwas, was man schöner machen könnte. So dauerte auch dieser Task deutlich länger als geplant.

12.2 Codestatistik

Da wir bei dieser Arbeit ein bestehendes Softwareprojekt erweitert haben, ist es nicht ganz einfach eine Codestatistik zu erstellen. Wir haben uns dabei an das Kapitel „Abgrenzung Code“ gehalten und versucht, einigermassen aussagekräftige Statistiken zu erstellen. Doch die Integration in den Bestehenden Code und die Synchronisation vom Fork mit dem Hauptrepository nach dem John Papa Styleguide Refactoring, haben uns die Auswertung erheblich erschwert.

12.2.1 Codekomplexität

Die Komplexität unseres Java Codes haben wir anhand der McCabe-Metrik berechnet. Die McCabe-Metrik berechnet sich aus der minimalen Anzahl der Testfälle, die nötig sind, um alle möglichen Stellen in einer Methode zu erreichen.

Bei gewissen Methoden sind mehrere Prüfungen nötig, um zu bestimmen, ob mit einem Alias gearbeitet wird. Diese Prüfungen erhöhen dabei die McCabe-Metrik. Im Nachfolgenden Diagramm sind die Durchschnitts- und Maximalwerte vom Diff Viewer sowie die vom gesamten Scenarioo Projekt als vergleich angegeben.

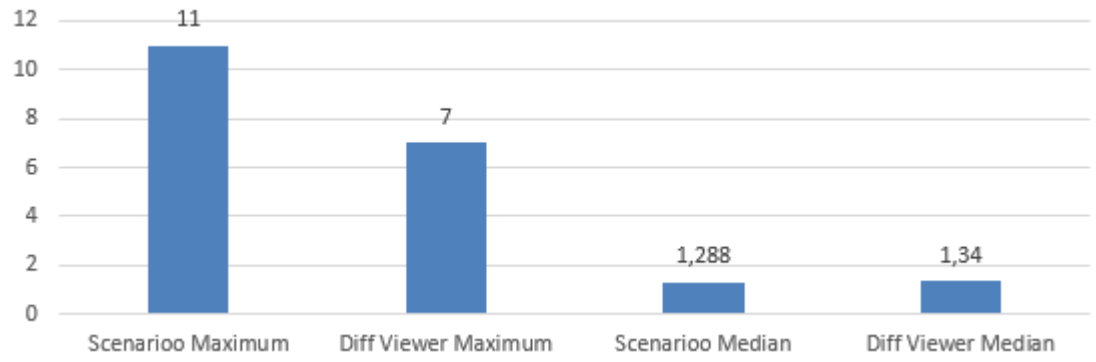


Abbildung 61: Codekomplexität nach McCabe-Metrik

Diff Viewer Maximum:

Package: org.scenarioo.business.diffViewer
Klasse: ComparisonExecutor
Methode: resolveComparisonConfiguration(comparisonConfiguration, baseBuildName)

Grund: Diese Methode wurde etwas Komplexer, da zuerst überprüft werden muss ob es sich um den lastSuccessFulAlias oder mostRecentAlias handelt.

Ansonsten hat keine von uns verfasste Methode einen Komplexitätswert in dieser Höhe erreicht.

12.2.2 Statische Code-Analyse

Die statische Code-Analyse im Frontend haben wir fortlaufend mit ESLint (http://eslint.org/) durchgeführt. Der Build-Server von Zühlke ist so konfiguriert, das die Regeln von ESLint die Grundvoraussetzung für einen erfolgreichen Build ist.

12.2.3 Anzahl Klassen

Insgesamt haben wir im scenarioo-server Projekt 23 neue Klassen angelegt. Die scenarioo-server Applikation hat total 167 Klassen.

12.2.4 Anzahl Codezeilen

12.2.4.1 Scenarioo-server

Im scenarioo-server Projekt lässt sich der von uns geschriebene Code ziemlich gut Auswerten, da er sich grösstenteils in eigenen Packages befindet. Das nachfolgende Diagramm zeigt die Anzahl Codezeilen aufgeteilt nach den Packages Main und Test. Als Vergleich wurde dargestellt wie viele Codezeilen das scenarioo-server Projekt insgesamt hat. Die Statistiken haben wir mit dem Eclipse Plugin „Metrics“ berechnet.

