ulm university universität ulm

Universität Ulm | 89069 Ulm | Germany

Fakultät für Ingenieurwissenschaften und Informatik
Institut für Datenbanken und Informationssysteme

Erweiterung eines Ansatzes für Bild-basierte GUI Testautomatisierung

Bachelorarbeit an der Universität Ulm

Vorgelegt von:
Stefan Kreidel
stefan.kreidel@uni-ulm.de

Gutachter:
Prof. Dr. Manfred Reichert
Dr. Dominic Müller (Daimler TSS GmbH)

Betreuer:
Rüdiger Pryss

2015

„Erweiterung eines Ansatzes für Bild-basierte GUI Testautomatisierung“
Fassung vom 20. April 2015

DANKSAGUNGEN:

An dieser Stelle möchte ich mich bei allen Personen, die direkt oder indirekt zur erfolgreichen Durchführung meiner Bachelorarbeit beigetragen haben, bedanken.

Dieser Dank gilt insbesondere Dr. Dominic Müller für die Übernahme der Zweitkorrektur und die hervorragende fachliche Betreuung meiner Arbeit. Insbesondere der „Brainstorming Modus“ war stets willkommen.

Besonders bedanken möchte ich mich auch bei Prof. Dr. Manfred Reichert und Rüdiger Pryss für die Ermöglichung der Durchführung dieser Arbeit, die konstruktive Unterstützung sowie die stete Hilfsbereitschaft.

Ein großer Dank gebührt weiterhin Björn Roy und Holger Seemüller für ihre Geduld und Hilfsbereitschaft bei all meinen Fragen.

Auch Barbara Bieselin möchte ich an dieser Stelle für ihre wertvolle Unterstützung danken.

Sperrvermerk

Die vorliegende Arbeit beinhaltet vertrauliche interne Informationen der Daimler TSS GmbH, die nicht für die Öffentlichkeit bestimmt sind und nur im Rahmen der Bachelorarbeit der Prüfungskommission zugänglich gemacht werden dürfen. Die Weitergabe oder Veröffentlichung des Inhalts der Arbeit im Gesamten oder in Teilen bedürfen einer vorherigen schriftlichen Zustimmung des Autors.

Satz: PDF-LaTeX 2ε

Inhaltsverzeichnis

1 Einleitung

Softwaresysteme sowie softwaregestützte und -gesteuerte Systeme haben in den letzten Jahren enorme Verbreitung gefunden. Nennenswerte Beispiele sind hier die stetig wachsende Anzahl an Smartphone und Tablet Besitzern [6] oder das enorme Wachstum des Internets in den letzten zwei Jahrzehnten [16].
Durch steigende Konkurrenz und immer höhere Anforderungen der Kunden ist die Qualität dieser Software ein entscheidender Faktor für den Erfolg eines Produkts [33].

Ein Mittel zur Sicherung eines bestimmten Qualitätsstandards ist das systematische Prüfen und Testen der Software. Ziel ist das Aufspüren von fehlerhaftem Verhalten, was die Grundlage zum Beseitigen dieser Fehler darstellt [22, 33]. Durch Automatisierung dieser Tests können der Testaufwand und die damit verbundenen Kosten meistens gesenkt werden [9, 32].
Ein Bestandteil, den es bei vielen Softwaresystemen zu testen gilt, ist die grafische Benutzeroberfläche (engl. *Graphical User Interface*, kurz *GUI*). Da sich die GUI im Laufe der Entwicklung in der Regel häufig ändert [21], muss der (automatisierte) GUI-Test ebenfalls häufig an diese Änderungen angepasst werden. Beispielsweise ist schon durch sich ändernden Layout oder Anzahl einzelner GUI-Elemente oft das Anpassen der Testfälle notwendig [20], wodurch der Testaufwand steigt.

Gerade im Bereich der GUI-Tests wäre daher ein Tool wünschenswert, welches einen effektiven und effizienten GUI Vergleich bietet. Insbesondere sollten auch einzelne fehlende Buchstaben oder minimale Falschpositionierung von Elementen gefunden werden. Weiterhin soll das Tool auch Automatisierung dahingehend unterstützen, geänderte GUIs ohne vorherige Anpassungen testen zu können.
Im Zuge seiner Masterarbeit wurde von Fabian Schwarz ein Tool, welches sich genau mit diesem Thema befasst, bereits prototypisch umgesetzt [29]. Die Evaluation des in Zusam-

menarbeit mit der *Daimler TSS GmbH*[1] entwickelten *image-based Comparison Automation Tool* (*iCAT*) hat gezeigt, dass der dahinterstehende Ansatz großes Potential hat. Daher befasst sich diese Arbeit erneut mit dem fortlaufend *iCAT v1* genannten Tool.
Wieder wurde die Arbeit bei der Daimler TSS GmbH in Ulm, Deutschland durchgeführt. Genauer gesagt entstand sie in Zusammenarbeit mit zwei Teams innerhalb der TSS: dem Team Test Automatisierung und dem Entwicklungsteam der Mercedes Benz Marketing Platform (MAP).
Der Fokus dieser Arbeit liegt auf der Neukonzeption des Tools mit dem Ziel, es in der Praxis tatsächlich operativ einsetzen zu können. Die bislang prototypisch umgesetzte Version iCAT v1 soll dabei deutlich generischer einsetzbar werden. Außerdem soll die neue Version die in den verschiedenen Teams entstandenen realen Anwendungsszenarien abdecken. Kern der fortlaufend *iCAT v2* genannten Version ist somit eine generische, zukunftsfähige und erweiterbare Architektur, welche vor allem flexible Vergleichsmöglichkeiten und eine einfache Verwendung von iCAT in agilen Projekten ermöglicht.

Der weitere Aufbau dieser Arbeit gliedert sich wie folgt:
In Kapitel 2 werden zunächst die relevanten Grundkenntnisse zum Thema Testen sowie die Besonderheiten von GUI-Tests vermittelt. Das Kapitel endet mit der Vorstellung des Bild-basierten Vergleichsansatzes sowie iCAT.
In Kapitel 3 wird der neu entwickelte Ansatz vorgestellt. Nach einer Aufzählung der in der Praxis relevanten Anwendungsszenarien und der damit verbundenen Einschränkungen von iCAT v1 werden davon die Anforderungen an den neuen iCAT Ansatz abgeleitet. Daraufhin wird anhand verwandter Arbeiten und Tools evaluiert, inwieweit die Anforderungen durch bereits vorhandene Ansätze abgedeckt werden können.
Im Anschluss daran wird in Kapitel 4 die Konzeption von iCAT v2 vorgestellt, worauf in Kapitel 5 die Implementierung dieser Konzeption beschrieben wird. Hierfür wird als erstes das Softwaredesign von iCAT v2 vorgestellt. Danach wird zunächst auf die iCAT Anwendung als Standalone Tool ohne GUI eingegangen und anschließend die entwickelte Webanwendung als mögliche Umsetzung einer GUI für iCAT beschrieben.
Kapitel 6 beinhaltet abschließend eine Zusammenfassung und Evaluation der Arbeit sowie einen Ausblick in die Zukunft.

[1] `http://www.daimler-tss.de`

2 Grundlagen GUI-Test

In diesem Kapitel werden die für diese Arbeit nötigen Grundlagen und Konzepte vorgestellt. Nachdem in Abschnitt 2.1 die Anforderungen an Softwaretests vorgestellt wurden gibt Abschnitt 2.2 einen Überblick über die verschiedenen Testarten in agilen Projekten. In Abschnitt 2.3 werden die Besonderheiten von GUI-Tests aufgezeigt. Nach der Vorstellung des Bild-basierten Vergleichsansatzes in Abschnitt 2.4 vermittelt der letzte Abschnitt 2.5 das grundlegende Verständnis zu iCAT, einem reinen GUI-Testtool.

2.1 Der Softwaretest

Qualität ist neben der reinen Funktionsvielfalt maßgeblich für den Erfolg eines Produkts verantwortlich. Von Kunden wird der Nutzen eines Produkts oft anhand von Qualitätsmerkmalen wie Leistungsfähigkeit, Benutzbarkeit und Komfort empfunden [24], wodurch die Sicherung eines definierten Qualitätsniveaus essentiell ist. Ein wichtiger Bestandteil der Qualitätssicherung ist der Test. In „The Art of Software Testing“ [22] wird dieser, bezogen auf Software, definiert als das „*Ausführen eines Programms, mit der Absicht, Fehler zu finden*“. Testen dient daher der Identifikation von Defekten und ist somit die Voraussetzung zur Beseitigung dieser.
Dabei muss jedoch berücksichtigt werden, dass der Softwaretest keine Fehlerfreiheit garantieren kann, da vollständiges Testen nicht möglich ist [22]. Eine sinnvolle Testfallauswahl ist somit wichtig, um mit einer Untermenge an Testfällen die größtmögliche Anzahl an Fehlern zu finden [22].

Obwohl das Erstellen von Testfällen im Wesentlichen für alle Softwareprojekte auf die gleiche Art abläuft, gibt es je nach Entwicklungsmodell erhebliche Unterschiede bei der Ausführung selbiger. Beim Wasserfallmodell von W. Royce [28] oder dem allgemeinen V-Modell nach

Boehm [4] wird beispielsweise ein sequentieller Ansatz verfolgt. Qualitätssicherung wird hier als eigene Phase am Ende der Entwicklung und somit nach der Implementierung angesehen. Im Gegensatz dazu wird bei agiler Projekten ein inkrementeller Entwicklungsansatz verfolgt [22]. Qualitätssicherung wird hier schon während der Implementierung betrieben. Da sich diese Arbeit unter anderem mit dem Testen in agilen Projekten befasst, wird dieser Prozess im Folgenden näher beschrieben.

2.2 Testen in agilen Projekten

Kern des agilen Entwicklungsansatzes ist, dass mit jeder Iteration einige der aktuell wichtigsten Funktionen umgesetzt werden, wodurch ein neues, lauffähiges Inkrement entsteht. Der Kunde entwickelt im Laufe der Entwicklung immer neue Vorstellungen vom finalen Produkt [22], woraus neue Anforderungen und Änderungswünsche entstehen. Um seine Zufriedenheit zu maximieren, wird jede neue lauffähige Version mit ihm besprochen. Danach werden die Anforderungen und als nächstes zu implementierende Funktionen entsprechend der aktuellen Vorstellungen vom finalen Produkt angepasst.

Da Testen in agilen Projekten während der Implementierung erfolgt [22], muss es den Entwicklungsiterationen folgen [8].
Grundlage zur Identifikation und Beseitigung ungewollter Seiteneffekte auf unveränderte Bereiche der Anwendung sind dabei Regressionstests. Unabhängig davon können Fehler in der Konfiguration einer Umgebung, welche unvorhersehbare Auswirkungen auf die Software haben können, mit Deployment-Tests gefunden werden. Um weiterhin alle nötigen Tests während jeder Iteration ausführen zu können, spielt Geschwindigkeit und damit Testautomatisierung eine wichtige Rolle [8, 9].

2.2.1 Regressionstests

Während jeder Iteration werden idealerweise parallel zur Implementierung neuer Funktionen Testfälle erstellt, mithilfe derer diese neuen Funktionen getestet werden können. Sobald diese Testfälle erfolgreich ausgeführt werden konnten, werden sie zu der Menge der Regressionstest-Testfälle hinzugefügt [9]. Ziel ist es, mithilfe dieser Regressionstests

zukünftig sicherstellen zu können, dass neue Funktionen keine ungewollten Seiteneffekte auf vorher funktionsfähige Teile der Anwendung haben.
Da das Beseitigen von Fehlern aufwendiger und somit teurer wird, je später sie gefunden werden [8], sollten Regressionstests immer nach funktionalen Erweiterungen, Verbesserungen oder Reparaturen ausgeführt werden [22], was idealerweise auf einer *Daily/Nightly-Build* Basis geschieht [9].

2.2.2 Deployment-Tests

Typischerweise wird Software und gerade Webanwendungen auf einer Entwicklerumgebung entwickelt. Um stets Zugriff auf eine aktuelle und lauffähige Version dieser Software zu haben, kann diese nach Erstellung auf einer andere Umgebung bereitgestellt werden (engl. *deployment*). Eine falsche Konfiguration dieser Umgebung kann jedoch fehlerhaftes Verhalten der bereitgestellten Software mit sich bringen [29].
Diese Konfigurationsfehler können mit Deployment-Tests gefunden werden. Dabei wird die Software beider Umgebungen gegeneinander verglichen.

2.2.3 Testautomatisierung

Testautomatisierung gilt als Schlüsselprozess für erfolgreiche agile Entwicklung [9]. Dabei darf Testautomatisierung allerdings nicht einzeln, sondern muss im Zusammenhang mit Regressionstests betrachtet werden. Automatisierung lohnt sich erst, wenn die mit jeder Testfallausführung gewonnene Zeit- und damit Kostenersparnis die Kosten der Erstellung einer automatisierbaren Regressionstestbasis unterbietet [32]. Da mit jeder Weiterentwicklung der Software deren Komplexität und damit auch die Anzahl der Testfälle für Regressionstests steigt, ist dies jedoch in der Regel der Fall [9].
Zusätzlich dazu ist manuelles Ausführen von Regressionstests fehleranfällig. Das Ausführen ähnlicher beziehungsweise immer gleicher Testfälle wird sehr schnell langweilig, wodurch die Wahrscheinlichkeit von Fehlern bei der Testfallausführung steigt [9].
Durch die mit Testautomatisierung gewonnene Zeit- und Kostenersparnis können Tests im Idealfall zuverlässig auf einer Daily/Nightly-Build Basis ausgeführt werden.

Es muss jedoch beachtet werden, dass manuelles Testen nicht komplett durch Testau-

tomatisierung ersetzt werden kann. Zum einen gibt es Tests, die günstiger manuelle als automatisiert ausgeführt werden können [10]. Zum anderen gibt es Fehler, die erst nach einer langen Kette an Bedienschritten auftreten und somit schwer zu reproduzieren sind. Eine Möglichkeit zum Aufspüren dieser Fehler sind explorative Tests, bei welchen der Tester die Anwendung nach dem Zufallsprinzip bedient. Aufgrund der schlechten Reproduzierbarkeit sollten diese immer manuell erfolgen [7].

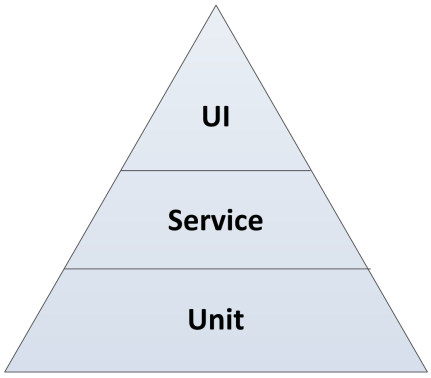

Abbildung 2.1: Die Test-Automatisierungs-Pyramide.

Obwohl Testautomatisierung entscheidende Vorteile hat, kann sie dennoch erhebliche Kosten mit sich bringen, wenn auf der falschen Ebene automatisiert wird. Mike Cohn schlägt daher vor, auf drei verschiedenen Ebenen zu automatisieren, was mithilfe der *Test-Automatisierungs-Pyramide* (vgl. Abbildung 2.1) dargestellt werden kann.
An der Spitze der Pyramide nehmen die Benutzerschnittstellen Tests (engl. *User Interface*, kurz *UI*) die kleinste Ebene ein. Dies ist darauf zurückzuführen, dass auf UI Ebene lediglich die Korrektheit der grafische Repräsentation einer Anwendung und nicht die darunter liegenden Funktionen getestet werden sollen [7].

2.3 GUI-Tests

Bei Software die über eine GUI bedient wird, macht diese oft 45 bis 60 Prozent des gesamten Programmcodes aus und ist somit eine der aus Sicht der Implementierung größten Softwarekomponenten [20]. Außerdem durchläuft die GUI während der Entwicklung vor allem in agilen Projekten mit am meisten Änderungen [21]. Gerade beim Ändern von vorhandenem Code ist jedoch die Wahrscheinlichkeit Fehler zu machen mit am größten

[22]. Häufig handelt es sich dabei um Fehler wie Layout Verschiebung um wenige Pixel oder einzelne falsche Buchstaben. Genau diese Fehler sind jedoch manuell meist schwer zu erkennen, weswegen es besonders wichtig ist, bei GUI-Tests effektiv und effizient zu sein.

Durch die häufigen Änderungen an der GUI sind wiederverwendbare Testfälle, die ohne Anpassungen auf der geänderten Software ausgeführt werden können, essentiell für erfolgreiches GUI-Testen [21]. Mit heutigen Methoden und Werkzeugen ist es jedoch schwer, robuste und wiederverwendbare Testfälle zu erstellen. Grund dafür ist, dass häufig verwendete Tools in der Regel keine reinen GUI-Testtools sind. Dadurch verwirklichen sie zumeist nicht Cohns Vorschlag, mit GUI-Tests nur die grafische Repräsentation einer Anwendung zu testen. Im Folgenden wird diese Problematik anhand des Browser Automatisierungstools *Selenium* demonstriert.

2.3.1 GUI-Tests mit Selenium

Kern von Selenium zum Automatisieren von Browsern ist die *Selenium WebDriver* Programmierschnittstelle (engl. *Application Programming Interface*, kurz *API*) [31]. Diese ist in der Lage den Browser wie ein menschlicher Nutzer zu bedienen. Dies ermöglicht im Wesentliches das Klicken auf HTML Elemente und Übermitteln von Text an diese. Zusätzlich dazu können Eigenschaften wie IDs, Sichtbarkeit, Position, Dimension und dargestellter Text von HTML Elementen ausgelesen, Screenshots der aktuell dargestellten Webseite aufgenommen sowie JavaScript Funktionen ausgeführt werden.

Selenium wird daher oft für Regressionstests von Web-Anwendungen genutzt. Dabei werden Funktionen der Anwendung geprüft, indem nach einer bestimmten Eingabe die daraus resultierende Ausgabe auf der GUI mit einem erwarteten Wert verglichen wird. Dies entspricht dem Prinzip, Anwendungsfunktionen über die UI zu testen.

Andersherum sind reine GUI-Tests mit diesem Ansatz mit einem hohen Aufwand verbunden: Alleine um das Layout einer Seite zu überprüfen, muss für jedes HTML Element dieser Seite die Position und Dimension mit einem Erwartungswert verglichen werden. Die dafür nötigen Testskripte müssen mit jedem neu hinzugekommenen oder entfernten Element überarbeitet werden oder robust genug sein, um mit fehlenden oder dynamisch erzeugten Elementen

umgehen zu können. Weiterhin müssen die Erwartungswerte zu Beginn der Entwicklung definiert und mit jeder Änderung an der GUI überarbeitet werden.
Besonders schwierig sind dabei Tests von GUIs, die mit Inhaltsverwaltungssystemen (engl. *Content Management System*, kurz *CMS*) erstellt wurden. CMS ermöglichen das Erstellen von GUIs nach dem Sandkastenprinzip. Die Entwickler können die GUIs mittels *Drag and Drop* aus einzelnen Elementen zusammenbauen, woraus das CMS den Quellcode der Anwendung erzeugt. Dies ermöglicht eine nahezu beliebige Anordnung und Darstellung von Inhalten [2]. Da die Entwickler nahezu keine Kontrolle und Kenntnis über den erstellten Quellcode haben, ist das Erstellen von Testskripten für solche GUIs sehr schwer bis nahezu unmöglich.

