Institut für Softwaretechnologie

Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Bachelorarbeit Nr. 168

# Studie über den Einfluss von geringfügigen Benutzerführungsanpassungen

**Study on the influence of slight improvments in user guidance**

Tobias Hirning

| | |
|---|---|
| **Studiengang:** | Softwaretechnik |
| **Prüfer/in:** | Prof. Dr. rer. nat. Stefan Wagner |
| **Betreuer/in:** | Dr. rer. nat. Rainer Schmidberger<br>Dipl.-Ing. Jan-Peter Ostberg |
| **Beginn am:** | 14. Juli 2014 |
| **Beendet am:** | 13. Januar 2015 |
| **CR-Nummer:** | D.2.5, D.2.9, H.5.2, K.6.3, K.6.4 |

## Kurzfassung

Diese Arbeit beschäftigt sich mit den Auswirkungen von geringfügigen Anpassungen der Benutzerführung eines Werkzeuges zum manuellen Test. Hierzu wurden Benutzer aus der Industrie auf verschiedene Arten befragt, die Ergebnisse kategorisiert und nach Priorisierung implementiert. Die Kategorisierung erfolgte sowohl nach eigenen Kriterien als auch nach den „Grundsätzen der Dialoggestaltung“ der DIN EN ISO 9241-110:2008-09. Die Bewertung des Einflusses der Änderungen erfolgt prognostisch, da keine Evaluation mit Probanden möglich war.

## Abstract

This thesis deals with the influences of slight improvements in user guidance of a tool for manual testing. For this industry-users have been interviewed using different methods. The results have been categorised and implemented after prioritisation. The categorisation was done following self-made criteria as well as the „Dialogue Principles“ of the DIN EN ISO 9241-110:2008-09. The evaluation of the influence of the changes was done prognostically, as no evaluation was possible with the probands.

# Inhaltsverzeichnis

# 1. Einleitung

## 1.1. Hintergrund dieser Arbeit

In diesem Abschnitt werden die Ausgangssituation und die Hintergründe diese Arbeit erläutert.

### 1.1.1. Ausgangssituation

Software ist voller Fehler. Manche Software hat mehr, manche weniger, aber keine ist fehlerfrei. Daher muss das Ziel aller am Entwicklungsprozess und Betrieb involvierten Parteien sein, Software mit möglichst wenig Fehlern zu schaffen.

Bei der Softwareentwicklung ist neben der eigentlichen Entwicklung ein großes Teilgebiet das Testen. Im Rahmen des Softwaretest wird dabei verifiziert, ob die Software die gestellten Anforderungen erfüllt und fehlerfrei arbeitet. Der Test beträgt dabei nach Schätzungen von Riedemann zwischen 15 % und 50 % am Gesamtentwicklungsaufwand.[1] In der Theorie und Praxis werden verschiedene Arten von Softwaretests beschrieben und verwendet, die sich in ihren organisatorischen und technischen Eigenschaften stark unterscheiden.

### 1.1.2. Problemstellung

Bei jeder Art des Tests darf der menschliche Anteil am Test nicht unterschätzt werden. Zwar werden viele Testarten automatisiert mittels Software durchgeführt, doch fast alle Testarten müssen von Menschen vorbereitet werden.[2] Hierbei strukturiert vorzugehen erfordert spezielle Techniken, theoretisches Wissen und geeignete Werkzeuge. In dieser Arbeit beschäftigt sich der Autor mit einem speziellen Werkzeug – Testsuite-Management (TSM)[3] – und dessen Einbindung in den Testprozess.

[1] [Rie97], S. 40; dort zitiert nach [Boe77].

[2] In der Praxis dürfte das sogenannte „Fuzzing“ die einzige Testart sein, die in allen Bereichen automatisiert abläuft. Hierbei wird der Prüfling mittels eines Werkzeuges mit zufälligen Daten „gefüttert“, um Fehleingaben zu simulieren.

[3] `http://tsmtest.sourceforge.net/`, abgerufen am 07.01.2015.