In Blau dargestellt:

Die Anzahl der Codezeilen innerhalb von Methoden. Kommentare und leere Zeilen werden nicht einberechnet.

In Grün dargestellt:

Die Anzahl der Codezeilen ausserhalb von Methoden. Beispielsweise fällst darunter die Methodendefinition selbst, Annotations, oder auch Klassenvariablen. Kommentare und leere Zeilen werden nicht einberechnet.

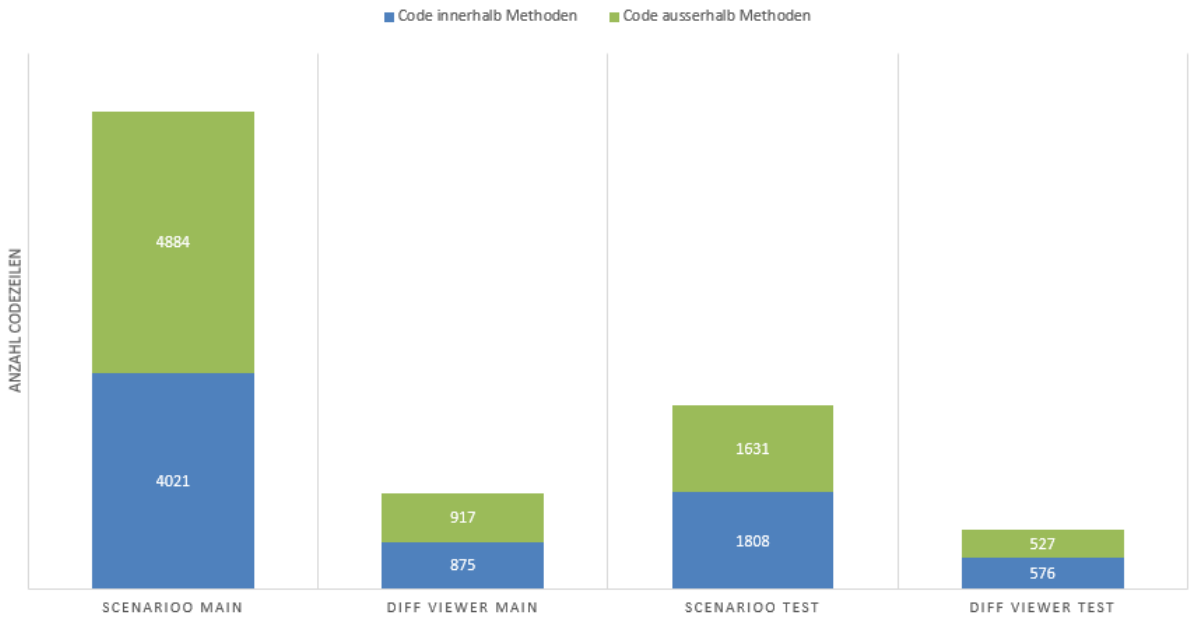


Abbildung 62: Anzahl Codezeilen scenarioo-server

12.2.4.2 Scenarioo-client

Im scenarioo-client Projekt ist von uns geschriebener Code relativ schwierig abzugrenzen. Grösstenteils haben wir die bestehenden Controller, Services und Views ergänzt. Trotzdem haben wir versucht, die Anzahl Codezeilen abzuschätzen. Dazu haben wir von einer Komponente mit wenigen Änderungen und von einer Komponente mit vielen Änderungen unsere Codezeilen herausgesucht. Diese Werte haben wir hochgerechnet, da

wir die meisten Komponenten ungefähr im gleichen Rahmen geändert oder erweitert haben. Die Statistiken haben wir mit dem Webstorm Plugin „MetricsReloaded“ berechnet.