2.4 Bild-basierter Vergleichsansatz

Um tatsächlich lediglich die grafische Repräsentation einer GUI testen zu können, verfolgt der Bild-basierte Vergleichsansatz eine neuartige Idee. Anstatt aktiv nach vordefinierten Elementen zu suchen und deren Eigenschaften gegen Erwartungswerte zu vergleichen, wird hierbei eine Momentaufnahme der aktuellen GUI gegen eine schon vorhandene Aufnahme verglichen. Die dabei gefundenen Unterschiede sind entweder ungewollte Fehler oder neu implementierte Features.

Das Erstellen einer solchen Aufnahme muss dabei nicht neu erfunden werden, vielmehr kann hierfür auf Screenshots zurückgegriffen werden. Vorteilhaft an diesem Ansatz ist, dass das Erstellen von Screenshots immer möglich ist, solange die entsprechende GUI dargestellt werden kann. Insbesondere kommt es zu keinem Testabbruch, sollte die Menge der GUI-Elemente geändert worden sein.
Auch der Aufwand zum Testen einer Ansicht kann deutlich gesenkt werden. Während beim Selenium-Standardvorgehen der Testaufwand mit jedem weiteren GUI-Element mindestens linear wächst, bleibt er bei diesem Ansatz pro Seite konstant, da weiterhin genau ein Screenshot aufgenommen und verglichen werden muss. Dies ist insbesondere bei komplexen Dialogen vorteilhaft.
Weiterhin muss das Erstellen der Screenshots oftmals nicht manuell geschehen, da viele Umgebungen wie beispielsweise die meisten Webbrowser diese Funktion schon mitbringen. Greift man auf die Screenshot Funktion dieser Umgebungen zurück, erhält man außerdem

garantiert die korrekte grafische Darstellung der aktuellen Anwendung, da ein Browser beispielsweise genau das aufnimmt, was er selber darstellt.

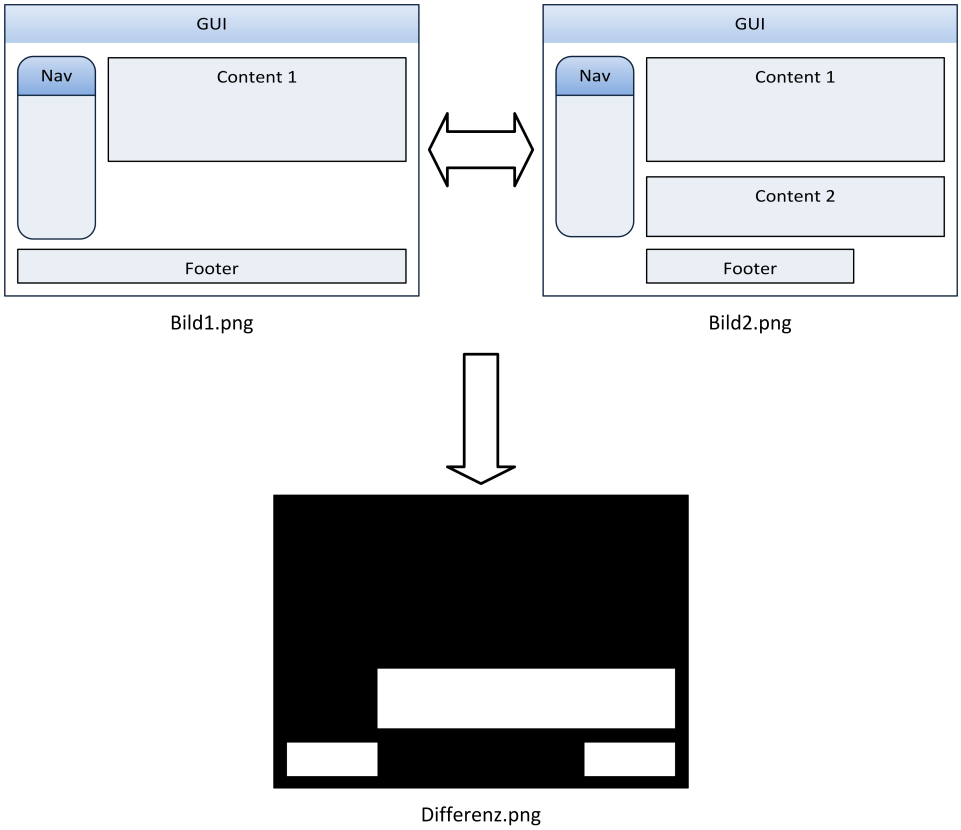

Abbildung 2.2: Der Bild-basierte Vergleich zweier GUIs und das daraus entstehende Differenzbild.

Wie in Abbildung 2.2 dargestellt, werden daraufhin zwei auf diese Art erstellte Screenshots Pixel für Pixel verglichen und daraus ein Differenzbild erstellt. Haben zwei verglichene Pixel unterschiedliche Farbwerte, so ist der entsprechende Pixel des Differenzbildes weiß, ansonsten schwarz. Somit stellen weiß Bereiche im Differenzbild gefundene Unterschiede dar.
Um eine genauere Eingrenzung der potentiellen Fehlerquelle zu ermöglichen, können die in Abbildung 2.2 dargestellten Regionen der einzelnen Screenshots (Navigation, Content 1, Content 2 und Footer) bei Vergleichen zusätzlich berücksichtigt werden. Dabei werden gefundene Unterschiede einer dieser Regionen zugeordnet.

2.5 iCAT – Das image-based Comparison Automation Tool

Das von Fabian Schwarz entwickelte iCAT stellt eine Implementierung des Bild-basierten Vergleichsansatzes für Webanwendungen dar [29]. Dadurch verwirklicht es Cohns Vorschlag, auf UI Ebene nur die grafische Darstellung zu testen.
Die Tests selber werden dabei nicht in Form von Testskripten erstellt. Stattdessen gibt der Tester, wie in Abbildung 2.3 dargestellt, eine Liste von URIs, eine Liste von Regionen und die Umgebung, auf der die Screenshots aufgenommen werden sollen, an. Die Umgebung ist der Server, auf dem die Anwendung bereitgestellt ist. Sie wird durch einen URI Präfix festgelegt. Die URI für jede einzelne Seite besteht somit immer aus `URI Präfix + Seitenpfad`. Um Screenshot-Sets auf unterschiedlichen Umgebungen aufzunehmen, muss nur dieser URI Präfix ausgetauscht werden.
iCAT nimmt daraufhin einen Screenshot für jede Webseite auf, speichert die Koordinaten und Dimensionen der Regionen für diese Seite und erstellt daraus ein sogenanntes Referenz-Set.

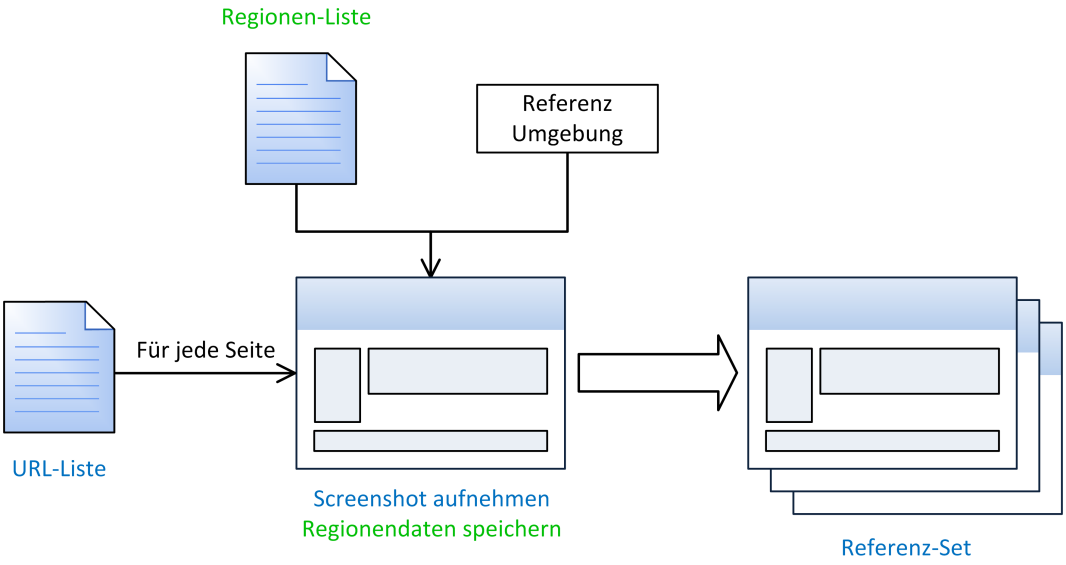

Abbildung 2.3: Erstellen eines Referenz-Sets.

Soll eine neue oder eine auf einer anderen Umgebung bereitgestellte Softwareversion gegen dieses Set verglichen werden, so muss nur die unter Umständen geänderte Umgebung angegeben werden. Wie in Abbildung 2.4 dargestellt, nimmt iCAT daraufhin unter Verwendung des Referenz-Sets, dessen Regionen-Liste und der neuen Umgebung Screenshots von

dieser Softwareversion auf, speichert die Regionendaten und erstellt daraus das sogenannte Test-Set. Die Screenshots beider Sets werden daraufhin paarweise unter Verwendung des Bild-basierten Vergleichsansatzes verglichen. Der eigentliche Bildvergleich wird dabei von ImageMagick, einem Kommandozeilentool zum Vergleichen und Bearbeiten von Bildern [15], übernommen. Mit Hilfe der dadurch entstandenen Differenzbilder wird anschließend ein Report erstellt [29].

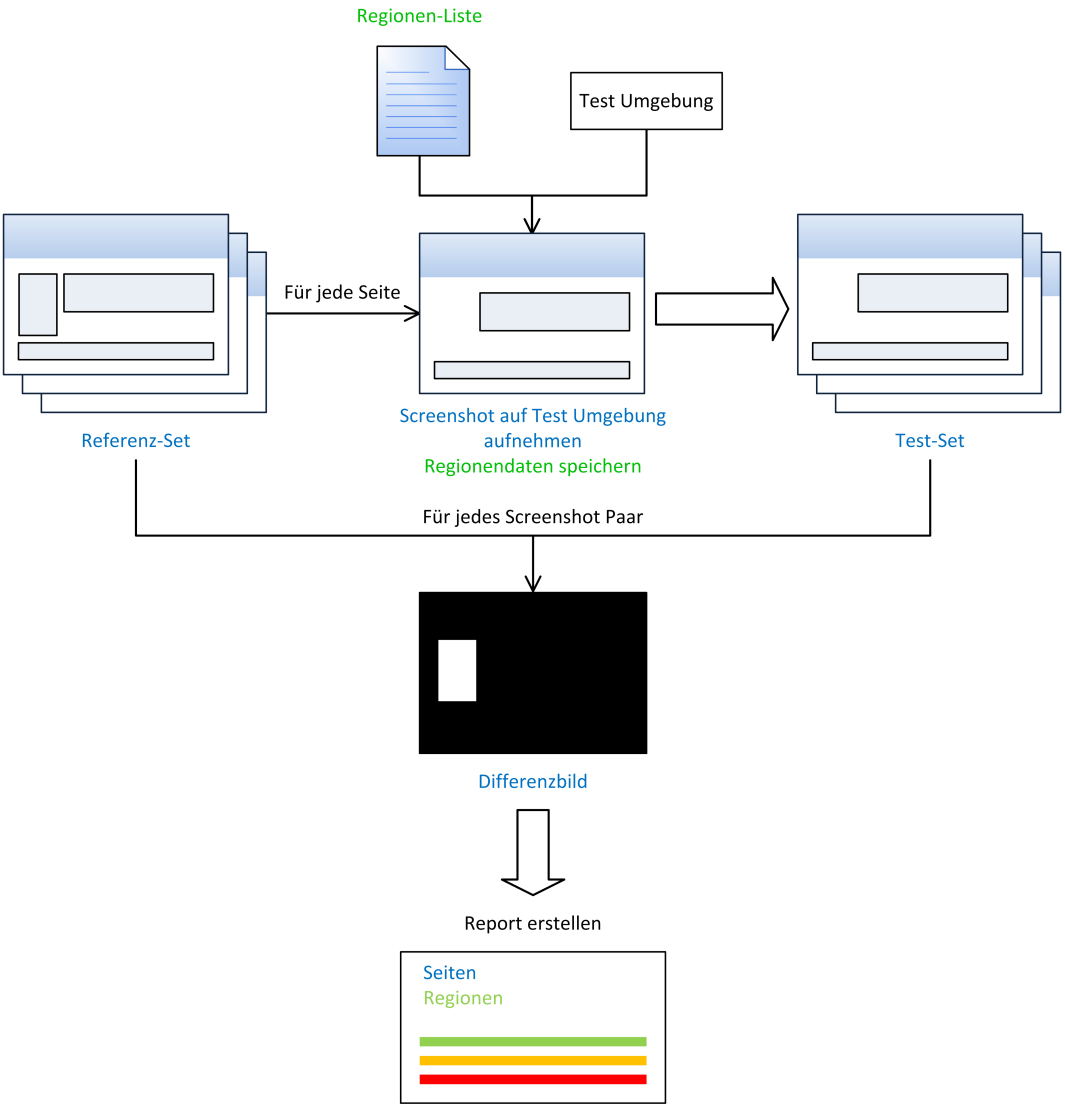

Abbildung 2.4: Erstellen eines Test-Sets und anschließendes Vergleichen mit einem Referenz-Set.

Vorteilhaft an diesem Ansatz ist, dass diese Tests durch Änderungen an der GUI meist nicht unbrauchbar werden. Zum einen sollen gerade fehlende oder neue GUI-Elemente gefunden werden, was dem Grundgedanken von Regressions- und Deployment-Tests entspricht. Der Test muss dafür nicht angepasst werden, da von der geänderten GUI nur ein neuer Screenshot aufgenommen werden muss. Zum anderen sind URI-Liste und Regionen-Liste schneller und einfacher anpassbar als Testskripte. Letztendlich sind durch den Verzicht auf Testskripte Tests auch für Nicht-Entwickler pflegbar.

3 Neu entwickelter Ansatz

Das ursprünglich fokussierte Anwendungsszenario war, visuelle Unterschiede, die im MAP Team häufig nach dem Deployment einer Anwendung auf eine andere Umgebung auftraten, pixelgenau und automatisiert zu identifizieren.
Nach einem knappen Jahr im Einsatz konnte das MAP Team erste Erfahrungen mit dem entwickelten Tool sammeln. Auch in anderen Teams der Daimler TSS entstand Interesse an iCAT, was insgesamt zu neue Ideen und Wünschen für den Einsatz von selbigem führte. Um diesen Ideen und Wünschen gerecht zu werden, liegt der Fokus dieser Arbeit darauf, iCAT von Grund auf neu zu konzeptionieren. Das Ziel dabei ist, iCAT in der Praxis auch tatsächlich operativ und für verschiedene Anwendungszwecke einsetzen zu können.
In Abschnitt 3.1 werden zunächst alle in der Praxis relevanten Szenarien für den Einsatz von iCAT aufgeführt. Diese bilden die Grundlage für die in Abschnitt 3.2 beschriebenen Einschränkungen des bisherigen iCAT Ansatzes, aus denen in Abschnitt 3.3 die Anforderungen an den neuen Ansatz abgeleitet werden. Abschließend wird in Abschnitt 3.4 anhand verwandter Arbeiten und ähnlicher Tools evaluiert, inwieweit die Anforderungen durch bereits vorhandene Ansätze abgedeckt werden können.

3.1 Anwendungsszenarien von iCAT

Um eine genaue Vorstellung von den relevanten Anwendungsszenarien für den Einsatz von iCAT zu erhalten, wurden mehrere Mitglieder des MAP Teams sowie einige Mitglieder anderer Teams in mehreren Interviews befragt.

Aus der Anforderungsanalyse ergaben sich folgende relevante und zum Teil neue Anwendungsszenarien für iCAT:

S.1 Smoke-Tests Nach der Implementierung neuer Features soll geprüft werden, ob unvorhergesehen Änderungen am Systemverhalten auftreten. Insbesondere sollen ungewollte Seiteneffekte auf unveränderte Teile des Codes gefunden werden.

S.2 Deployment-Tests Nach der Entwicklung und Überprüfung einer neuen Softwareversion auf einer dafür vorgesehenen Entwickler-Umgebung soll diese Version auf einer Produktiv-Umgebung bereitgestellt werden. Mithilfe von Deployment-Tests soll dabei sichergestellt werden, dass die Konfiguration der Produktiv-Umgebung keine Seiteneffekte auf die GUI der Software hat.

S.3 Neue Features überprüfen Da iCAT darauf ausgelegt ist, Unterschiede zwischen zwei Screenshots zu finden, wird der Vergleich bei korrekter Implementierung eines neuen Features auf jeder betroffenen Seite fehlschlagen. Ob die von iCAT gefunden Unterschiede tatsächlich den neuen Features entsprechen und ob diese alle korrekt dargestellt werden, muss daher zunächst manuell überprüft werden.
Wurden alle Features korrekt implementiert, soll dieses aktuelle Test-Set zukünftig als Referenz-Set für Smoke Tests einsetzbar sein. Traten bei der Implementierung Fehler auf, wäre es denkbar, das aktuelle Test-Set als Ausgangspunkt für *Test Driven Bug-Fixing* zu verwenden. Bei zuvor korrekt implementierten Features findet iCAT nach dem Bug-Fixing idealerweise keine visuellen Unterschiede. Bei zuvor nicht korrekt implementierten Features sollte iCAT hingegen Unterschiede finden. Manuell muss dabei nur geprüft werden, ob neu gefundene Unterschiede den vorher fehlerhaften Features entsprechen und ob diese nun korrekt dargestellt werden.

S.4 Zusammengehörigkeit von Tests Wünschenswert ist eine Zusammengehörigkeit zwischen den aktuellen und vorherigen, beziehungsweise nachfolgenden Tests der gleichen Software sowie eine Verknüpfung der Tests zur zugrundeliegenden Softwareversion. Dadurch kann nachvollzogen werden, wann neue Features implementiert und in welchen Schritten gegebenenfalls aufgetretene Fehler behoben wurden.

S.5 iCAT Schnittstelle für Selenium Momentan kann iCAT nur Screenshots von Webseiten und deren einzelner Zustände aufnehmen, die durch eine URI abbildbar sind. Eine logische Erweiterung stellt dabei die Kombination aus iCAT und einem Selenium WebDriver Testskript dar. Mit Hilfe des WebDrivers kann dabei eine Webanwendung bedient und über die Schnittstelle zu iCAT eine Screenshot Aufnahme ausgelöst wer-

den. Somit können beispielsweise Assistentsoberflächen (engl. *Wizards*) bedient und visuell verglichen werden.