## 1.2. Aufgabenbeschreibung

Die folgenden Abschnitte 1.2.1 *Hintergrund* und 1.2.2 *Themenstellung* wurden wörtlich aus der Aufgabenbeschreibung der Betreuer übernommen. Lediglich kleine Rechtschreibkorrekturen wurden vorgenommen.

### 1.2.1. Hintergrund

Das Open-Source-Testwerkzeug TSM wurde 2012 im Rahmen eines Studienprojekts des Studiengangs Softwaretechnik an der Universität Stuttgart entwickelt. TSM unterstützt den Tester beim Entwurf, der Ausführung und dem Auswerten der sogenannten manuellen Tests. In der ursprünglichen Projektplanung des TSM-Studienprojekts war eine Pilotphase vorgesehen, in der TSM in der industriellen Praxis eingesetzt werden soll und Erkenntnisse dieses Einsatzes wiederum in der abschließenden Implementierungsphase berücksichtigt werden sollten. Aus Termingründen fand dieser Piloteinsatz allerdings nicht statt.

### 1.2.2. Themenstellung

Aufgabe der Bachelorarbeit ist, es diese Pilotphase mit ausgewählten interessierten Anwendern nachzuholen. Im ersten Schritt sollen die Pilot-Kunden mit der aktuellen Version von TSM praktische Erfahrung sammeln. Eine Benutzerunterstützung wird hierzu vom Bacheloranten bereitgestellt. Anschließend sollen die Defizite der Software durch Interviews oder Fragebögen festgestellt werden. Diese Defizite werden dann katalogisiert, priorisiert und in implementierbare Aufgaben aufgeteilt. Hierbei sollen auch Metriken, z. B. User Satisfaction oder Time to Task Completion, als quantitatives Bewertungsmittel in Betracht gezogen werden. Der Schwerpunkt soll dabei auf der Verbesserung der Benutzerführung liegen und weniger auf der Implementierung neuer Funktionen, allerdings sollen auch andere Dokumente wie z. B. ein Online-Tutorial, einfache Installation oder andere akzeptanzfördernde Aspekte mit einbezogen werden. Ein Teil dieser Verbesserungen soll dann in enger Abstimmung mit den Pilot-Kunden umgesetzt werden. Der Einfluss auf die Zufriedenheit der Anwender durch die einzelnen Verbesserungen soll dabei wieder mittels Interviews oder Fragebögen erhoben werden.

## 1.3. Danksagungen

Zuvörderst möchte ich mich bei meinen beiden Betreuern Dipl-Ing. Jan-Peter Ostberg und Dr. Rainer Schmidberger für Ihre Unterstützung und stets konstruktive Kritik bedanken. Sie haben mir ermöglicht ein Softwareprojekt weiterzuverfolgen, in welches bereits zuvor im Studium jede Menge Zeit und Arbeit geflossen ist, das aber in einem unbefriedigenden Zustand hinterlassen wurde.

Mein zweiter Dank gilt der STL GmbH, die mir die Möglichkeit gab, ein Free/Libre-Open-Source-Software-Projekt zu verbessern, mir dabei freie Hand lies und Unterstützung bot.

Insbesondere wurde mir ermöglicht, echte Anwender aus der Industriepraxis als Probanden zu gewinnen und einen Einblick in deren Testprozesse zu erhalten.

Abschließend gilt mein Dank den unzähligen Personen, die freie und Open-Source-Software geschaffen haben. Ihre Arbeit hat nicht nur TSM ermöglicht, sondern war auch ausschließliche technische Grundlage für die Erstellung dieser Arbeit.

## Gliederung

Die Arbeit ist in folgender Weise gegliedert:

**Kapitel 2 – Grundlagen und Theorie:** Hier werden werden die Grundlagen dieser Arbeit beschrieben.

**Kapitel 3 – Datenerhebung und Implementierung:** Hier wird die Befragung und Umsetzung beschrieben.

**Kapitel 4 – Zusammenfassung und Ausblick** fasst die Ergebnisse der Arbeit zusammen und gibt einen Ausblick in die Zukunft.