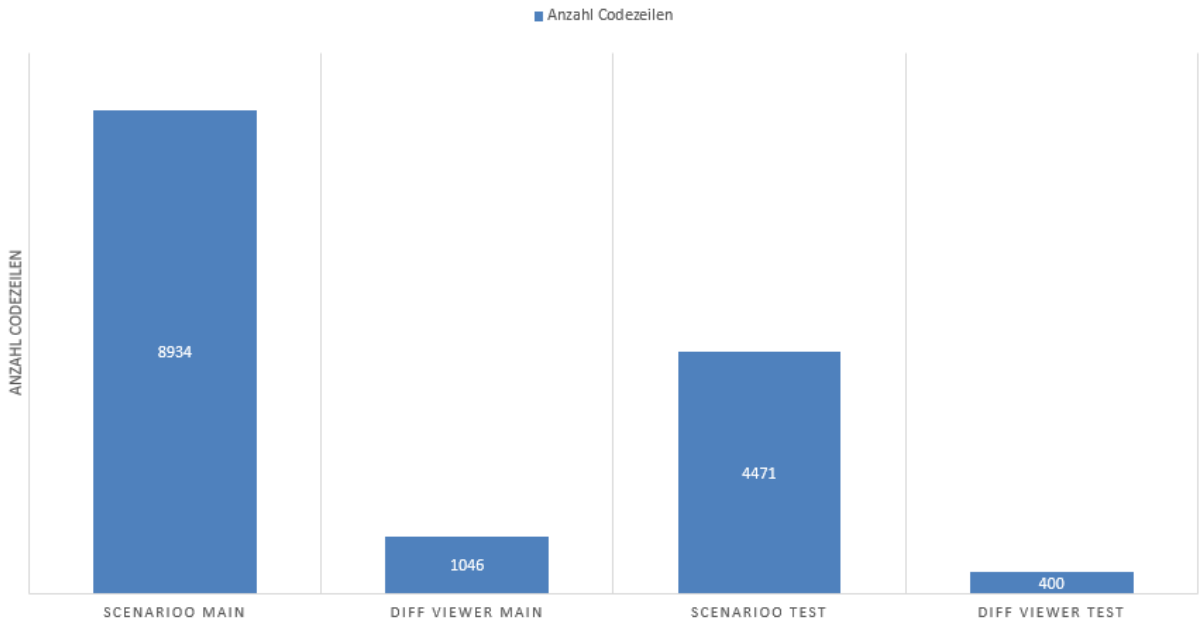


Abbildung 63: Anzahl Codezeilen scenarioo-client

12.2.5Testabdeckung

12.2.5.1 Scenarioo-server

Insgesamt hat das scenarioo-server Projekt eine Testabdeckungsrate von 39.5%. Die Abdeckung unserer Klassen ist darin einberechnet.

Die von uns geschriebenen Packages haben folgende Testabdeckung:

Tabelle 26: Testabdeckung Diff Viewer Packages

Package	Abdeckung
Org.scenarioo.rest.diffViewer	0.0%
Org.scenarioo.business.diffViewer	53.2%
Org.scenarioo.business.diffViewer.comparator	90.2%
Org.scenarioo.dao.diffViewer.impl	86.1%
Org.scenarioo.model.diffViewer	90.2%

Das REST Package wurde im Backend nicht getestet, da die Aufrufe über die End-to-End Tests geprüft werden. Org.scenarioo.business.diffViewer hätte eine Testabdeckung

von 70%. Doch weil nicht der komplette ComparisonExecutor getestet wird, ist der gemessene Wert lediglich 53%.

Die Testabdeckung haben wir mit EclEmma (www.eclemma.org) analysiert.

12.2.5.2 Scenarioo-client

Im scenarioo-client haben wir sämtliche Funktionen mittels End-to-End Tests überprüft. Wir haben kein Tool gefunden, welches uns den dadurch abgedeckten Code ermittelt. Das Resultat dieser End-to-End Test ist allerdings online Verfügbar:

http://demo.scenarioo.org/scenarioo-fork-diffviewer/#/?branch=scenarioo-self-docu&build=last%20successful&comparison=Disabled&tab=usecases