S.6 Mobile Web Applikationen Die bisherigen Konzepte und Szenarien sollen auf mobile Web Applikationen übertragen werden. Statt eines Desktop Browsers soll iCAT (bzw. das entsprechende Selenium Testskript) einen mobilen Browser bedienen. In wieweit visuelle Vergleiche aufgrund von Paradigmen wie *Responsive Layout* möglich und sinnvoll sind, muss dabei zuvor evaluiert werden.

3.2 Einschränkungen von iCAT v1 in der Praxis

Obwohl iCAT bereits viele Probleme des Selenium-Standardverfahrens zum Testen von GUIs löst, gibt es dennoch einige Ansatzpunkte für Verbesserungen und Erweiterungen. Gerade im Hinblick auf die relevanten Anwendungsszenarien sowie Anwendungsfälle lassen sich folgende, weiterhin bestehende Probleme von iCAT v1 formulieren:

P.1 Sehr spezifisch für ein Projekt Der ursprüngliche Fokus von iCAT war, GUI Fehler nach dem Deployment der Marketing Platform auf eine andere Umgebung zu finden. Da iCAT spezifisch für die MAP entwickelt wurde, sind beispielsweise die für einen Vergleich verwendeten Regionen im Quellcode hart codiert. Einerseits müsste dadurch der Quellcode für jedes andere Projekt geändert werden und andererseits kann iCAT so nicht gleichzeitig für mehrere Projekte verwendet werden.
Weiterhin wird die Verzeichnisstruktur für abgespeicherte Screenshots auf dem Dateisystem anhand eines bestimmten, projektspezifischen URI Musters aufgebaut. Dadurch ist iCAT nicht für Projekte einsetzbar, deren URIs nicht diesem Muster entsprechen.

P.2 Eingeschränkt in der Benutzung iCAT beinhaltet momentan keine Unterstützung für den Vergleich von sich stark unterscheidenden Softwareversionen. Führt beispielsweise eine Änderung an einer Webseite zu einer Änderung ihrer Größe (und damit zu einer Änderung der Auflösung des Screenshots), so kann iCAT diesen Screenshot nicht mit einem Screenshot einer früheren Softwareversion (und damit anderer Auflösung) vergleichen.
Weiterhin kann iCAT keine Screenshot-Sets mit unterschiedlicher Seiten- oder Re-

gionenanzahl vergleichen. Dies ist insbesondere von Nachteil, wenn mit jeder neuen Softwareversion nach und nach einzelne Webseiten hinzukommen.

P.3 Eingeschränkt in agiler Entwicklung nutzbar Da der ursprüngliche Fokus von iCAT auf Deployment-Tests lag, ist es nicht möglich, ein für einen Vergleich erstelltes Test-Set zukünftig als Referenz-Set wiederzuverwenden. Gerade in agilen Projekten ist dies von Nachteil. Steht nach einem Vergleich fest, dass keinerlei ungewollte Regression gefunden wurde und alle neuen Features korrekt implementiert sind, so muss für diese Softwareversion manuell ein neues Referenz-Set aufgenommen werden, obwohl alle Screenshots schon in Form des aktuellen Test-Sets vorliegen.
Weiterhin bietet die GUI des alten iCAT keine Möglichkeit, die Zusammengehörigkeit von Screenshot-Sets, Software und Softwareversion strukturiert darzustellen, wodurch der Überblick schnell verloren geht.
Auch bei der Verwendung in agilen Projekten muss iCAT stets manuell bedient werden. Besonders bei der Verwendung von Systemen zur kontinuierlichen Integration (engl. *Continuous Integration*, kurz *CI*) von Software ist es umständlich, Regressionstests mit iCAT manuell starten zu müssen.

3.3 Anforderungen

Die in 3.1 beschriebenen Szenarien und Anwendungsfälle sowie die in 3.2 aufgezählten Einschränkungen und Ansatzpunkte für Verbesserungen bilden die Grundlage zur Motivation dieser Arbeit.
Da im Rahmen dieser Bachelorarbeit nicht alle Anwendungsszenarien und vorhandenen Einschränkungen berücksichtigt werden konnten, wurde eine Priorisierung vorgenommen. Die geplante Neukonzeption beruht auf den folgenden Anforderungen:

A.1 GUIs sollen Bild-basiert anhand von Screenshots verglichen werden.

Durch den pixelgenauen Vergleich und das Highlighting der Unterschiede können diese sehr effizient und zuverlässig gefunden werden.

A.2 iCAT soll generisch und für verschiedene Projekte einsetzbar sein.

iCAT soll, ohne spätere Änderungen am Quellcode vornehmen zu müssen, Screenshot-Sets

von beliebigen Webanwendungen aufnehmen können.

A.3 iCAT soll flexible Vergleichsmöglichkeiten von Screenshot-Sets bieten.

Jedes aufgenommene Screenshot-Set soll für Vergleiche verwendet werden können, auch wenn sich die Zusammensetzung der Sets oder einzelne Screenshots stark unterscheiden.

A.4 Es soll möglich sein, ganze Screenshot-Sets vollständig automatisiert zu vergleichen.

Nicht nur Screenshots, sondern ganze Screenshot-Sets sollen ohne weiteren Konfigurationsaufwand vollständig automatisiert verglichen werden.

A.5 Die Ergebnisse eines Vergleichs sollen in einem Report detailliert dargestellt werden.

Die bei einem Vergleich erstellen Differenz-Screenshots sollen analysiert und das Ergebnis aller Analysen in einem Report grafisch aufbereitet dargestellt werden. Dadurch kann der Tester leicht die für ihn relevanten Differenz-Screenshots identifizieren.

A.6 Der Einsatz von iCAT in agilen Projekten soll unterstützt werden.

Die inkrementelle Entwicklung in agilen Projekten hat in der Regel viele Softwareversionen samt Regressionstests zur Folge. iCAT soll es ermöglichen, den Überblick über viele Screenshot-Sets samt Verknüpfung zu einer entsprechenden Softwareversion zu behalten.

A.7 iCAT soll eine GUI für die manuelle Bedienung besitzen.

Durch eine GUI soll iCAT einerseits auch für Nicht-Entwickler benutzbar sein und andererseits die Zusammenhänge zwischen Software, Screenshot-Sets und Reports visuell erkenntlich gemacht werden.

A.8 iCAT soll eine CLI für die Integration in eine CI-Umgebung besitzen.

Durch eine Kommandozeilen-Schnittstelle (engl. *Command-Line Interface*, kurz *CLI*) soll es möglich sein, iCAT über Konfigurationsparameter auch aus einer CI-Umgebung heraus zu verwenden.

3.4 Verwandte Arbeiten

Um zu überprüfen, inwieweit die Anforderungen durch bereits vorhandene Ansätze abgedeckt werden können, wurden die von Schwarz aufgeführten Tools abermals überprüft und zusätzlich nach weiteren ähnlichen oder gänzlich neuartigen Tools im Bereich der GUI-Tests gesucht. Die einzelnen Tools und das Resultat dieser Evaluation wird im Folgenden aufgeführt:

Wraith
Wraith ist ein von BBC News entwickeltes und in Ruby implementiertes Open Source Front-End und Regressionstest Tool [3] zum Vergleichen von Webanwendungen auf verschiedenen Umgebungen. Ähnlich wie beim momentanen iCAT Ansatz wird ein Browser zum Aufnehmen der Screenshots und ImageMagick zum Vergleichen dieser benutzt. Da die zu vergleichenden Webseiten dabei in Form zweier Listen aus URIs definiert sind und als Resultat des Tests lediglich die entstandenen Differenzbilder gespeichert werden, können viele der an den neuen iCAT Ansatz gestellten Anforderungen nicht erfüllt werden.

QualityBots
QualityBots ist ein von Google entwickeltes Tool zum Vergleichen von Webseiten auf verschiedenen Chrome Channels [14]. Es soll Entwickler dabei unterstützen Layout Änderungen mit der Veröffentlichung neuer Chrome Versionen frühzeitig zu erkennen. Dafür werden die Document Object Models (DOMs) der verschiedenen Channels Pixel für Pixel verglichen und eine Abweichung in Prozent berechnet [14]. Dieses Vorgehen ermöglicht jedoch keinerlei Wiederverwendung. Weiterhin ist es auch nicht zum Vergleich verschiedener Softwareversionen oder Umgebungen gedacht.

Piwik - Screenshot Testing
Piwik ist eine Open Source Web Analyseplattform [25]. Teil dieser Plattform sind die sogenannten Screenshot Tests, bei denen Screenshots zunächst mithilfe von PhantomJS aufgenommen und danach anhand ihrer Bytewerte verglichen werden. Nur falls dieser Vergleich fehlschlägt, wird mithilfe von ImageMagick ein Differenzbild erstellt [25]. Die von Piwik ausgeführten Tests müssen jedoch geskriptet werden, wobei dem Tester im Wesentlichen zwei Funktionen zur Auswahl stehen: Einen Screenshot aufnehmen und zwei bereits aufgenommene Screenshots vergleichen [26].

Zwei der Vorteile von Piwik sind, dass aufgenommene Screenshots für zukünftige Tests wiederverwendet werden können und für jeden Test ein grafischer Report erstellt wird. Andererseits ist das Erstellen und Überarbeiten bei Skripten aufwendiger als bei URI- und Regionen-Listen. Weiterhin müssen bei Piwik nicht nur für das Aufnehmen, sondern auch für das Vergleichen Skripte erstellt werden. iCAT wiederum benötigt nur für die Aufnahme jeweils eine URI-Liste und Regionen-Liste.

Screenster
Screenster ist ein von Agile Engine entwickeltes Tool zur Automatisierung visueller Tests von Webanwendungen [1]. Ein Testfall wird erstellt, indem ein Tester die benötigten Seiten nacheinander manuell aufruft. Screenster speichert für jede Seite die URI sowie einen Screenshot, der für diesen Testfall die Referenz darstellt. Um eine weiterentwickelte Softwareversion gegen die aufgenommene zu vergleichen, wird der Testfall automatisiert erneut ausgeführt. Dadurch entstandene Screenshots werden gegen die Referenz-Screenshots verglichen und daraus ein Report erstellt.
Da die gespeicherten URIs vor dem erneuten Ausführen nicht geändert werden können, ist kein Vergleich verschiedener Umgebungen möglich. Ebenso wenig möglich ist eine Wiederverwendung der dadurch entstandenen Test-Screenshots. Somit werden einige der an den neuen iCAT Ansatz gestellten Anforderungen nicht erfüllt.

3.4.1 Fazit

Die folgende Tabelle 3.1 fasst zusammen, welche der an das neue iCAT gestellten Anforderungen von den Tools erfüllt werden.
Einige der genannten Tools realisieren tatsächlich einen iCAT ähnlichen Ansatz, allen voran Piwik mit seinem grafischem Reporting und der Möglichkeit zur Wiederverwendung von Screenshots. Jedoch scheint keines der Tools alle in Abschnitt 3.3 aufgelisteten Anforderungen zu erfüllen. Gerade die flexiblen Vergleichsmöglichkeiten oder der automatisierte Vergleich ganzer Screenshot-Sets werden von keinem Tool vollständig unterstützt (wraith und Screenster bieten jeweils die Möglichkeit Softwareversionen oder Umgebungen zu vergleichen, nicht jedoch beides). Weiterhin wird auch die schon an iCAT v1 gestellte Anforderung, Vergleiche im Report auf Regionenebene zu analysieren [29], nach wie vor von keinem Tool unterstützt.

Daher wurde entschlossen, dass sich die Konzeption und Umsetzung des neuen iCAT lohnt. Die Konzeption wird in Kapitel 4, die eigentliche Implementierung in Kapitel 5 beschrieben.

	wraith	**QualityBots**	**Piwik**	**Screenster**
A.1 Bild-basierter Vergleich	✓	X	✓	✓
A.2 Generisch einsetzbar	X / ✓	X	✓	✓
A.3 Flexible Vergleichsmöglichkeiten	X	X	X / ✓	X
A.4 Automatisierter Screenshot-Set Vergleich	X	X / ✓	X	X
A.5 Detailliertes Reporting	X	X	X / ✓	X / ✓
A.6 Einsatz in agilen Projekten unterstützt	X	X	X / ✓	X
A.7 GUI für manuelle Bedienung	✓	X	X	✓
A.8 CLI für Integration in CI-Umgebung	X	X	X	X

Legende:
- ✓ Wird vollständig erfüllt
- X / ✓ Wird nur teilweise erfüllt
- X Wird nicht erfüllt

Tabelle 3.1: Evaluation der Tools hinsichtlich der zu erfüllenden Anforderungen.

4 Konzeption

In diesem Kapitel wird die Konzeption konkret ausgearbeitet. Dabei werden in Abschnitt 4.1 zunächst die Lösungskonzepte für die in Kapitel 3 definierten Anforderungen vorgestellt, welche die Grundlage für die in Abschnitt 4.2 beschriebene Architektur von iCAT v2 bilden.

4.1 Lösungskonzepte

Die in Abschnitt 3.3 definierten Anforderungen fassen die wesentlichen Eigenschaften und Funktionsweisen des neuen iCAT Ansatzes bereits zusammen. Als Basis für die Umsetzung werden im Folgenden die Lösungskonzepte für jede einzelne Anforderung erläutert.

Anforderung A.1: *Bild-basierter Vergleich*

- Nach wie vor kommt der Bild-basierte Vergleichsansatz zum Einsatz, um einen pixelgenauen Screenshot Vergleich samt Highlighting der Unterschiede auf einem Differenz-Screenshot zu ermöglichen.

- Die Screenshots können in Regionen unterteilt werden. Bei der Analyse eines Screenshot Vergleichs werden gefundene Unterschiede entsprechend ihrer Position den Regionen zugewiesen.

Anforderung A.2: *Generisch einsetzbar*

- Die für eine Screenshot-Set Aufnahme verwendeten Regionen werden nach dem Vorbild der URI-Liste in einer Regionen-Liste angegeben.

- Der Verzeichnispfad, unter dem Screenshots auf dem Dateisystem gespeichert sind, wird nach einem allgemeinen Schema aufgebaut. Screenshots des selben Screenshot-

Sets werden im selben Ordner gespeichert und nach dem Namen der zugrundeliegenden Webseite sowie einem eindeutigen Suffix benannt.

Anforderung A.3: *Flexible Vergleichsmöglichkeiten*

- iCAT v2 kennt zwei Basisprozesse:
  - Das Aufnehmen von Screenshot-Sets
  - Das Vergleichen von Screenshot-Sets
- Es wird nicht zwischen Referenz-Set und Test-Sets unterschieden. Welches Screenshot-Set die Funktion des Referenz- und Test-Sets übernimmt, wird erst beim Vergleichen zweier Sets festgelegt.
- Für jedes Screenshot-Set werden für alle Screenshots die Koordinaten und Dimensionen aller gefundenen Regionen gespeichert.
- Umgebungen können angelegt und beim Aufnehmen eines Screenshot-Sets ausgewählt werden.
- Screenshots werden auf der angegebenen Umgebung aufgenommen, wodurch die in der URI-Liste angegebene Umgebung überschrieben wird.
- Der Browser, in dem die Screenshots aufgenommen werden, kann vor jeder Aufnahme ausgewählt werden.
- Screenshots unterschiedlicher Auflösung werden verglichen, indem für jedes Screenshot-Paar die einzelnen Regionen paarweise aus den Screenshots herausgeschnitten und Bild-basiert verglichen werden. Haben dabei zwei Regionen unterschiedliche Dimensionen, ist ein Vergleich auch mit diesem Verfahren nicht möglich.

Anforderung A.4: *Automatisierter Screenshot-Set Vergleich*

- Vor dem eigentlichen Vergleich wird zuerst eine vergleichbare Schnittmenge der beiden Screenshot-Sets erstellt. Dadurch ist eine Auf- und Abwärtskompatibilität von Softwareversionen mit unterschiedlicher Seitenanzahl gewährleistet.

- Bei einem Vergleich wird auf dem Test-Screenshot nur nach Regionen gesucht, die auch auf dem Referenz-Screenshot gefunden wurden. Dadurch ist eine Auf- und Abwärtskompatibilität von Softwareversionen mit unterschiedlicher Regionenanzahl gewährleistet.

Anforderung A.5: *Detailliertes Reporting*

- Screenshot Vergleiche werden einer der drei Kategorien *erfolgreich, teilweise erfolgreich* und *fehlgeschlagen* zugeordnet.
- Die Ergebnisse der Regionen Vergleiche fließen mit in den Report ein.
- Die Screenshot Vergleiche lassen sich nach obigen drei Kategorien oder nach fehlgeschlagenen Regionen filtern. Dadurch können schnell die für einen Vergleich relevanten Unterschiede gefunden werden.

Anforderung A.6: *Einsatz in agilen Projekten*

- Für jedes Projekt wird ein neuer Test angelegt.
- Ein Test kann in Versionen unterteilt werden. Dadurch kann eine Verknüpfung zur zugrundeliegenden Softwareversion hergestellt werden.
- Pro Test muss eine initiale URI-Liste angeben werden. Diese kann mit jeder neuen Testversion gegebenenfalls ersetzen werden.
- Pro Test muss eine initiale Regionen-Liste angegeben werden. Diese kann mit jeder neuen Testversion gegebenenfalls ersetzen werden.
- Für jede Testversion können beliebig viele Screenshot-Sets aufgenommen werden. Diese verwenden alle die URI- und Regionen-Liste dieser Testversion.

Anforderung A.7: *GUI für manuelle Bedienung*

- Die Bedienung von iCAT v1 hat sich durch die Verwendung einer GUI als sehr einfach erwiesen. Auch Nicht-Entwickler können das Tool somit verwenden. Außerdem kann die Strukturierung von Screenshot-Sets in Tests und Testversionen nur mit einer

GUI sinnvoll genutzt werden. Daher soll auch für iCAT v2 eine GUI in Form einer Webanwendung entwickelt werden.

Anforderung A.8: *CLI für Integration in CI-Umgebung*

- iCAT besitzt eine CLI wodurch eine spätere Integration in eine CI-Umgebung (z.B. Jenkins [18]) ermöglicht wird. Durch Aufrufparameter kann iCAT von dieser CI-Umgebung dabei entsprechend konfiguriert werden.

4.2 Architektur

Nachfolgend wird für die definierten Anforderungen und entsprechenden Lösungskonzepte eine Grobarchitektur ausgearbeitet. Zunächst wird auf die im vorhergehenden Abschnitt aufgeführten Basisprozesse näher eingegangen. Danach werden die von iCAT verwendeten Entitäten sowie deren Zusammenhänge beschrieben.