**Anhang A – Quelltextmetriken** gibt einen kurzen Überblick über die erhobenen Metriken.

**Anhang B – LaTeX-Vorlage** enthält die Urheberrechtsvermerke und die Lizenz der verwendeten LaTeX-Vorlage.

# 2. Grundlagen und Theorie

## 2.1. Terminologie rund um das Thema „Testen“

In diesem Abschnitt wird die in dieser Arbeit verwendete Terminologie zum Thema „Testen“ näher erläutert.

Beim Thema Softwaretest unterscheiden der Autor hier zwei Bereiche, über die ein kurzer Überblick gegeben werden soll, damit die Einbettung des Werkzeugs TSM in den Bereich des Softwaretests ersichtlich ist.

### 2.1.1. Prüfling

Ein „Prüfling“ ist die Hard-, Software oder das System, welches getestet wird.

### 2.1.2. Automatisches Testen

Der automatisierte Test erfolgt unmittelbar durch ein Softwarewerkzeug. Er erfordert zumeist nur zu Beginn die Beteiligung eines Menschen am Testvorgang. Hierzu definiert dieser vorab, was getestet werden soll und wie die Soll-Werte aussehen. Jede Werkzeugart erfordert eine andere Herangehensweise und deckt unterschiedliche Systemebenen des Prüflings ab.

#### 2.1.2.1. Modul- oder Komponententest

Auf unterster Ebene steht typischerweise der Modul- oder Komponententest. Module oder Komponenten werden als kleinste logische Untergliederung von Software betrachtet, die eine funktional abgeschlossenen Einheit bilden. Sie verfügen im Allgemeinen über klar definierte Schnittstellen und eine Komplexität, die mit vertretbarem Aufwand gehandhabt werden kann. Für viele Programmiersprachen stehen Softwarepakete bereit, die Unterstützung bei der Erstellung und Durchführung von Modultests bieten.[4]

[4] Für das hier einschlägige Java sei beispielsweise JUnit genannt. Siehe: `http://junit.org/`, abgerufen am 14.08.2014.

#### 2.1.2.2. Testroboter

Unter einem Testroboter[5] ist ein Programm zur Testautomatisierung zu verstehen, das die Interaktion eines menschlichen Testers mit dem Prüfling aufzeichnet und anschließend beliebig oft automatisiert wiederholen kann.

### 2.1.3. Manuelles Testen

Unter manuellem Testen ist das Testen eines Prüflings durch direkte Interaktion eines Menschen mit ihm zusammengefasst. Hierbei bedient der Tester den Prüfling wie ein späterer Benutzer unter Verwendung der Benutzeroberfläche.

### 2.1.4. Systemtest

Der Systemtest testet das vollständige System (oder mindestens einen großen Teil davon) und simuliert dabei das spätere lauffähige System. Damit prüft er das Zusammenspiel der Module und kann Fehler finden, die erst durch Kombination der Module entstehen und somit bei anderen Testarten nicht gefunden werden können. Er kann sowohl automatisiert als auch manuell erfolgen. Der Autor beschränkt sich hier auf die manuelle Variante.

### 2.1.5. Testerfolg

In TSM gilt ein Testfall und damit ein einzelner Test dann als „erfolgreich“, wenn er keine Fehler aufgezeigt hat. Die Semantik ist somit der in der Literatur verwendeten genau entgegengesetzt. Dieser Unterschied rührt daher, dass während der Entwicklung von TSM die Auffassung des internen Kunden – dieser wurde durch einen Betreuer dargestellt – und des Industriepartners dahingehend war, dass eine Ausführung erfolgreich ist, wenn sie keine Fehler aufgezeigt hat. Diese Auffassung dürfte sich auch mit der Verkehrsauffassung des Begriffs des erfolgreichen Tests decken, auch wenn die in der Literatur[6] vertretene Semantik formal korrekt ist.

## 2.2. Was ist TSM?

Bei der Software Testsuite-Management (TSM) handelt es sich um ein Open-Source-Testwerkzeug, das Unterstützung bei manuellen Softwaretests bietet. TSM wurde im Rahmen eines Studienprojekts des

[5] Auch als „Capture&Replay“-Werkzeug oder „Capture&Playback“-Werkzeug bezeichnet.