Anhang

A: Inhalt der CD

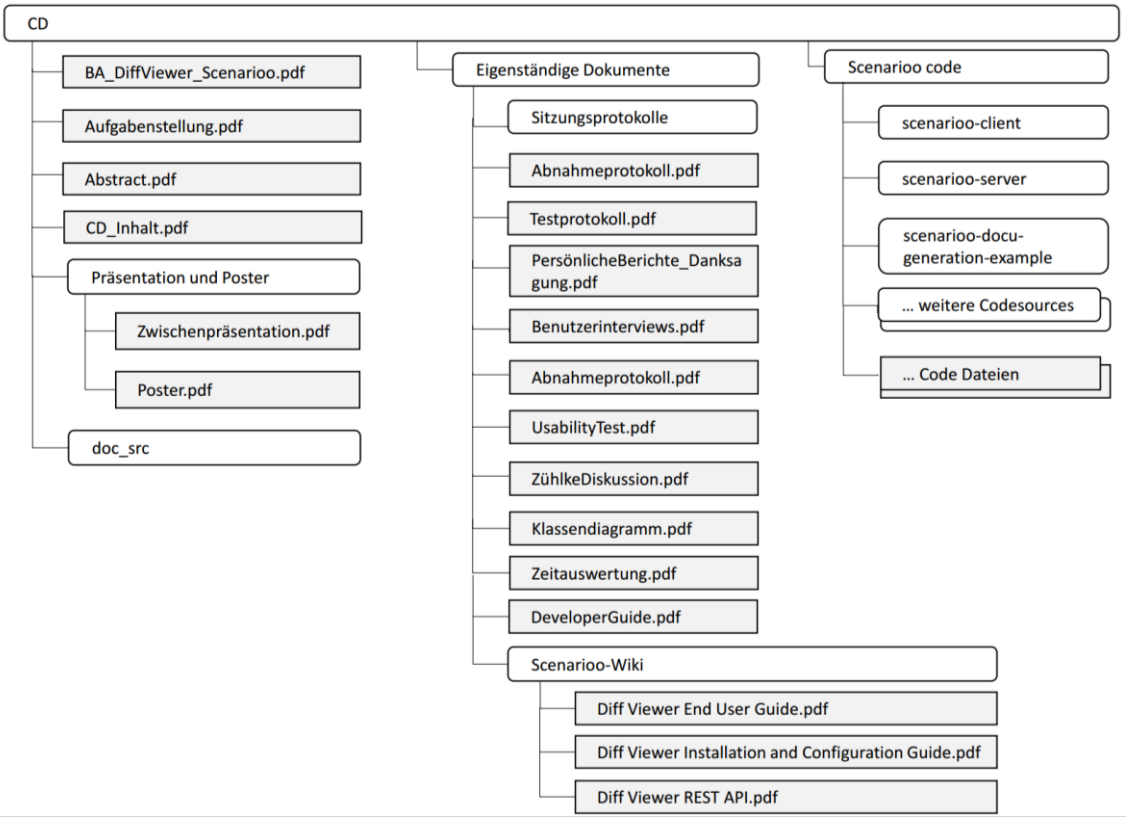


Abbildung 64: Inhalt der CD

C: Abkürzungsverzeichnis

Tabelle 27: Abkürzungsverzeichnis

Abkürzung	**Bedeutung**
ART	Adaptive Random Testing
CD	Compact Disk
CI	Continuos Integration
FHNW	Fachhochschule Nordwestschweiz
HTML	Hypertext Markup Language
IDE	Integrated Development Environment (dt. „Integrierte Entwicklungsumgebung“)
JAXB	Java Architecture for XML Binding
MAE	Mean Absolute Error
MSE	Mean Squared Error
MVC	Model View Controller
NFA	Nicht-funktionale Anforderungen

Abkürzung	Bedeutung
PAE	Peak Absolute Error
PSNR	Peak Signal to Noise Ratio
REST	Representational State Transfer
RMSE	Root Mean Squared Error
UI	User Interface
URL	Uniform Resource Locator
XML	Extensible Markup Language (dt. „erweiterbare Auszeichnungssprache“)

D: Glossar

Diff Viewer Begriffe

Das nachfolgende Glossar betrifft vor allem den Code und ist daher in Englisch verfasst.