4.2.1 Basisprozesse

Der Kern von iCAT v2 besteht darin, Screenshot-Sets aufzunehmen und diese danach beliebig untereinander zu vergleichen. Ein solches Screenshot-Set stellt dabei die von einem bestimmten Browser dargestellte visuelle Repräsentation (eines Teils) einer Softwareversion auf einer bestimmten Umgebung dar. Das Vergleichen dieser Sets ermöglicht somit den visuellen Vergleich verschiedener Softwareversionen, Umgebungen und Browser.
Die beiden Basisprozesse des Aufnehmens und Vergleichens können dabei wie folgt in einzelne Zwischenschritte unterteilt werden:

Screenshot-Set aufnehmen Wie in Abbildung 4.1 dargestellt, muss für die Aufnahme eines neuen Screenshot-Sets eine URI-Liste, eine Regionen Liste, eine Umgebung und ein Browser angegeben werden.
iCAT startet nun den angegebenen Browser und öffnet jede in der URI-Liste aufgeführte Webseite. Dabei wird der Host jeder URI jeweils mit dem in der Umgebung angegebenen

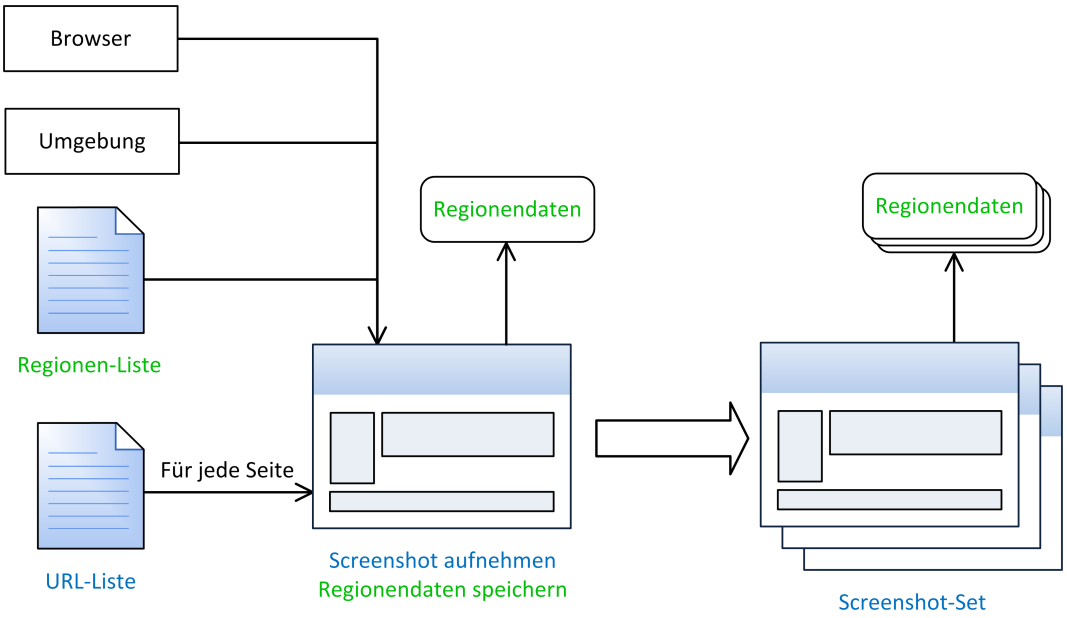

Abbildung 4.1: Aufnahme eines Screenshot-Sets.

Host überschrieben. Ist die Webseite vollständig geladen, wird ein Screenshot aufgenommen. Anschließend wird auf der Seite nach allen in der Regionen-Liste angegebenen Regionen gesucht und die Dimensionen und Positionen aller gefundenen gespeichert.
Durch die Angabe dieser Listen und Parameter können Screenshot-Sets von beliebigen Webanwendungen aufgenommen werden, solange jede einzelne Ansicht der Webanwendung über eine URI abbildbar ist. Somit ist iCAT nun generisch und für nahezu beliebige Projekte einsetzbar.

Screenshot-Sets vergleichen Nach obigem Verfahren erstellte Screenshot-Sets können nun beliebig gegeneinander verglichen werden.
Dafür wird zunächst eine vergleichbare Schnittmenge beider Sets erstellt. Dies ist notwendig, da sich die Anzahl der Webseiten mit jeder Softwareversion ändern kann. Durch das Erstellen einer Schnittmenge ist die Auf- und Abwärtskompatibilität entsprechender Screenshot-Sets gewährleistet. Somit können alle Seiten die Bestandteil beider Softwareversionen sind gegeneinander verglichen werden.
Danach werden die Seiten, wie in Abbildung 4.2 zu sehen, Paarweise nach dem Prinzip des Bild-basierten Vergleichsansatz verglichen, wobei ein Differenzbild erstellt wird.

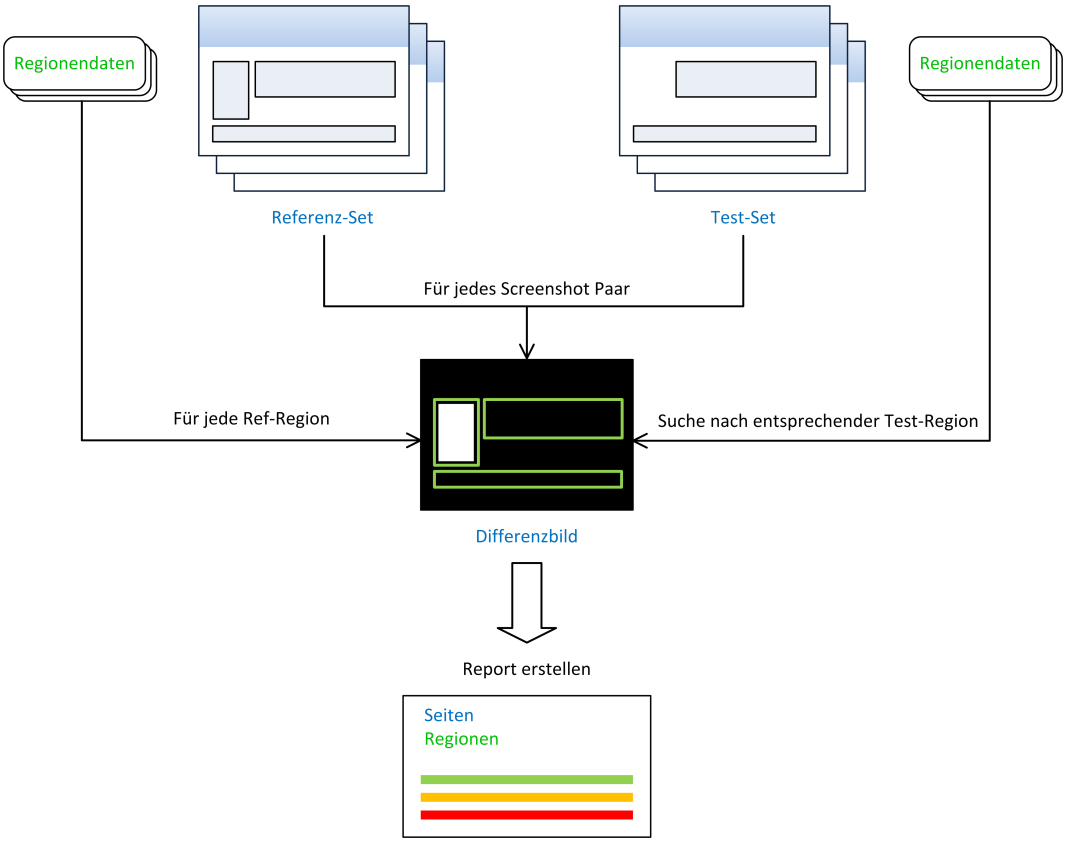

Abbildung 4.2: Vergleich zweier Screenshot-Sets unter Berücksichtigung aller Regionen und anschließendes Erstellen eines grafischen Reports.

Dabei werden die bei der Aufnahme definierten Regionen berücksichtigt. Beinhaltet ein Referenz-Screenshot Regionen, die auf dem entsprechenden Test-Screenshot nicht gefunden werden, so gilt dies als Fehler. Andersherum werden Regionen, die nur auf dem Test-Screenshot gefunden werden, ignoriert. Für alle auf beiden Screenshots gefundenen Regionen wird mit Hilfe des Differenzbildes überprüft, ob gefundene Unterschiede diese Region betreffen (dargestellt durch grüne Umrandungen in Abbildung 4.2). Überlappt eine Region mit einem gefunden Unterschied (in der Abbildung weiß dargestellt), so wird der Unterschied entsprechend dem überlappenden Bereich dieser Region zugeschrieben. Somit ist es möglich, Screenshots auf Regionenebene zu vergleichen und die Quellen der Unterschiede genauer einzugrenzen.
Abschließend wird anhand dieser Vergleiche ein grafischer Report erstellt. Die genauen Bestandteile dieses Reports werden in Abschnitt 5.5 näher erläutert.

Haben zwei zu vergleichende Screenshot Paare unterschiedliche Auflösungen, kann der Bild-basierte Vergleichsansatz nicht angewandt werden. Wie in Abbildung 4.3 zu sehen, ist es jedoch sehr ungenau, die komplette Webseite als fehlerhaft zu deklarieren, wenn der eigentliche Unterschied beispielsweise nur durch einen Cookie Hinweis entstanden ist. Der komplette Inhalt der Seite rutscht aufgrund dieses Hinweises nach unten, ist jedoch eventuell mit dem der zu vergleichenden Seite identisch.
Um die Screenshots dennoch vergleichen zu können, werden die Regionen aus beiden Screenshots ausgeschnitten und einzeln Bild-basiert verglichen (vgl. Abbildung 4.3). Diese herausgeschnittenen Regionen werden als extra Bilddatei zusätzlich zu den Screenshots gespeichert.
Obwohl dieses Screenshot Paar nach wie vor Unterschiede und damit potentielle Fehler oder Features aufweist, ist es durch dieses Vorgehen möglich, große Bereiche der Screenshots zu vergleichen. Gerade wenn sich die GUI einzelner Seiten zwischen zwei Softwareversionen stark ändert, ist es mit dieser Vorgehensweise dennoch möglich, schon vorher vorhandene Bereiche dieser Seiten zu vergleichen.

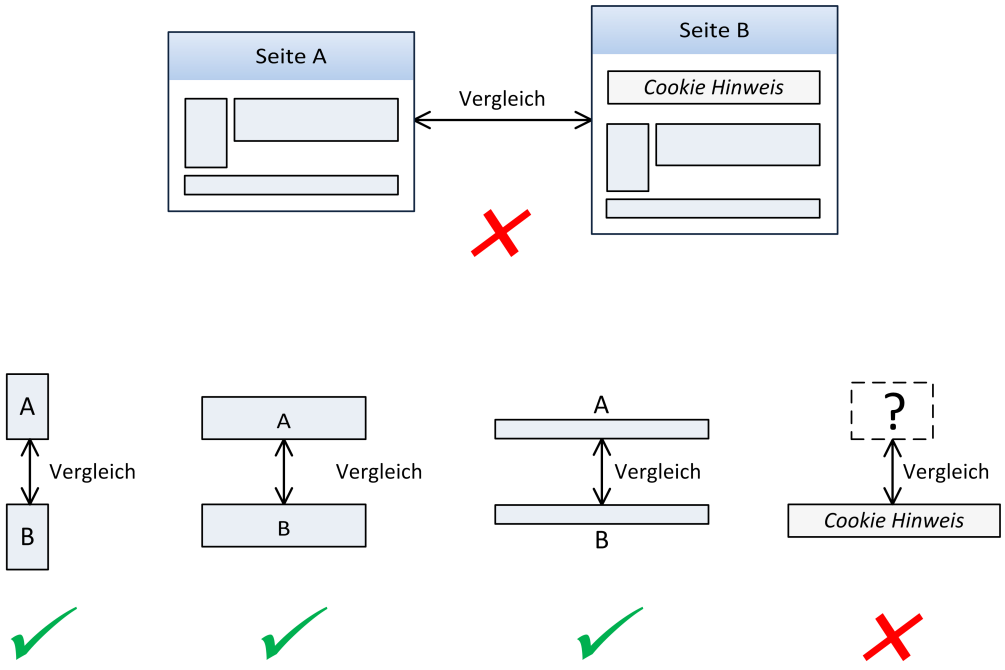

Abbildung 4.3: Als Vorgehensweise zum Vergleichen von Screenshots unterschiedlicher Auflösung werden die einzelnen Regionen Bild-basierter verglichen.

Durch den nahezu auflösungsunabhängigen Vergleich, das automatische Erstellen einer vergleichbaren Screenshot Schnittmenge und das Berücksichtigen von lediglich den auf einem Referenz-Screenshot gefundenen Regionen, sind die Vergleichsmöglichkeiten sehr flexibel. Selbst sich stark unterscheidende Screenshot-Sets sind somit (soweit möglich) vergleichbar.

4.2.2 Entitäten

Um die oben genannten Prozesse zu realisieren, kennt iCAT intern sechs normale und zwei schwache Entitäten. Diese sowie deren Zusammenhänge werden in Abbildung 4.4 in Form eines UML Entity-Relationship Diagramms dargestellt.

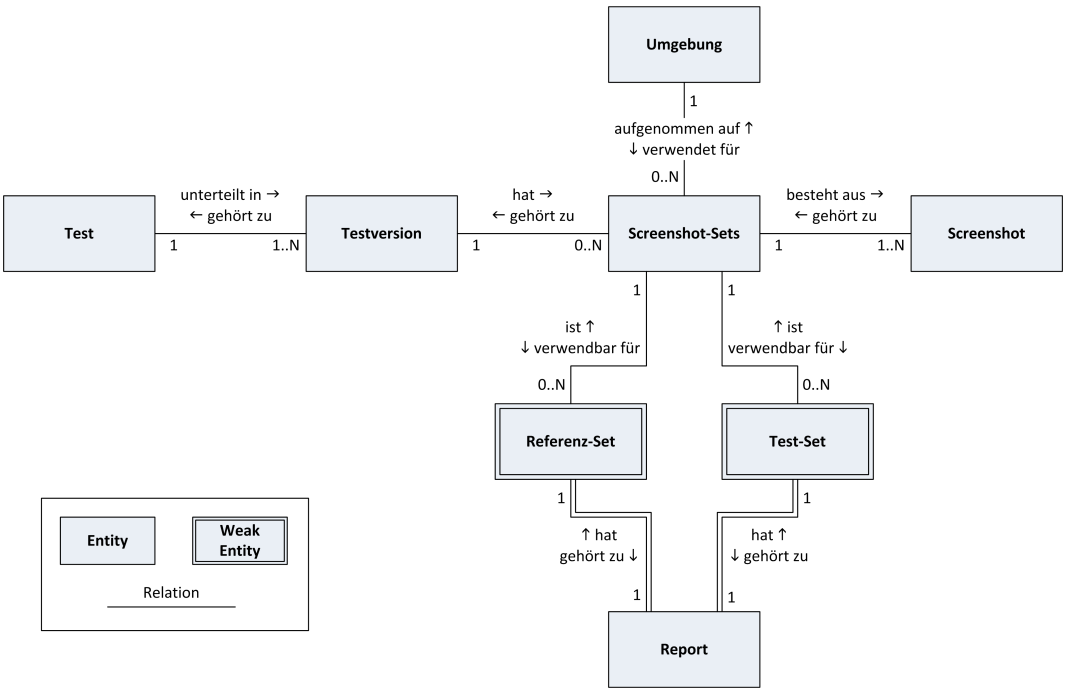

Abbildung 4.4: Zusammenhang der von iCAT verwendeten Entitäten dargestellt als UML Entity-Relationship Diagramm.

Durch diesen Aufbau ist es möglich, Screenshot-Sets zu strukturieren und Verknüpfungen zwischen einem Set, der zugrundeliegenden Softwareversion und der für die Aufnahme verwendeten Umgebung herzustellen. Dadurch können dem Benutzer folgende Funktionen geboten werden:

Umgebungen anlegen und verwalten Die bei der Aufnahme eines Screenshot-Sets benötigte Umgebung kann zunächst zentral angelegt werden. Für jede Aufnahme kann eine dieser Umgebungen ausgewählt werden.

Test anlegen Es können beliebige Tests angelegt werden (etwa einen pro Software und Testart). Dabei muss eine initiale URI- und Regionen-Liste angegeben werden. Alle Screenshot-Sets und Testausführungen der zugrundeliegenden Software werden diesem Test zugeordnet, wodurch ein Zusammenhang entsteht.

Testversionen anlegen Tests können in Versionen unterteilt werden. Eine solche Testversion entspricht bei einer eins zu eins Verknüpfung der zugrundeliegenden Softwareversion. Die für den Test angegebene URI-Liste und Regionen-Liste kann beim Anlagen einer neuen Version überarbeitet oder ersetzt werden.

Screenshot-Sets aufnehmen und vergleiche Für jede Softwareversion können nun beliebig viele Screenshot-Sets aufgenommen und danach innerhalb des selben Tests beliebig verglichen werden. Insgesamt sind folgende Vergleichsarten möglich:

- Vergleich verschiedener Softwareversionen
- Vergleich verschiedener Umgebungen
- Vergleich verschiedener Browser

5 Implementierung

Dieses Kapitel beschreibt die Implementierung des neu konzipierten iCAT. Nach der Vorstellung der verwendeten Entwicklungsumgebung und Technologien in Abschnitt 5.1 gibt Abschnitt 5.2 einen Überblick über das Softwaredesign des Tools. Daraufhin wird die Softwarearchitektur für die iCAT Anwendung (Abschnitt 5.3) und die umgesetzte Webanwendung als grafische Benutzerschnittstelle (Abschnitt 5.4) im Detail beschrieben. Im abschließenden Abschnitt 5.5 wird der bei einem Vergleich erstellte Report vorgestellt.

5.1 Entwicklungsumgebung

Als integrierte Entwicklungsumgebung (engl. *Integrated Development Environment*, kurz *IDE*) wurde die *Eclipse IDE für Java EE Developers* in Version 4.4.1 verwendet. Da *Java* als Programmiersprache schon für die Umsetzung von iCAT v1 zum Einsatz kam, war es naheliegend diese auch für die Umsetzung von iCAT v2 einzusetzen. Um das Bauen des Projekts inklusive aller Abhängigkeiten zu erleichtern, wurde *Maven* in Form des M2E (Maven Integration für Eclipse) Plugins verwendet. Als Versionskontrollsystem wurde *Git* eingesetzt. Einerseits war dadurch ein laufendes Backup möglich, andererseits wurde ein späteres Zusammenführen mehrerer Entwicklungszweige erleichtert, wodurch das MAP Team parallel mit und an der ersten Version von iCAT weiterarbeiten konnte. Um als nächstes zu implementierende Features im Team zu planen, wurde *JIRA* verwendet.
Zusätzlich kommen in der iCAT Anwendung und der Webanwendung weitere Tools und Frameworks zum Einsatz. Diese werden an entsprechender Stelle in den nachfolgenden Abschnitten vorgestellt.

5.2 Softwaredesign

Das Tool kann im Wesentlichen in zwei Bereiche unterteilt werden: Die *iCAT Anwendung*, welche die Datenschicht, Anwendungslogik und Schnittstelle zur Präsentationsschicht beinhaltet, und die *Anwendungen der Präsentationsschicht*. Nachfolgend wird zuerst die Architektur des kompletten Tools abstrakt beschrieben. In den darauf folgenden Abschnitten werden die iCAT Anwendung sowie die Webanwendung als Vertreter der Präsentationsschicht vorgestellt.