Tabelle 28: Glossar Diff Viewer Begriffe

English	German	Description
base branch	Basis Branch	the currently selected branch
base build	Basis Build	the currently selected build
base use case	Basis Use Case	the currently selected use case
base scenario	Basis Szenario	the currently selected scenario
base step	Basis Schritt	the currently selected step
comparison branch	Vergleichsbranch	branch for comparison
comparison build	Vergleichsbuild	build for comparison
comparison use case	Vergleichsusecase	use case for comparison
comparison scenario	Vergleichsszenario	scenario for comparison
comparison step	Vergleichsschritt	step for comparison
comparison name	Vergleichsname	name of comparison
comparison configuration	Vergleichskonfiguration	configuration for a comparison build
diff information	Diff Information	information out of comparisons
diff screenshot	Diff Screenshot	screenshot that shows differences between two different screenshots

Technische Begriffe

Tabelle 29: Glossar technischer Begriffe

Begriff	Bedeutung
Agile Tester	aka „Embedded Tester“ = im Entwicklungs-Team / Scrum-Prozess eingebetteter Tester
Algorithmus	Eindeutige, ausführbare Folge von Anweisungen zur Lösung eines Problems
AngularJS	Ein JavaScript MVC Framework

Begriff	Bedeutung
Build	Builds unterscheidenen unterschiedliche Versionen einer Software
Burndown Chart	Vergleicht die Ist- und Sollstunden von einem Sprint
Continuous Integration	Prozess des fortlaufenden Zusammenfügens von Komponenten zu einer Anwendung
End-to-End Test	Diese Form der Test testen alle Komponenten eines Systems gemeinsam
Entity	Java Klasse, welche zur Datenhaltung dient
Framework	Ein Programmiergerüst das den Rahmen bereitstellt, mit dem ein Entwickler eine Anwendung erstellt
GitHub	Onlinedienst um Code von Softwareprojekten bereitzustellen
GraphicsMagick	GraphicsMagick ist ein Fork von ImageMagick, der eine bessere Performance Vorteile verspricht.
ImageMagick	ImageMagick ist eine Softwaresuite zur Erstellung und Bearbeitung von Rastergrafiken.
Junit	Junit ist ein beliebtes Framework um Unit-Tests zu schreiben
living documentation	Eine dynamische Methode einer Systemdokumentation, die aktuell, spezifisch und einfach zu verstehen ist
Markdown	Eine Vereinfachte Auszeichungssprache, in der wir die Wiki Inhalte geschrieben haben.
Minimum Viable Product	Ein Produkt, welches nur über die nötigen Basisfunktionen verfügt, damit es eingesetzt werden kann
Open Source Projekt	Projekt, welches frei zugänglich ist und von jedermann weiterentwickelt werden kann
PerceptualDiff	Ein Bildvergleichstool, das ein rechnerisches Modell des menschlichen Sehvermögens benutzt, um zwei Bilder zu vergleichen.
Product Backlog	Enthält alle noch nicht umgesetzten User Stories
Product Owner	Ist die Ansprechsperson auf Kundenseite im Prozessmodell Scrum
Progress-Bar	Fortschrittsanzeige in Form eines ICONS
Regressionstest	Wiederholte Testfälle die sicherstellen dass sich nichts verändert hat.
Ruby	Eine objektorientierte Programmiersprache welche zur Laufzeit interpetiert wird
Scrum	Dynamisches Prozessmodell mit mehreren Iterationen (Sprints)
Scrum Master	Diejenige Person, welche den Scrum Prozess überwacht
Scrum Team	Umfasst alle Personen welche an der Umsetzung der Funktionalität beteiligt sind
Selenium	Ein portables Softare Testing Framework für Webapplikationen
Selenium	Eine Testumgebung für Webseiten mit der sich Interaktionen ausführen lassen.

Begriff	Bedeutung
Sprint	Stellt eine Iteration im Prozessmodell Scrum dar
Sprint Backlog	Enthält alle in diesem Sprint umzusetzenden User Stories
Third Party Library	Eine wiederverwendbare Softwarekomponente von einem Drittanbieter
Tier	Einezlne Schicht einer Softwarearchitektur
UI testing	Der Prozess eine grafische Benutzeroberfläche zu testen
Use Cases	Ein Anwendungsfall, der die Funktionalität eines Systems auf Basis von einfachen Modellen beschreibt
User Story	Eine in Alltagssprache formulierte Software-Anforderung.

E: Quellenverzeichnis

10 Punkte für eine barrierefreie Website. 2016. http://www.access-for-all.ch/barrierefreiheit/barrierefreies-webdesign/10-punkte-fuer-eine-barrierefreie-website.html (Zugriff am 5. 06 2016).

Agilemanifesto. kein Datum. http://www.agilemanifesto.org/ (Zugriff am 01. März 2016).

AngularJS. 22. Mai 2016. https://de.wikipedia.org/w/index.php?title=AngularJS&oldid=154584763 (Zugriff am 7. Juni 2016).

Artefakte Reporting. kein Datum. http://scrum-master.de/Scrum-Artefakte_und_Reporting (Zugriff am 01. März 2016).

Boehm, Barry. «A Prioritized Top-Ten List of Software Risk Items.» In *Barry W. Boehm's Lifetime Contributions to Software Developement, Management, and Research* , von Barry Boehm, 362. New Jersey: John Wiley & Sons, 2007.

Cort J. Willmott*, Kenji Matsuura. *Advantages of the mean absolute error (MAE) over.* Delaware 19716, USA: CLIMATE RESEARCH, 2079.

«Demo Scenarioo.» *Scenarioo.* 2016. demo.scenarioo.org (Zugriff am 18. März 2016).

Farbtiefe (Computergrafik). 10. März 2016. https://de.wikipedia.org/wiki/Farbtiefe_(Computergrafik) (Zugriff am 03. Juni 2016).

Jasmine Introduction. kein Datum. http://jasmine.github.io/2.4/introduction.html (Zugriff am 06. Juni 2016).

JUnit - About. 18. April 2016. http://junit.org/junit4/ (Zugriff am 06. Juni 2016).

Less (Stylesheet-Sprache). 3. April 2016. https://de.wikipedia.org/w/index.php?title=Less_(Stylesheet-Sprache)&oldid=153133933 (Zugriff am 7. Juni 2016).

Mockito. 2015. http://mockito.org/ (Zugriff am 06. Juni 2016).

Nielsen, Jakob. *Why You Only Need to Test with 5 Users.* Nielsen Norman Group. 19. März 2000. https://www.nngroup.com/articles/why-you-only-need-to-test-with-5-users/ (Zugriff am 09. Mai 2016).

Peuker, Dr. Sibylle. «Evaluation.» *User Interfaces 2.* HSR, 2015.

Protractor Home. 29. April 2016. http://www.protractortest.org/ (Zugriff am 06. Juni 2016).

Regression testing definition. 12. Juli 2012. http://www.softwaretestingclass.com/regression-testing-definition/ (Zugriff am 01. Juni 2016).

Rolf Bruderer, Zühlke. «Living Documentation by Example with UI Tests in Practice.» 2014.

—. *Zühlke Blog.* 2015. http://blog.zuehlke.com/scenarioo-automatisierte-dokumentation-user-interface-tests/ (Zugriff am 18. März 2016).

—. «Zühlke Blog.» *Scenarioo Aufbau.* 2015. http://blog.zuehlke.com/wp-content/uploads/2015/06/scenarioo_architektur_und_schritte-755x496.png (Zugriff am 18. März 2016).

—. «Zühlke Blog.» *Dokumentationsmodell.* 2015. http://blog.zuehlke.com/wp-content/uploads/2015/06/scenarioo_logical_documentation_model1.png (Zugriff am 4. April 2016).

Rollen Team. 2015. http://scrum-master.de/Scrum-Rollen/Scrum-Rollen_Team (Zugriff am 01. März 2016).

Rouse, Margaret. *living documentation.* Dezember 2014. http://searchsoftwarequality.techtarget.com/definition/living-documentation (Zugriff am 01. Juni 2016).

Scenarioo Development Team. 2014. https://github.com/scenarioo (Zugriff am 18. März 2016).