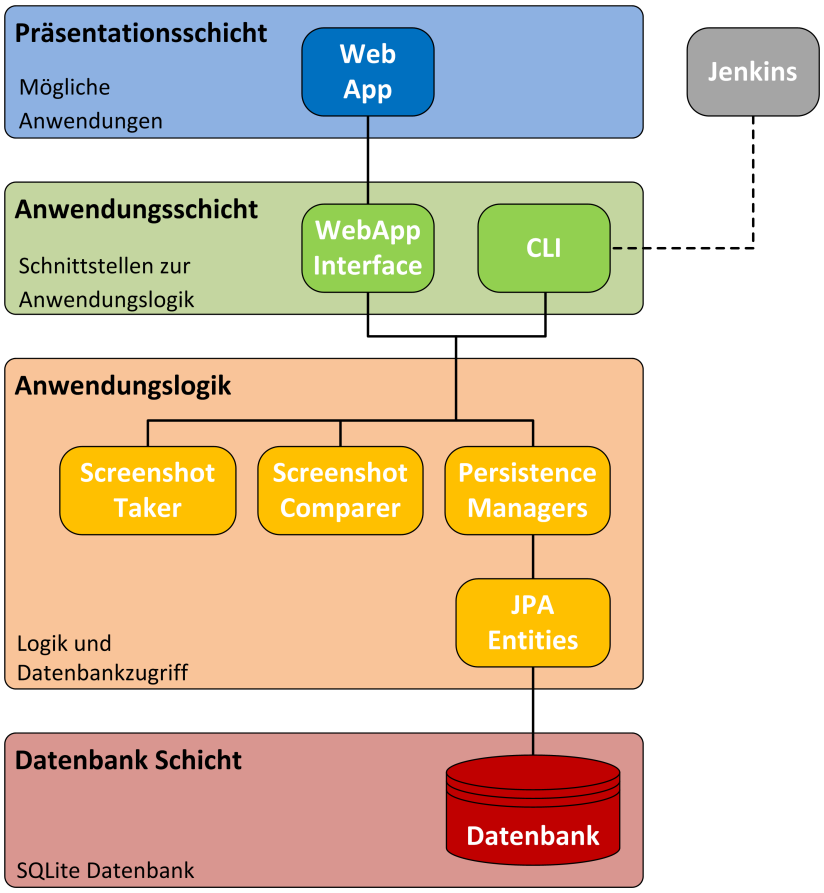

Abbildung 5.1: Abstrakte iCAT Architektur mit den wichtigsten Bestandteilen unterteilt in mehrere Schichten.

Die Architektur des neuen Ansatzes kann Abbildung 5.1 entnommen werden. Die Grafik fasst die wichtigsten Bestandteil zusammen und gliedert sie, gemäß Eric Evans Idee [11], in einer Schichtenarchitektur.

Die oberste Schicht beinhaltet dabei mögliche Anwendungen zur Interaktion. Eine Webanwendung dient dabei sowohl der grafischen Repräsentation als auch der Entgegennahme von Benutzereingaben. Die im Umfang dieser Arbeit umgesetzte Webanwendung wird in Abschnitt 5.4 beschrieben. Bei einer Integration in eine CI-Umgebung, wie beispielsweise Jenkins, wird auf eine grafische Darstellung verzichtet. Zur automatisierten Bedienung von iCAT werden nur Aufrufparameter und Ergebniswerte übergeben.
Die Anwendungsschicht ist als Schnittstelle zwischen der Anwendungslogik und Präsentationsschicht bzw. externen Anwendungen zur Präsentation und Bedienung zu verstehen. Dies ist notwendig, da die Anwendungen der Präsentationsschicht jeweils unterschiedliche Vorgehensweisen zur Bedienung externer Anwendungen (in diesem Fall iCAT) haben.
Die darunterliegende Anwendungslogik beinhaltet sämtliche Logik zum Aufnehmen und Vergleichen von Screenshots. Sie beinhaltet außerdem die Persistenzschicht, die bewusst nicht in einer eigenen Schicht realisiert wurde. Grund dafür ist die Verwendung von *EclipseLink* als Referenzimplementierung der *Java Persistence API* (JPA). Durch objektrelationale Abbildung ermöglicht diese den Zugriff auf Tabellen relationaler Datenbanken mittels *JPA Entities*, bei welchen es sich um sogenannte *POJOs* (*Plain Old Java Object*) handelt. Statt SQL Queries an einen Datenbank absetzen zu müssen, ist es möglich, wie gewohnt auf „normalen“ Java Objekten zu arbeiten, wobei die JPA das Abbilden der Objektvariablen auf Tabellenspalten übernimmt [23]. Für jedes JPA Entity gibt es eine entsprechende Manager Klasse, welche das Hinzufügen, Löschen und Aktualisieren der Entities in der Datenbank ermöglicht. Da die JPA Entities vom Rest der Anwendungslogik verwendet werden, war es naheliegend, die beiden Schichten nicht zu trennen.
Auf der untersten Schicht kommt SQLite als Datenbank zum Einsatz. Bei dieser handelt es sich um eine leichtgewichtige und serverlose SQL Datenbank, welche keinerlei Konfiguration benötigt. Auch wenn die Performanz von SQLite in Kombination mit EclipseLink vergleichsweise eher schlecht ist [19], sind die Vorteile dennoch überwiegend. Die in Abschnitt 5.3 beschriebene iCAT Anwendung soll als Kommandozeilen Anwendung ohne vorherige Konfiguration und Einrichtung einer Datenbank ausgeführt werden können. Genau dies ist mit SQLite möglich. Da außerdem das Aufnehmen und Vergleichen von Screenshots die zeitintensivsten Prozesse von iCAT darstellt, ist die Performanz der Datenbank vernachlässigbar.

iCAT verwendet die Selenium WebDriver API als Backend zum Öffnen und Bedienen des Browsers. Es werden nacheinander die Seiten aus der URI-Liste geladen und die Regionen

aus der Regionen-Liste identifiziert. Dieses Backend wurde aus iCAT v1 übernommen und nicht verändert. Daher wird es im Folgenden nicht näher beschrieben.

5.3 iCAT Anwendung

Als erstes wurde die iCAT Anwendung an sich überarbeitet. Bei dieser handelt es sich weiterhin um eine reine Java Kommandozeilenanwendung. Sie beinhaltet sowohl die komplette Anwendungslogik, als auch die Datenschicht und kann somit als Backend verstanden werden.
Diese Anwendung setzt die meisten Anforderungen aus Abschnitt 3.3 sowie die daraus abgeleitete Konzeption (vergleiche Abschnitt 4.2) um. Lediglich Anforderung A.6 (Unterstützung für agile Entwicklung) und A.7 (GUI) können nur in Kombination mit der in Abschnitt 5.4 vorgestellten Webanwendung umgesetzt werden.
Im Wesentlichen bietet es dem Nutzer zwei Funktionen: Das Aufnehmen eines neuen Screenshot-Sets und das Vergleichen zweier Sets.

Abbildung 5.2 zeigt ein UML Aktivitätsdiagramm, welches den Arbeitsablauf zum Aufnehmen eines neuen Screenshots-Sets beschriebt. Der Einfachheitshalber geht es nur auf die grundlegenden Schritte ein. So ist beispielsweise die in Abschnitt 4.2.1 beschriebene Angabe eines Browser und einer Umgebung nicht dargestellt.
Zu Beginn der Aufnahme öffnet iCAT einen Browser. Um sicherzustellen, dass im Browser-Zwischenspeicher (Cache) abgelegte Daten keinen Einfluss auf spätere Set Aufnahmen haben, wird der Browser zuvor jeweils auf den Auslieferungszustand zurückgesetzt. iCAT öffnet jede in der URI-Liste angegebene Webseite und wartet bis diese vollständig geladen ist. Nach der Aufnahme des Screenshots in Form einer PNG Datei wird auf der Webseite nach allen in de Regionen-Liste definierten Regionen gesucht und für alle gefundenen Regionen die Position und Dimension gespeichert. Die genaue Umsetzung der Screenshot- und Datenspeicherung wird in Abschnitt 5.3.3 beschrieben.

Der Arbeitsablauf zum Vergleichen zweier Screenshot-Sets wird durch das in Abbildung 5.3 dargestellte UML Aktivitätsdiagramm beschrieben. Auch hier werden einfachheitshalber Arbeitsschritte wie das Erstellen einer vergleichbaren Screenshot-Schnittmenge (vergleiche Abschnitt 4.2.1) nicht dargestellt.

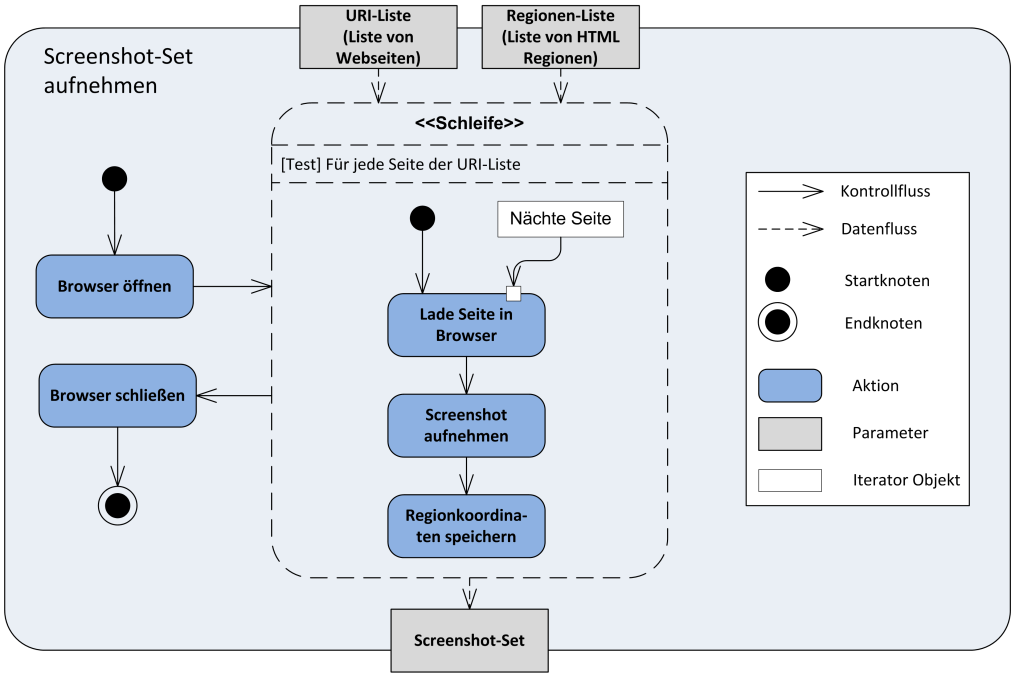

Abbildung 5.2: Das Aufnehmen eines Screenshot-Sets dargestellt als UML Aktivitätsdiagramm.

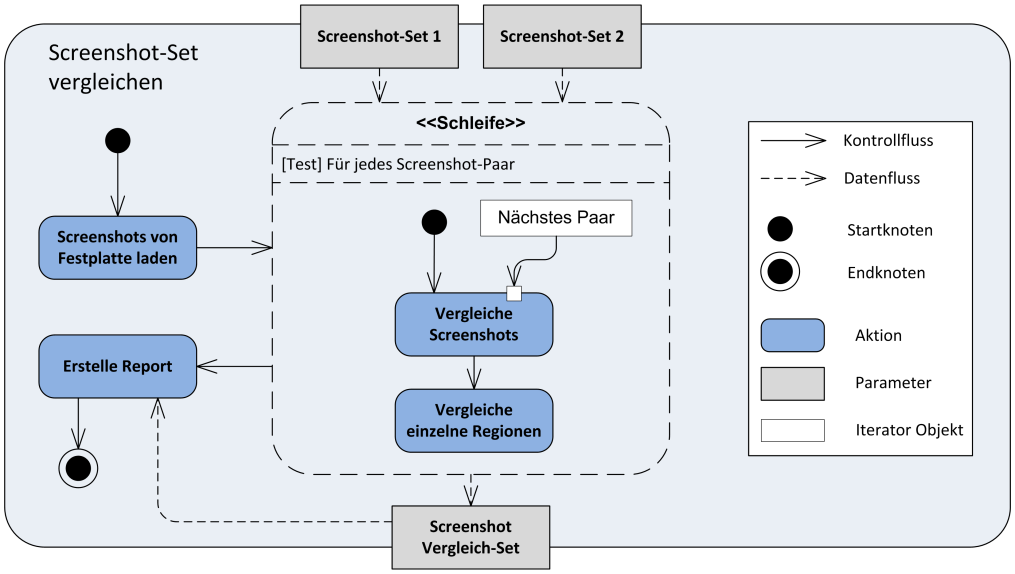

Abbildung 5.3: Das Vergleichen zweier Screenshot-Sets dargestellt als UML Aktivitätsdiagramm.

Nach dem Laden der beiden Screenshot-Sets werde die Screenshots beider Sets paarweise verglichen. Für jedes Paar werden weiterhin die einzelnen Regionen verglichen. Dadurch wird ein sogenanntes Vergleichs-Set erstellt. Jeder Eintrag in diesem Set enthält die Ergebnisse des Vergleichs eines Screenshot-Paars samt Regionen. Mithilfe dieses Vergleichs-Set wird anschließend ein Report in Form einer HTML bzw. XHTML Seite erstellt. Dieser wird in Abschnitt 5.5 genauer beschrieben.

5.3.1 Konfiguration

Die Konfiguration von iCAT kann auf zwei Arten erfolgen: Mittels Parameterübergabe in der Anwendungsschicht (kommt für die Webanwendung zum Einsatz) oder durch die Angabe einer Konfigurationsdatei (kommt für die iCAT Anwendung zum Einsatz).
Bei letzterer handelt es sich um eine Java `Properties` Datei, welche eine Reihe an Schlüssel-Wert-Paaren beinhaltet, beispielsweise den zu verwendenden Browser oder die Umgebung. Der Inhalt der `icat.properties` genannten Datei kann dem Anhang A entnommen werden.

5.3.2 Softwarearchitektur

Alle relevanten Klassen sowie deren Zusammenhänge werden in Abbildung 5.4 dargestellt. Zu Gunsten eines erleichterten Verständnisses sind einige „Helferklassen“, JPA Entities, die von der Anwendungslogik nicht verwendet werden, alle JPA Manager Klassen sowie die mit Selenium zusammenhängenden Klassen (vergleiche Abschnitt 5.2) nicht dargestellt. Alle dargestellten Klassen beinhalten wiederum lediglich die wichtigsten Attribute und Operationen.
Die `Standalone` und `WebAppInterface` Klasse sind Bestandteil der Anwendungsschicht. Erstere entspricht dabei der in Abbildung 5.1 dargestellten CLI, letztere des WebApp Interfaces. Beide sind somit als Schnittstelle zur Präsentationsschicht zu verstehen und bieten daher im Wesentlichen jeweils eine Methode zum Aufnehmen eines neuen Screenshot-Sets und eine zum Vergleichen zweier Sets. Die `main()` Methode der `Standalone` Klasse erwartet als Parameter den Pfad zur Konfigurationsdatei, in welcher unter anderem der Ausführungsmodus (Aufnahme oder Vergleich) angegeben ist.
Je nach Methode wird entweder auf den `ScreenshotTaker` oder den `Screenshot-`

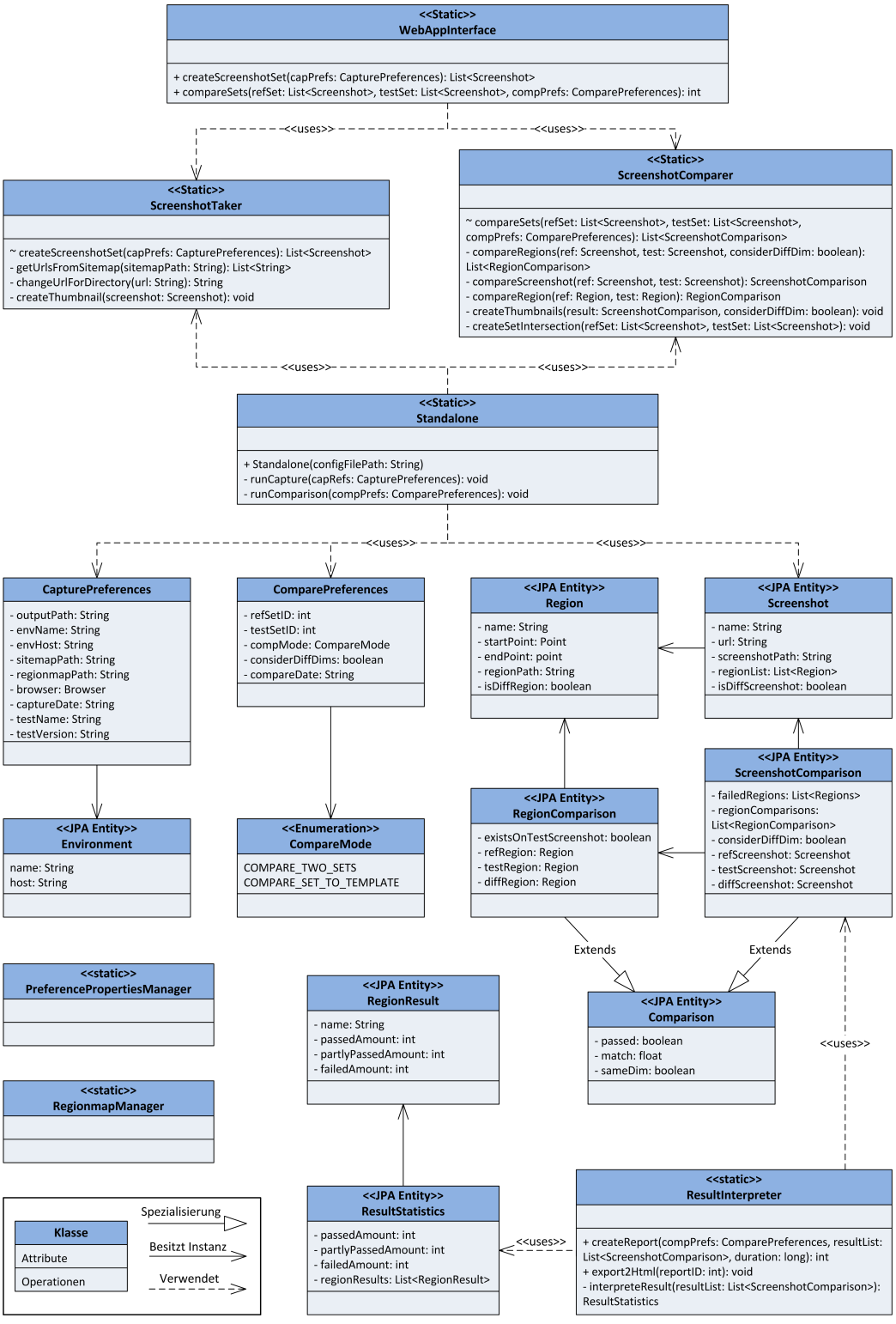

Abbildung 5.4: UML Klassendiagramm mit den wichtigsten Klassen der iCAT Anwendung.