«Scenarioo Struktur.» *Github.* 2016. https://cloud.githubusercontent.com/assets/5416988/3470838/f6fab378-02bd-11e4-8405-06d4d89c90a3.jpg (Zugriff am 4. April 2016).

«Scrum Übersicht.» *Mountaingoatsoftware.* 2016. https://www.mountaingoatsoftware.com/uploads/articles/ScrumLargeLabelled.png (Zugriff am 01. März 2016).

Selay, Elmin. *Adaptive Random Testing for Image Comparison.* Digital Image Computing: Techniques and Applications (DICTA): IEEE, 2014.

T. Chai, and R. R. Draxler1. *Root mean square error (RMSE) or mean absolute error (MAE)? – Arguments against avoiding RMSE in the literature.* University of Maryland: Geosci. Model Dev, 2014.

Visual Regression Testing in "Scenarioo". 2014.
http://web.fhnw.ch/technik/projekte/i/ip514/steiner-bernert/index.html (Zugriff am 2016. Juni 01).

F: Eigenständige Dokumente

Die eigenständigen Dokumente werden nicht veröffentlicht. Diese befinden sich nur auf der CD, da diese teilweise vertrauliche Informationen beinhalten oder den Rahmen dieser Arbeit sprengen würden.

Abnahmeprotokoll

Im Abnahmeprotokoll hat Zühlke die Anforderungen, den Code, die Dokumentation sowie das Projektmanagement abgenommen. Zudem enthält es ein Feedback zu unserer Arbeit von Rolf Bruderer und Daniel Suter.

Testprotokoll

Im Testprotokoll wird überprüft, ob die funktionalen sowie die nicht-funktionalen Anforderungen erreicht wurden. Auch protokolliert es die Arten, wie wir den Code getestet haben.

Persönliche Berichte und Danksagung

Enthält die persönlichen Berichte von Pascal Forster und Manuel Scheuber sowie eine Danksagung an alle Projektbeteiligten.

Benutzerinterviews

Zu Beginn dieser Arbeit wurden Interviews mit Scenarioo Benutzern und Entwicklern durchgeführt um die Anforderungen zu erarbeiten. Das Dokument Benutzerinterviews enthält den Fragebogen der dabei eingesetzt wurde sowie die anonymisierte Auswertung der Interviews.

Usability Test

Das Dokument Usability Test enthält die Aufgabenstellung, welche die Usability Testkandidaten erhalten haben. Ein weiterer Bestandteil sind die Fragestellungen im Vor- und Nachinterview.

Diskussion mit Zühlke

Enthält eine Zusammenfassung der Diskussionsbestandteile und Beschlüsse aus Meetings und dem Mailverkehr Zühlke.

Sitzungsprotokolle

Ein Ordner, in dem sämtliche Sitzungsprotokolle mit all ihren Traktanden und Beschlüssen abgelegt sind.

Klassendiagramm

Das Klassendiagramm zeigt eine Gesamtübersicht aller Komponenten im Diff Viewer.

Diff Viewer Installation and Configuration Guide

Der Diff Viewer Installation and Configuration Guide beschreibt die Schritte, die nötig sind, um das Diff Viewer Feature erstmals in Betrieb zu nehmen. Diese Anleitung ist ausserdem im Scenarioo Wiki verfügbar.

Diff Viewer End User Guide

Der Diff Viewer End User Guide beschreibt welche Möglichkeiten das Diff Viewer Feature bietet und wie das Feature bedient werden kann. Diese Anleitung ist ausserdem im Scenarioo Wiki verfügbar.

Diff Viewer REST API

Das Dokument Diff Viewer REST API beschreibt alle möglichen Aufrufe der Diff Viewer REST API und enthält jeweils einen Beispielsaufruf. Diese Anleitung ist ausserdem im Scenarioo Wiki verfügbar.

Developer Guide

Im Developer Guide wird beschrieben welche Schritte ein Entwickler vornehmen muss, um die Entwicklungsumgebung von Scenarioo inklusive dem Diff Viewer Feature vorzunehmen.

Zeitauswertung

In der Zeitauswertung sind die von uns gebuchten Zeiten nach Task angegeben.