`Comparer` zugegriffen. Ersterer bekommt dabei eine `CapturePreferences` Instanz übergeben, welche die für die Aufnahme relevanten Informationen enthält, unter anderem eine `Environment` Instanz mit dem Namen und Host der Umgebung. Der `ScreenshotTaker` bekommt eine `ComparePreferences` Instanz übergeben, welche die relevanten Informationen für den Vergleich enthält. Dazu gehört auch eine Instanz des `CompareMode`.
`ScreenshotTaker` und `ScreenshotComparer` arbeiten beide mit Listen von `Screenshot` und `Region` Instanzen sowie Instanzen der entsprechenden von `Comparison` erbenden Klassen. Erstere beinhalten alle für eine(n) Screenshot bzw. Region relevanten Metadaten und letztere die für ein Vergleichsergebnis einer/s Screenshots bzw. Region relevanten Metadaten.
Nach einem Vergleich greifen sowohl `Standalone` als auch `WebAppInterface` auf den `ResultInterpreter` zu. Mithilfe der in `ResultStatistics` und `RegionResult` enthaltenden Informationen zum Vergleich erstellt dieser einen Report.

5.3.3 Datenspeicherung

Wie in Abschnitt 5.2 bereits erwähnt, arbeitet die iCAT Anwendung auf einer SQLite Datenbank. Da in der Präsentationsschicht theoretisch beliebig viele Anwendungen denkbar sind, war es naheliegend, die Datenschicht in der iCAT Anwendung umzusetzen, statt in jede Anwendung der Präsentationsschicht eine eigene Datenschicht zu integrieren. Durch die Verwendung von SQLite ist weiterhin keine Datenbankkonfiguration und -installation erforderlich.
Durch die Verwendung der JPA ist es möglich, die zu verwendende Datenbank an einer Stelle zentral festzulegen. Dadurch ist ein späterer Austausch selbiger leicht möglich.
In der Datenbank selber werden alle während einer Aufnahme oder eines Vergleichs anfallenden Daten gespeichert. Einzig die Bilddateien und HTML Reports werden direkt im Dateisystem abgelegt. In der Datenbank sind jeweils deren Dateipfade gespeichert.

Abbildung 5.5 stellt die Datenbankstruktur der iCAT Anwendung in Form eines *UML Entity-Relationship-Modells* dar. Jedes Entity dieses Modells entspricht dabei einem JPA Entity. Für jedes Entity werden zuerst der Primärschlüssel (engl. *Primary Key*, kurz *PK*) und die Fremdschlüssel (engl. *Foreign Key*, kurz *FK*) und danach die Attribute angegeben. Durch

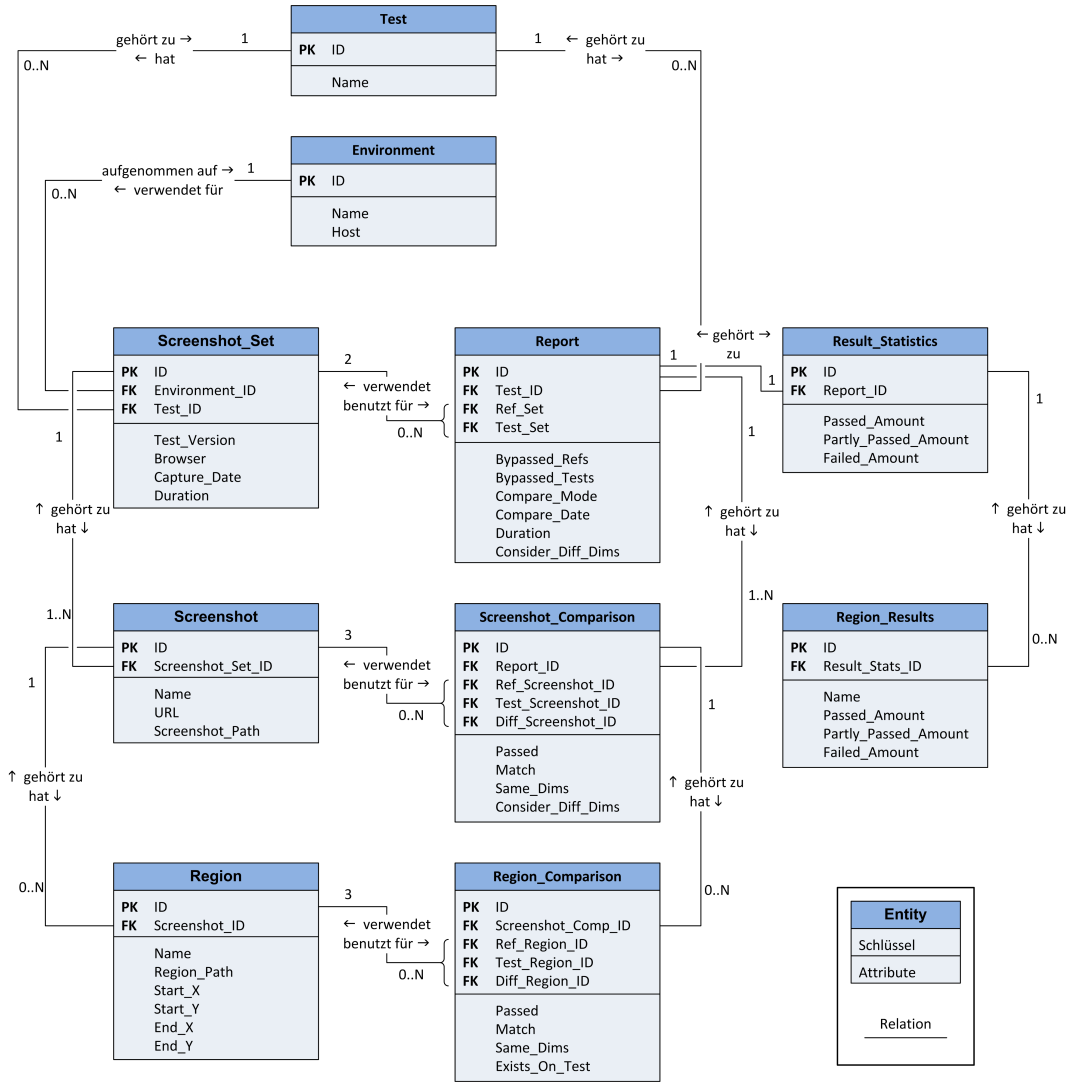

Abbildung 5.5: Datenbankstruktur der iCAT Anwendung in Form eines UML ER-Modells.

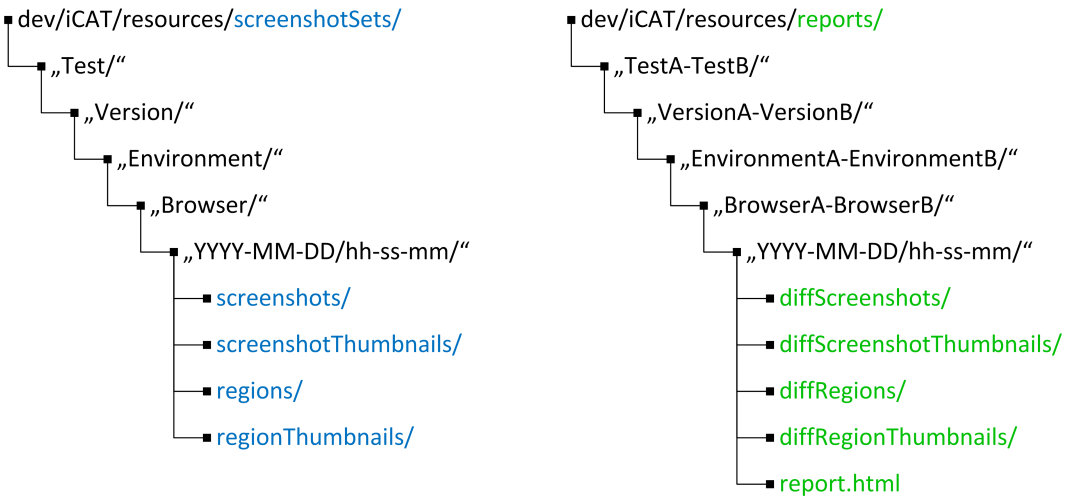

Abbildung 5.6: Die Verzeichnisstruktur für Bilddateien (Screenshots und Regionen) sowie Reports.

die Relationen wird dargestellt, wie die Entities zueinander in Verbindung stehen und auf welche Tabellenspalte ein FK jeweils zeigt.

Die Verzeichnisstruktur, nach der die Bilddateien für Screenshots und Regionen sowie die HTML Reports im Dateisystem gespeichert werden, ist in Abbildung 5.6 dargestellt.
Im *resources* Ordner befinden sich zwei Unterordner. Im *screenshotSets* Unterordner sind alle aufgenommenen Screenshot-Sets in Form von Bilddateien für Screenshots und Regionen und deren Thumbnails gespeichert. Um ein Set auch beim Durchsuchen des Dateisystems eindeutig zuordnen zu können, wird die Struktur entsprechend *Testnamen*, *Testversion*, *Umgebungsnamen*, *Browsername* und *Aufnahmedatum* aufgebaut (vgl. linke Seite der Abbildung 5.6).
Im *reports* Unterordner befinden sich alle Reports in Form der HTML Datei und Bilddateien für Differenzbilder von Screenshots und Regionen. Da ein Screenshot-Set zu beliebig vielen Reports gehören kann (und somit für jeden Screenshot dieses Sets beliebig viele Differenzbilder existieren können), ergibt es Sinn, diese Differenzbilder zusammen mit den Reports und nicht mit den Screenshot-Sets abzulegen. Ähnlich wie bei den Screenshot-Sets ist auch die Struktur für die Reports anhand der Namen von *Test*, *Testversion*, *Umgebung* und *Browser* sowie dem *Vergleichsdatum* aufgebaut (vgl. rechte Seite der Abbildung 5.6).

5.4 Webanwendung

Nach der erfolgreichen Fertigstellung der iCAT Anwendung wurde im nächsten Schritt die existierende Webanwendung so überarbeitet, dass sie als GUI für iCAT v2 verwendet werden kann.
Die Vorteile einer solchen Webanwendung sind zum einen die erleichterte Bedienung und Konfiguration im Vergleich zur Kommandozeilen Schnittstelle der iCAT Anwendung. Auch Nicht-Entwicklern ist es somit möglich, iCAT zu verwenden.
Zum anderen kann die Strukturierung von Screenshot-Sets in Tests und Testversionen nur mit eine GUI übersichtlich und sinnvoll dargestellt werden. Um für einen Vergleich anhand von Umgebung und Testversion die richtigen Screenshot-Sets auswählen zu können, ist es sehr hilfreich, diese zuvor entsprechend gruppiert dargestellt zu bekommen.
Außerdem kann mit einer GUI übersichtlich dargestellt werden, welche Reports zu welchem Screenshot-Set und welcher Softwareversion gehören. Dadurch kann leicht nachvollzogen werden, wann neue Features oder Fehler sowie deren Beseitigung auftraten.
Letztendlich können durch das Bereitstellen der Webanwendung auf einem Server mehrere Nutzer gleichzeitig auf iCAT zugreifen und dadurch mit den selben Daten arbeiten. Somit muss iCAT auch nicht auf dem PC jedes Testers installiert und konfiguriert werden.

Für die Entwicklung der Webanwendung wurde die selbe Entwicklungsumgebung verwendet, wie schon für die iCAT Anwendung (vergleiche Abschnitt 5.1). Als Anwendungsserver kam die open source Version von *GlassFish* [13] zum Einsatz. Dieser unterstützt in der verwendeten Version 4.1 die Java EE 7 Plattform. Zum Installieren der Webanwendung auf den GlassFish Server wurde das *maven-glassfish-plugin* [17] verwendet. Die Maven Integration für Eclipse wurde abermals verwendet, um das Projekt inklusive aller Abhängigkeiten automatisiert zu bauen.

5.4.1 Softwarearchitektur und verwendete Technologien

Die Webanwendung wurde mit der Java Enterprise Edition (Java EE) erstellt. Das Java Server Faces (JSF) Framework *PrimeFaces* [27] kam für die Benutzerschnittstelle zum Einsatz. Dieses ermöglicht es vorgefertigte JSF Komponenten in XHTML Webseiten einzubauen. Das *Bootstrap* CSS-Framework [5] wurde in Form des Bootstrap Themes für PrimeFaces

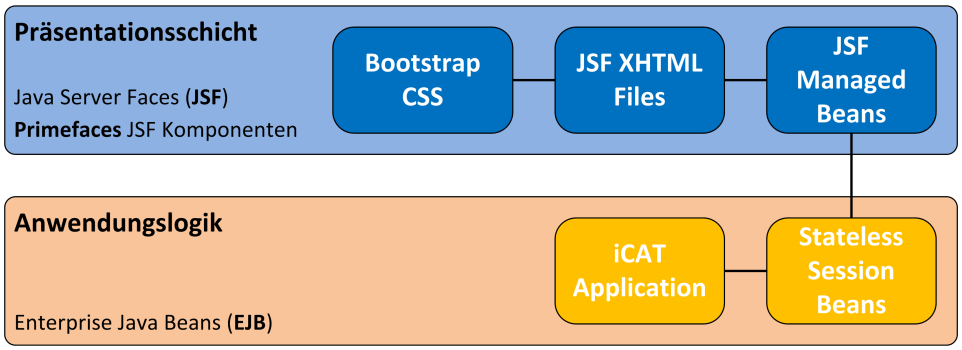

Abbildung 5.7: Architektur der Webanwendung unterteilt in Schichten.

für die Formatierung und das Layout der XHTML Dateien verwendet. Die Datenübertragung zwischen Client und Server wird (abgesehen vom Weiterleiten auf eine andere Seite) durch AJAX realisiert. Enterprise Java Beans (EJB) kamen für die Implementierung der Logik der Webanwendung zum Einsatz. Diese verwenden außerdem die iCAT Anwendung für das Aufnehmen und Vergleichen von Screenshot-Sets sowie für das Abspeichern von Reports. Die einzelnen Schichten der Webanwendung sowie der Zusammenhang der einzelnen Komponenten werden in Abbildung 5.7 dargestellt.

Für jede Facelets XHTML Webseite wurde eine JSF Managed Bean Klasse erstellt. Diese beinhalten Variablen und Felder samt Getter und Setter sowie alle Methoden, die von den Facelets verwendet werden. Über die Managed Beans wird mithilfe der Dependency Injection (`@EJB` Annotation) auch auf die EJB Session Beans referenziert.

Die Webanwendung verwendet asynchrone Methodenaufrufe für langwierige Prozesse (Aufnehmen und Vergleichen von Screenshot-Sets). Dies wird über die `@Asynchronous` Annotation der EJB 3.1 Spezifikation erreicht. Keine asynchronen Methodenaufrufe für diese Prozesse zu verwenden, würde die Benutzerschnittstelle während des gesamten Prozesses blockieren und den Anschein erwecken, die Anwendung sei abgestürzt oder reagiere nicht. Während der Aufnahme- oder Vergleichsprozess ausgeführt wird, wird der PrimeFaces AJAX polling Mechanismus verwendet, um zu überprüfen, ob der Prozess bereits beendet ist. Da die HTTP 1.1 Spezifikation keine Server initiierten Datenübertragungen vorsieht, ruft dieser polling Mechanismus clientseitig periodisch eine entsprechende Methode der Managed Bean auf und überprüft, ob das Ergebnis des asynchron ausgeführten Prozesses

bereits verfügbar ist. Ist dies der Fall, wird der Client zum erstellten Screenshot-Set oder Report weitergeleitet.

5.4.2 Aufbau einzelner Ansichten

Die Webanwendung besteht aus vier Webseiten:

- Die Umgebungsseite (*environment.xhtml*)
- Die Testübersichtsseite (*testOverview.xhtml*)
- Die Testseite (*test.xhtml*)
- Die Reportseite (*report.xhtml*)

Im Folgenden werden Screenshots der Umgebungs-, Testübersichts- und Testseite vorgestellt. Der Report wird in Abschnitt 5.5 genauer erklärt.
Das Basislayout aller Seiten wird durch eine Templateseite (*layout.xhtml*) vorgegeben. In dieser wurde auch die Navigationsleiste realisiert, die immer oben auf jeder Seite angezeigt wird.

Die Umgebungsseite (Abbildung 5.8) listet alle bisher angelegten Umgebungen mit Namen und Host auf. Über den „Add new Environment“ Button ist es möglich, eine neue Umgebung über ein Formular anzulegen. Durch einen Rechtsklick auf eine existierende Umgebung wird ein Kontextmenü geöffnet, über welches die Umgebung gelöscht werden kann.

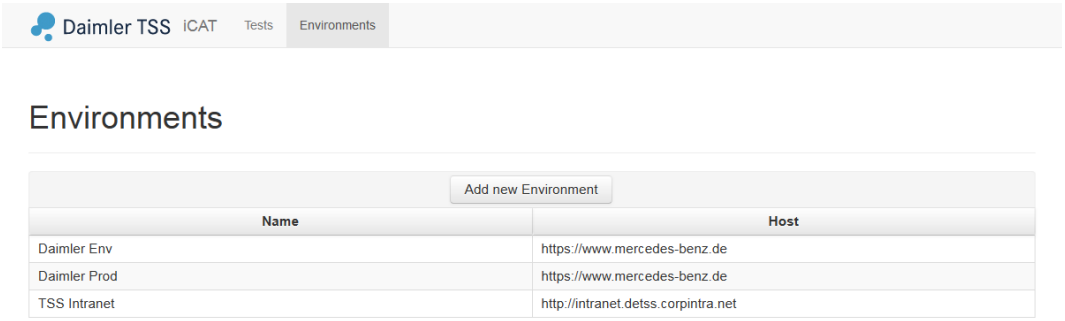

Abbildung 5.8: Webseite zum Anlegen und Löschen von Umgebungen.

Die Testübersichtsseite (Abbildung 5.9) listet alle bisher angelegten Test mit Namen und der aktueller Testversion auf. Über den „Add new Test“ Button kann ein neuer Test über ein Formular angelegt werden. Dafür muss dem neuen Test ein eindeutiger Name zugewiesen werden. Weiterhin muss die initiale Testversion angegeben und für diese eine URI- sowie Regionen-Liste hochgeladen werden. Über das mit einem rechten Mausklick erreichbare Kontextmenü kann der Test geöffnet oder gelöscht werden.

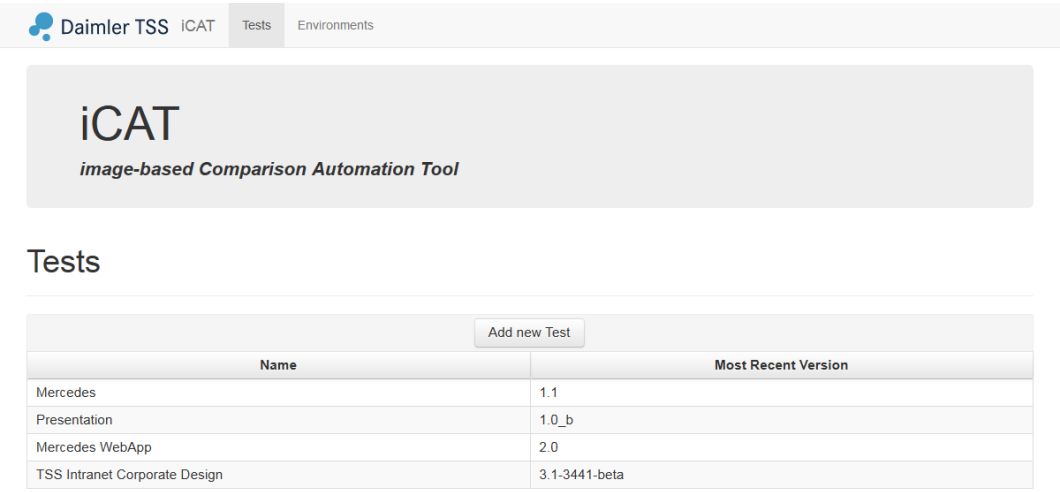

Abbildung 5.9: Webseite zum Anlegen und Löschen von Tests.

Die Testseite (Abbildung 5.10) zeigt alle für diesen Test angelegten Testversionen. Jeder Version sind die dafür aufgenommenen Screenshot-Sets zugeordnet. Über den „New Version“ Button kann eine neue Testversion angelegt werden. Über ein Formular kann für diese Version eine neue URI- und Regionen-Liste hochgeladen werden. Wird keine neue Liste hochgeladen, wird die momentan aktuelle Liste auch für die neue Version verwendet.
Für jede Version können beliebig viele Screenshot-Sets aufgenommen werden. Mit einem Klick auf den „New Set“ Button öffnet sich ein Formular, in dem der für das neue Set zu verwendende Browser und die Umgebung angegeben werden muss. Danach wird das Set wie in Abschnitt 4.2.1 dargestellt aufgenommen.
Sobald zwei oder mehr solcher Sets aufgenommen wurden, können diese, wie in Abschnitt 4.2.1 dargestellt, verglichen werden. Mit einem Klick auf den „Ref“ Button wird das Referenz-Set ausgewählt. Danach werden alle „Test“ Buttons (mit Ausnahme des Sets, welches als Referenz gewählt wurde) aktiviert. Nach der Auswahl eines Test-Sets erscheint ein Formular, in dem angegeben werden kann, ob bei unterschiedlicher Screenshot Auflösung

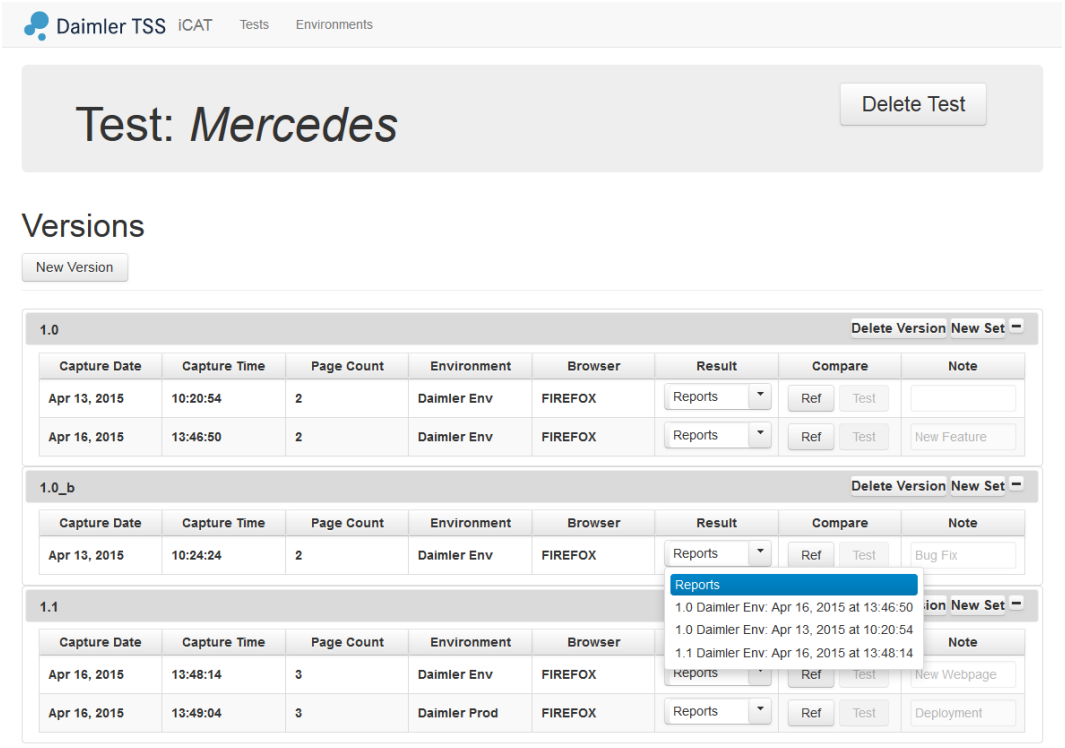

Abbildung 5.10: Webseite zum Anlegen und Löschen von Testversionen sowie zum Aufnehmen und Vergleichen von Screenshot-Sets.

ein Regionenvergleich stattfinden soll (vergleiche 4.2.1). Da dieser Regionenvergleich sehr zeitintensiv ist, kann er hier deaktiviert werden.

Sobald für ein Screenshot-Set Vergleiche ausgeführt wurden, kann der dadurch entstandenen Report von beiden verwendeten Screenshot-Sets aus über die Dropdown-Liste in der „Result“ Spalte erreicht werden.

5.5 Report

Die Ergebnisse eines Vergleichs zweier Screenshot-Sets werden in einem grafischen Report in Form einer einzelnen HTML/XHTML Seite dargestellt. Für die iCAT Anwendung wird diese nach einem Vergleich automatisch mit Hilfe der *FreeMarker* Template Engine [12] erstellt. Bei der Verwendung der Webanwendung wird keine HTML Seite erstellt, sondern der Report direkt in der Webanwendung angezeigt. Der Report besteht aus den folgenden drei Teilen.

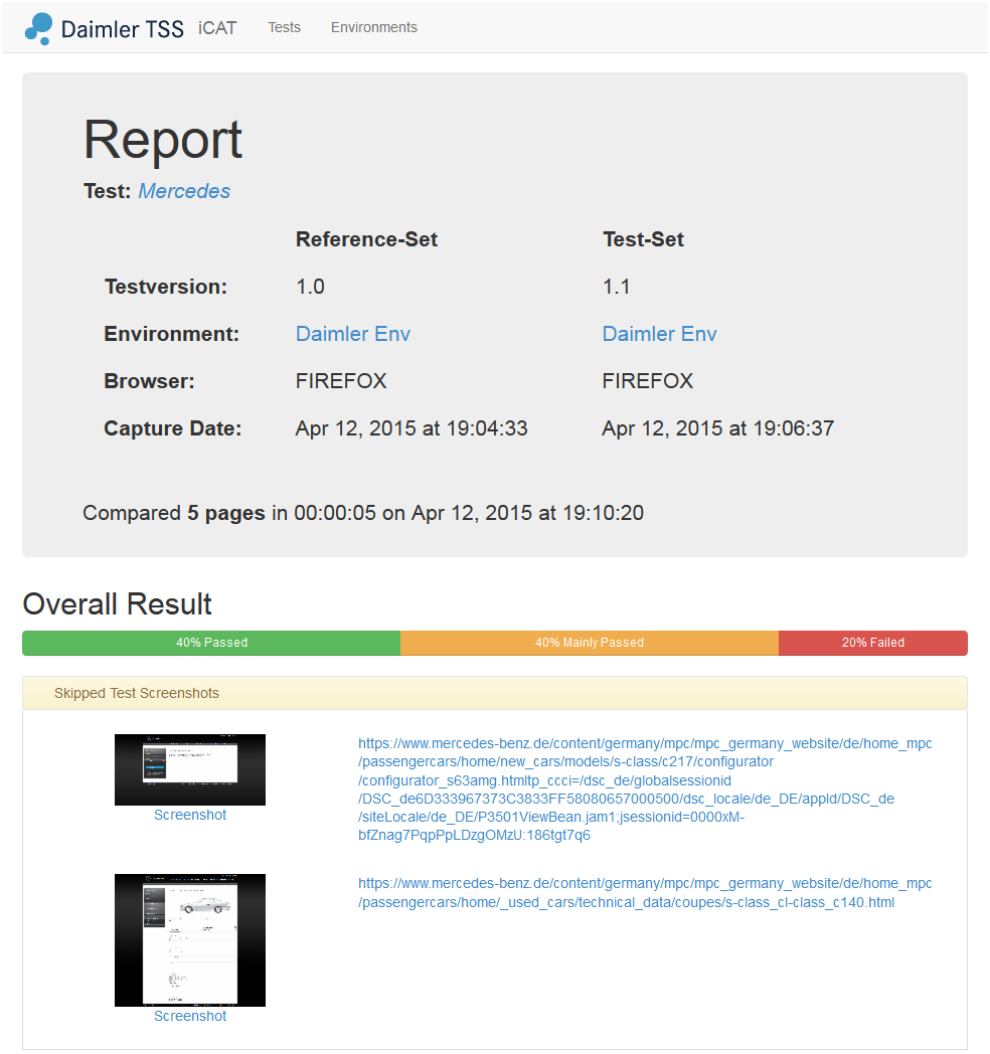

Abbildung 5.11: Oberer Teil des Reports mit Informationen zu den Screenshot-Sets und dem Vergleich sowie dem Gesamtresultat und übersprungenen Webseiten.

Im oberen Teil des Reports (Abbildung 5.11) werden Informationen zu den beiden Screenshot-Sets sowie zum Vergleich selber angezeigt. Diese beinhalten Name und Version des Tests, die Umgebung, den Browser und das Aufnahmedatum der beiden Sets sowie Datum und Dauer des Vergleichs und die Anzahl der verglichenen Seiten. Danach wird die Anzahl an erfolgreichen (*passed*), größtenteils erfolgreichen (*mainly passed*) und fehlgeschlagenen (*failed*) Vergleiche in Prozent angegeben. Ein Vergleich ist „größtenteils erfolgreich“, wenn mehr als ein bestimmter, festlegbarer Prozentsatz beider Screenshots übereinstimmen. Haben Screenshots unterschiedliche Auflösungen, müssen alle Regionen dieser Screenshots im Mittel mindestens diesen Prozentsatz an Übereinstimmung haben. Für den abgebildeten Report lag dieser bei 80 Prozent. Als letztes werden im oberen Teil die (sofern vorhanden) Seiten beider Sets aufgelistet, die durch das Erstellen einer vergleichbaren Schnittmenge übersprungen wurden (vergleiche Abschnitt 4.2).

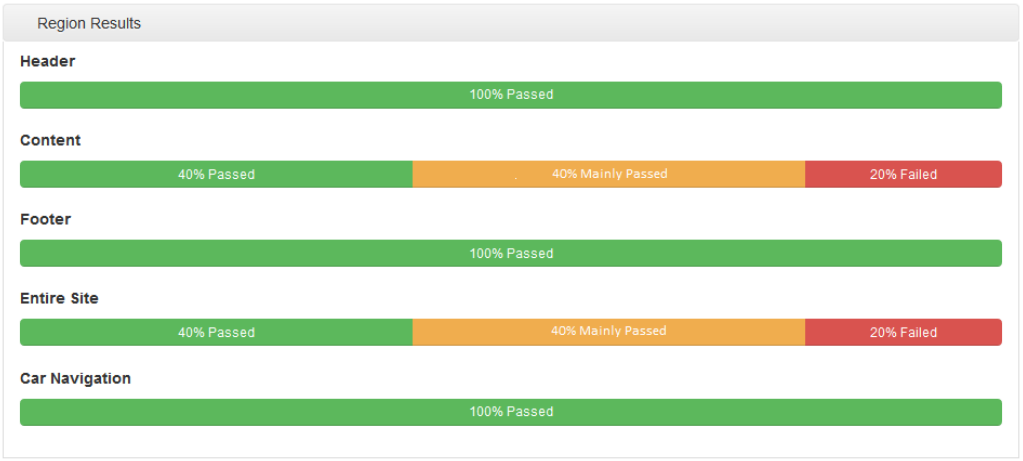

Abbildung 5.12: Mittlerer Teil des Reports mit der Anzahl erfolgreicher, größtenteils erfolgreicher und fehlgeschlagener Regionenvergleiche in Prozent.

Im mittleren Teil (Abbildung 5.12) werden für alle in der Regionen-Liste angegebenen Regionen die Anzahl erfolgreicher, größtenteils erfolgreicher und fehlgeschlagener Vergleiche auch in Prozent angegeben. Regionen, die auf einem Referenz-Screenshot nicht gefunden wurden, werden dabei ignoriert. Regionen, die auf einem Test-Screenshot nicht gefunden wurden, gelten als fehlgeschlagen. Auch bei Regionen muss die Übereinstimmung mindestens dem festgelegten Prozentsatz entsprechen, sodass der Vergleich als „größtenteils erfolgreich“ gilt.

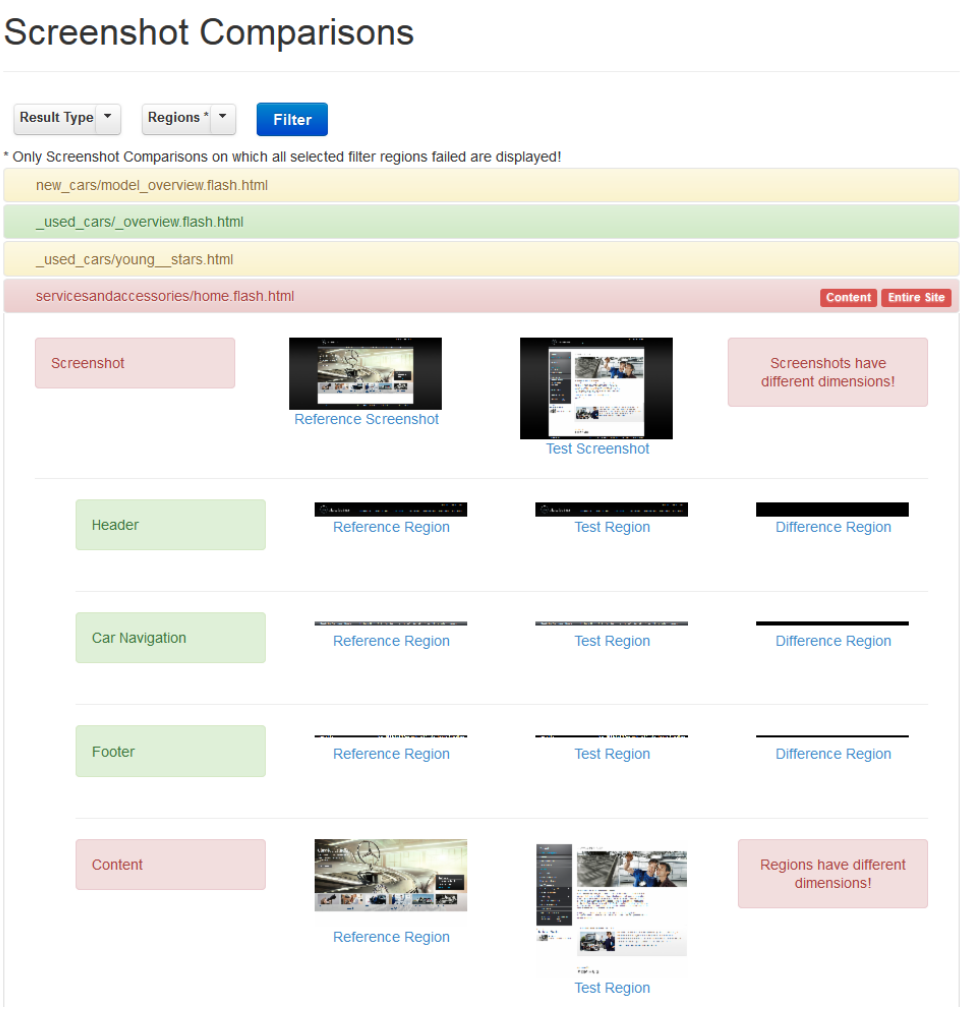

Abbildung 5.13: Unterer Teil des Reports mit den Referenz-, Test- und Differenzbilder sowie weiterer Informationen aller Screenshot Vergleiche.

Im unteren Teil (Abbildung 5.13) werden die Vergleiche der einzelnen Screenshots aufgelistet. Die Screenshot Vergleiche können nach den drei Kategorien (erfolgreich, größtenteils erfolgreich und fehlgeschlagen) sowie den fehlgeschlagenen Regionen gefiltert werden. Jeder Vergleich ist entsprechend der Kategorie mit einer Farbe hinterlegt und es werden fehlgeschlagene Regionen dieses Vergleichs aufgelistet. Für jeden Screenshot Vergleich werden Referenz-, Test- und Differenzbild angezeigt. Hat ein Screenshot Paar unterschiedliche Auflösungen, so wird zusätzlich für jede einzelne Region ein Referenz-, Test- und Differenzbild angezeigt. Alle Bilder können durch einen Klick in Originalgröße angezeigt werden.

6 Fazit

Nach der Konzeption und Umsetzung aller im Rahmen dieser Bachelorarbeit wichtigen Aspekte wird das daraus entstandene Tool *iCAT v2* nachfolgend zusammengefasst und bewertet. Im abschließenden Ausblick wird die Weiterentwicklung inklusive neuer Einsatzmöglichkeiten und Funktionen vorgestellt.

6.1 Zusammenfassung und Evaluation der umgesetzten Anwendung

Im Umfang dieser Arbeit wurde ein überarbeiteter und erweiterter Ansatz des *image-based Comparison Automation Tools* (iCAT) von Grund auf neu konzipiert und umgesetzt. Die *iCAT v2* genannte Version wurde so konzipiert, dass sie in der Praxis nun auch tatsächlich operativ eingesetzt werden kann. Neben dem nach wie vor pixelgenauen und automatisierten Vergleich besitzt iCAT nun weiterhin folgende Eigenschaften:

Sehr generisch und für verschiedene Projekte einsetzbar: Es können nun Screenshot-Sets von beliebigen Webanwendungen aufgenommen werden. Dies liegt zum einen an der Angabe von Webseiten und Regionen in entsprechenden Listen. Zusammen mit der Angabe einer beliebigen Umgebung kann damit jede mögliche Webanwendung abgedeckt werden, solange die einzelnen Seiten über eine URI abbildbar sind. Zum anderen wird die Verzeichnisstruktur für abgespeicherte Screenshots auf dem Dateisystem nun nicht mehr anhand eines projektspezifischen URI Musters aufgebaut.

Flexible Vergleichsmöglichkeiten: Jedes aufgenommene Screenshot-Set kann für zukünftige Vergleiche sowohl als Referenz-Set als auch als Test-Set verwendet werden. Zusätzlich zum Vergleich verschiedener Umgebungen und Browser ist es somit auch

leicht möglich, verschiedene Softwareversionen zu vergleichen. Dafür muss lediglich jeweils ein Screenshot-Set der entsprechenden Softwareversionen ausgewählt und für den Vergleich entweder als Referenz- oder Test-Set definiert werden. Für Softwareversionen, deren GUIs sich stark unterscheiden, bietet iCAT die Möglichkeit auch Screenshots unterschiedlicher Auflösung sowie Screenshot-Sets unterschiedlicher Seiten- und Regionenanzahl zu vergleichen.

In agiler Entwicklung gut einsetzbar: Um die aufgenommenen Screenshot-Sets zu strukturieren und logisch zu gruppieren bietet iCAT zwei Arten von Unterteilungen an. Zuerst werden alle Screenshot-Sets, die sich auf die gleiche Software beziehen, dem selben Test zugeordnet. Dieser Test ist wiederum in Versionen unterteilt, wodurch eine Verknüpfung von Testversion und Softwareversion hergestellt werden kann. Jedes Screenshot-Set, das einer bestimmten Testversion zugeordnet ist, bezieht sich somit auf die zugrunde liegende Softwareversion und beinhaltet somit auch genau die Features dieser Softwareversion. Durch diese Zusammenhänge können durch Vergleiche identifizierte Features oder Fehler sowie deren Behebung leicht nachvollzogen werden.

Die iCAT Anwendung beinhaltet eine eigene Daten- und Anwendungsschicht. Dadurch ist es möglich, alle für das Aufnehmen und Vergleichen von Screenshot-Sets benötigen Daten zentral zu verwalten. Somit können in der Präsentationsschicht beliebig viele Anwendungen umgesetzt werden, die alle auf die gleiche Datenschicht der iCAT Anwendung zugreifen und somit keine eigene Datenschicht benötigen. Dadurch ist die iCAT Anwendung ausreichend für die Bedienung über eine GUI und Integration in eine CI-Umgebung vorbereitet.

Durch die Neukonzeption ist iCAT nun auch in der Praxis operativ einsetzbar. Dies kann beispielsweise der Abbildung 6.1 entnommen werden. Das darin abgebildete Diagramm stellt den geplanten Einsatz von iCAT im MAP Team dar, also dem Team, für welches das Tool ursprünglich entwickelt wurde. Werden bei einer Testdurchführung keine Unterschiede gefunden, so soll das aktuelle Test-Set zukünftig als Referenz-Set verwendet werden können. Werden hingegen Unterschiede gefunden, muss zunächst manuell überprüft werden, ob es sich dabei um gewollte Features oder ungewollte Fehler handelt. Falls die Unterschiede gewollt sind, soll dieses Test-Set zukünftig als Referenz-Set verwendet werden können. Falls sie nicht gewollt sind, wird ein Bug erstellt und nach dessen Behebung der Test erneut durchgeführt. Dieses Vorgehen ist mit iCAT nun möglich und wird durch die leicht zu bedienende Webanwendung auch effizient unterstützt.

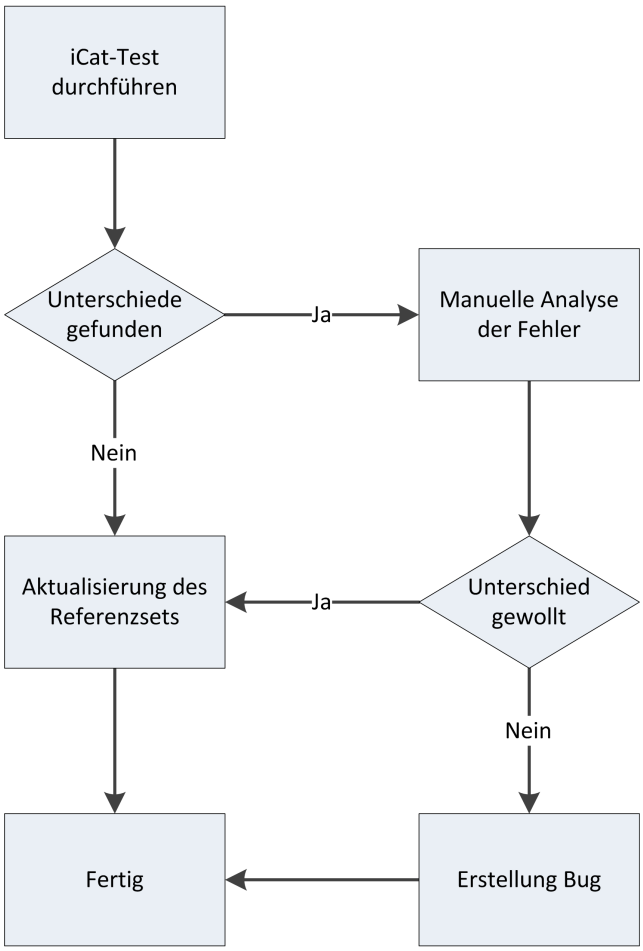

Abbildung 6.1: Geplanter Einsatz von iCAT im MAP Team [30][1].

Während der internen Präsentation des neuen iCAT entstand auch in anderen Teams großes Interesse an diesem. Sie sehen in iCAT das Potential, ihre bisherigen Testprozesse zu erweitern und in der Effizienz zu steigern, wodurch das Tool in Zukunft häufiger eingesetzt werden könnte.

[1] Diagramm aus einem internen Dokument der Daimler TSS GmbH

6.2 Ausblick

Die flexible und universelle Einsetzbarkeit, die strikte Trennung von „Aufnahme“ und „Vergleich“ in zwei voneinander unabhängige Basisfunktionen sowie die Umsetzung einer einheitlichen Datenschicht in der iCAT Anwendung selber bilden die Grundlage für den Einsatz von iCAT v2 in allen in Kapitel 3 beschriebenen Anwendungsszenarien.
Die Szenarien S.1, S.2 und S.3 können bereits vollständig abgedeckt werden. Auch die in Abschnitt 3.2 diskutierten Probleme des alten Ansatzes werden nahezu vollständig gelöst. Einzige Ausnahme ist die tatsächliche Integration von iCAT in eine CI-Umgebung. Durch die Kommandozeilenschnittstelle ist iCAT zwar auf diese Integration vorbereitet, jedoch war es im Rahmen dieser Arbeit nicht möglich, die Integration auch tatsächlich durchzuführen. Diese Integration sowie die Szenarien S.4 und S.5 bilden Ansatzpunkte für mögliche Erweiterungen von iCAT.

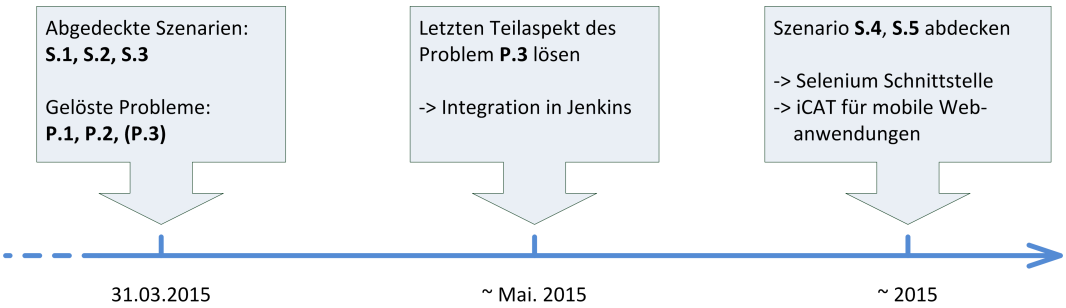

Abbildung 6.2: Stand zum Ende der Arbeit sowie geplante Weiterentwicklung von iCAT in der Firma (Daimler TSS GmbH).

Der aktuelle Stand der iCAT Anwendung und die vorläufig geplante Weiterentwicklung dieser sind in Abbildung 6.2 zusammengefasst. Zunächst soll iCAT in die CI-Umgebung Jenkins integriert und somit Problem P.2 vollständig gelöst werden. Eine denkbare Anwendung von iCAT wäre dabei, die Webanwendung zum Anlegen von Tests und Versionen zu verwenden und mit Jenkins nur das Erstellen und Vergleichen von Sets innerhalb eines solchen Tests zu starten. Wird die iCAT Anwendung auf einem Server bereitgestellt, so muss die CLI dafür beispielsweise um eine REST Schnittstelle erweitert werden. Dadurch ist es innerhalb eines Netzwerkes möglich, iCAT auch als Kommandozeilen-Anwendung remote zu steuern.
Weiterhin fest geplant ist eine iCAT Schnittstelle für Selenium (Szenario S.4). Durch diese soll es möglich sein, iCAT von außen zu steuern. Mit Selenium kann eine Webanwendung

bedient werden, wodurch Zustände erreichbar sind, die mit einer URI nicht abgedeckt werden können. Durch die iCAT Schnittstelle kann Selenium die Aufnahme eines Screenshots von entsprechenden Zuständen auslösen.
Außerdem soll es zukünftig möglich sein, iCAT auch für mobile Webanwendungen einzusetzen (Szenario S.5). Statt eines Desktop Browsers soll iCAT (bzw. das entsprechende Selenium Testskript) einen mobilen Browser bedienen.

Unabhängig von diesen neuen Einsatzgebieten kann iCAT auch in der Funktionsvielfalt weiter gesteigert und die Benutzbarkeit verbessert werden.
Die Möglichkeit, ein Screenshot-Set gegen ein Template zu vergleichen wäre hilfreich, um beispielsweise die korrekte Umsetzung einer *Corporate Design* Richtlinie zu überprüfen. Bei einem Template handelt es sich um eine einzelne Webseite, die nur wenige zu überprüfende Regionen beinhaltet. Manuell muss zuerst geprüft werden, ob beispielsweise ein Logo auf diesem Template korrekte dargestellt wird. Möchte man die korrekte Darstellung des Logos nun für alle Seiten eines Screenshot-Sets überprüfen, so muss dieses nur gegen dieses eine Template verglichen werden. In diesem Fall wird dabei für jede Seite nur die Logo-Region beachtet.
Die Logik der iCAT Anwendung ist für diesen Template Vergleich bereits vorbereitet. Lediglich die Webanwendung muss um entsprechende Funktionen zum Anlegen von Templates und Starten eines entsprechenden Vergleichs noch erweitert werden.

A iCAT Konfigurationsdatei

Listing A.1: Parameter für den Ausführungsmodus

```
# Different modes for execution are available. Please choose from
# the following:
# 1: Create a new screenshot set
# 2: Compare two screenshot sets
run.mode=2
```

Listing A.2: Parameter für das Vergleichen von zwei Screenshot-Sets

```
# the two sets that should be compared referenced by ids
refSet.id=1
testSet.id=2

# Different modes for comparison are available. Please choose
# from the following:
# 1: Compare two screenshot sets
# 2: Compare a screenshot set and a template page
compare.mode=1

# true if each region should be compared (pixel by pixel) if two
# screenshots have different dimensions
differentDimensions=true

# root output path for difference screenshots and reports
comp.output.path=C:/Dev/iCAT/Resources/Reports/
```

Listing A.3: Parameter für das Erstellen eines neuen Screenshot-Sets

```
# the test the new set should be created for referenced by id
# (must be empty if name is set)
test.id=
# the test the new set should be created for referenced by name
# (must be empty if id is set)
test.name=Mercedes
# the test version the new set should be created for
test.version=1.0

# environment referenced by id (must be empty if name and host are set)
env.id=
# environment referenced by name and host
# (both must be empty if id is set)
env.name=Daimler Env
env.host=https://www.mercedes-benz.de

# which browser Selenium should drive (possible values are firefox,
# chrome or ie)
browser=firefox

# sitemap and regionmap
sitemap.path=Resources/Sitemap/ba_sitemap_mercedes.html
regionmap.path=Resources/RegionMap/ba_regionmap_mercedes.html

# root output path for the screenshots
cap.output.path=C:/Dev/iCAT/Resources/ScreenshotSets/
```

Listing A.4: Allgemeine Aufnahme- und Vergleichsparameter

```
# The path to the ImageMagick command line tools
imagemagick.path=C:/Program Files/ImageMagick-6.8.9-Q16
# fuzz value for the ImageMagick compare tool
# (colors within this distance are considered equal)
imagemagick.fuzz=20%

# time to wait for pages to load (in ms)
pages.waitTime=2500

# amount of pixels that need to be identical for two images so that it
# is regarded as "partly passed" (percentage, 1 based)
match.pixelCount=0.8
```

Listing A.5: Proxydaten

```
# proxy settings
vm.proxy.host=
vm.proxy.port=
vm.proxy.username=KREIDES
vm.proxy.password=
```

Begriffserklärung

AJAX Asynchronous JavaScript and XML

API Application Programming Interface

CI Continuous Integration

CLI Command-Line Interface

CMS Content Management System

CSS Cascading Style Sheets

DOM Document Object Model

GUI Graphical User Interface

HTML Hypertext Markup Language

HTTP Hypertext Transfer Protocol

EJB Enterprise Java Beans

iCAT image-based Comparison Automation Tool

IDE Integrated Development Environment

JPA Java Persistence API

JSF Java Server Faces

MAP Marketing Platform

Begriffserklärung

POJO Plain Old Java Object

UI User Interface

UML Unified Modeling Language

URI Uniform Resource Identifier

XHTML Extensible Hyptertext Markup Language

Literaturverzeichnis

[1] AGILE ENGINE: *Screenster: Visual UI test automation for web applications*. http://www.creamtec.com/products/screenster/index.html. – Stand: 26.01.2015

[2] AMATO, Giuseppe ; GENNARO, Claudio ; RABITTI, Fausto ; SAVINO, Pasquale: Milos: A Multimedia Content Management System for Digital Library Applications. In: *Research and Advanced Technology for Digital Libraries*. Springer, 2004, S. 14–25

[3] BBC-NEWS: *wraith - Front-end regression testing tool*. https://github.com/BBC-News/wraith. – Stand: 22.01.2015

[4] BOEHM, B. W.: Guidelines for Verifying and Validating Software Requirements and Design Specifications. In: *Proceedings of Euro IFIP*, 1979, S. 711–719

[5] BOOTSTRAB: *Front-end Framework*. http://getbootstrap.com/. – 16.01.2015

[6] BUNDESVERBAND DIGITALE WIRTSCHAFT: *Faszination Mobile: Verbreitung, Nutzungsmuster und Trends*. http://www.bvdw.org/presseserver/studie_faszination_mobile/BVDW_Faszination_Mobile_2014.pdf. – Stand: 24.03.2015

[7] COHN, Mike: *Succeeding with Agile*. Addison-Wesley Professional, 2009. – Web ISBN-13: 978-0-321-66053-4

[8] COLLINS, Eliane ; LUCENA, Vicente: Iterative Software Testing Process for Scrum and Waterfall Projects with Open Source Testing Tools Experience. In: *On Testing Software and Systems: Short Papers* (2010), S. 115

[9] CRISPIN, Lisa ; GREGORY, Janet: *Agile Testing: A Practical Guide for Testers and Agile Teams*. Addison-Wesley Professional, 2008. – Web ISBN-13: 978-0-321-61694-4

[10] DUSTIN, Elfriede ; RASHKA, Jeff ; PAUL, John: *Automated Software Testing: Introduction, Management, and Performance*. Addison-Wesley Professional, 1999. – Print ISBN-13: 978-0-201-43287-9

[11] EVANS, Eric: *Domain-Driven Design: Tackling Complexity in the Heart of Software*. Addison-Wesley Professional, 2003. – Print ISBN-13: 978-0-321-12521-7

[12] FREEMARKER: *Java Template Engine*. `http://freemarker.org/`. – Stand: 13.03.2015

[13] GLASSFISH: *World's first Java EE 7 Application Server*. `https://glassfish.java.net/`. – Stand: 16.04.2015

[14] GOOGLE: *QualityBots*. `https://code.google.com/p/qualitybots/`. – Stand: 23.01.2015

[15] IMAGEMAGICK: *ImageMagick Compare Command-Line Tool*. `http://www.imagemagick.org/`. – Stand: 04.03.2015

[16] INTERNET SYSTEMS CONSORTIUM: *Domain Survey*. `https://www.isc.org/services/survey/`. – Stand: 24.03.2015

[17] JAVA.NET: *Maven Glassfish Plugin*. `https://maven-glassfish-plugin.java.net/`. – Stand: 16.04.2015

[18] JENKINS-CI: *An extensible open source continuous integration server*. `http://jenkins-ci.org/`. – 15.01.2015

[19] JPA PERFORMANCE BENCHMARK (JPAB): *EclipseLink with SQLite embedded*. `http://www.jpab.org/EclipseLink/SQLite/embedded.html`. – Stand: 09.02.2015

[20] MEMON, Atif ; BANERJEE, Ishan ; HASHMI, Nada ; NAGARAJAN, Adithya: DART: A Framework for Regression Testing „Nightly/daily Builds“ of GUI Applications. In: *Software Maintenance, 2003. ICSM 2003. Proceedings. International Conference on* IEEE, 2003, S. 410–419

[21] MEMON, Atif M. ; SOFFA, Mary L.: Regression Testing of GUIs. In: *SIGSOFT Softw. Eng. Notes* 28 (2003), Nr. 5, S. 118–127. – ISSN 0163–5948

[22] MYERS, Glenford J. ; SANDLER, Corey ; BADGETT, Tom: *The Art of Software Testing - Third Edition*. John Wiley & Sons, 2011. – Web ISBN: 1-118031-96-2

[23] ORACLE: *Java Persistence API*. `http://www.oracle.com/technetwork/java/javaee/tech/persistence-jsp-140049.html`. – Stand: 11.03.2015

[24] PARTSCH, H: *Softwaretechnik*. Universität Ulm, 2014. – Vorlesungsskript

[25] PIWIK: *Our latest improvement to QA: Screenshot Testing*. `https://piwik.org/blog/2013/10/our-latest-improvement-to-qa-screenshot-testing/`. – Stand: 23.01.2015

[26] PIWIK: *Screenshots UI tests*. `https://github.com/piwik/piwik/blob/master/tests/README.screenshots.md`. – Stand: 23.01.2015

[27] PRIMEFACES: *Ultimate JSF Framework*. `http://primefaces.org/index`. – Stand: 16.01.2015

[28] ROYCE, Winston W.: Managing the Development of Large Software Systems. In: *Proceedings of IEEE WESCON* Bd. 26 Los Angeles, 1970

[29] SCHWARZ, Fabian: *Test Automation in agile web-based Software Development*, Universität Ulm, Masterarbeit, 2014

[30] SEEMÜLLER, Holger: Verwendung von iCAT für Smoketests. – Internes Dokument der Daimler TSS GmbH

[31] SELENIUM: *Browser Automation*. `http://www.seleniumhq.org/`. – Stand: 13.02.2015

[32] SHAHAMIRI, Seyed R. ; KADIR, Wan Mohd Nasir W. ; MOHD-HASHIM, Siti Z.: A Comparative Study on Automated Software Test Oracle Methods. In: *Software Engineering Advances, 2009. ICSEA'09. Fourth International Conference on* IEEE, 2009, S. 140–145

[33] SPILLNER, Andreas ; LINZ, Tilo: *Basiswissen Softwaretest: Aus- und Weiterbildung zum Certified Tester; Foundation Level nach ISTQB-Standard*. 3., überarbeitete und aktualisierte Auflage. dpunkt.verlag, 2005. – ISBN: 3-89864-358-1

Name: Stefan Kreidel Matrikelnummer: 780468

Erklärung

Ich erkläre, dass ich die Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Ulm, den .....

Stefan Kreidel