[6] Beispielsweise definiert [LL10], S. 481 einen Test als „erfolgreich“, wenn er einen Fehler aufzeigt. Findet der Test keinen Fehler so gilt dieser als „erfolglos“. Diese Definition folgt der Grundidee, dass es die Aufgabe eines Tests ist, Fehler aufzuzeigen.

Bachelor-Studiengangs Softwaretechnik an der Universität Stuttgart entwickelt.[7] Das Studienprojekt wurde durch einen Industriepartner[8] begleitet, der Einblick in seinen Testprozess gewährte. Bei diesem wurden zu diesem Zeitpunkt die Verwaltung und Durchführung von Tests mittels Textverarbeitungsprogrammen[9] und Netzwerklaufwerken[10] realisiert.

### 2.2.1. Technischer Unterbau

TSM wurde in der Programmiersprache Java in der Version 6 entwickelt. Als externe Bibliotheken kamen JDOM[11] (XML-Bibliothek) und iText[12] (PDF-Bibliothek) zum Einsatz.

TSM verwendet das Eclipse-Framework[13]. Bei Eclipse handelt es um eine integrierte Entwicklungsumgebung für Java, die ursprünglich von IBM entwickelt wurde und später als freie Software veröffentlicht wurde. Heutzutage steht Eclipse unter der Eclipse Public License in der Version 1.0[14].

Die Daten (Testfälle, Testfallprotokolle und Bilder) werden als XML-Dateien in der lokalen Installation vorgehalten. Für jeden Testfall und für jedes Protokoll existiert eine Datei. Projekte und Pakete werden durch Ordner im Dateisystem repräsentiert. Die Hierarchie im Dateisystem entspricht der Hierarchie in TSM.

### 2.2.2. Logische Strukturen in TSM

TSM arbeitet mit „Projekten“, die als oberstes Ordnungselement für eine Ansammlung von Daten fungieren. Ein Projekt kann beliebig viele Pakete beinhalten.

Unterhalb von Projekten gibt es „Pakete“ als logische Struktur. Mit ihnen lassen sich einzelne Daten strukturieren und bündeln. Ein Paket kann beliebig viele Testfälle und Testfallprotokolle beinhalten.

Der Begriff „Testfall“ umfasst in TSM eine Liste von Anweisungen an den Tester, die sogenannten „Testschritte“, eine Kurzbeschreibung, Vorbedingungen, eine geschätzte Dauer, die Zuordnung zu

[7] Das Studienprojekt wurde von acht Personen von April 2012 bis April 2013 unter Leitung der Abteilung Software Engineering des Instituts für Softwaretechnologie der Universität Stuttgart durchgeführt. Siehe: `http://www.informatik.uni-stuttgart.de/studierende/studiengaenge/softwaretechnik/studienprojekte-a.html`, abgerufen am 14.08.2014.

[8] Ein mittelständisches IT-Unternehmen aus Stuttgart mit mehreren Standorten in Deutschland und anderen Ländern.

[9] Im Wesentlichen Microsoft Word mit den dazugehörigen Kompatibilitätsproblemen zwischen verschiedenen Programmversionen.

[10] Einfache Netzwerklaufwerke ohne Versionsverwaltung.

[11] Siehe `http://www.jdom.org/`, abgerufen am 07.11.2014.

[12] In der Version 4.2, die eine Verbindung mit EPL-lizenziertem Quelltext zulässt. – Siehe `https://github.com/ymasory/iText-4.2.0`, abgerufen am 07.11.2014.

[13] Siehe `https://www.eclipse.org/`, abgerufen am 07.11.2014.

[14] Siehe `https://www.eclipse.org/org/documents/epl-v10.php`, abgerufen am 15.12.2014.

einem Tester, einen Autor und eine Priorität. Zu einem Testfall können beliebig viele Testschritte und Testfallprotokolle gehören.

Ein „Testschritt“ besteht aus einer Aktion, die der Tester ausführen soll und einem erwarteten Ergebnis, dass der Prüfling liefern soll. Die Testschritte sind sequentiell geordnet und nummeriert.

Als „Testfallprotokoll“ wird die Aufzeichnung einer Testfallausführung bezeichnet. In ihm sind zusätzlich zu den Daten des Testfalls für jeden Testschritt ein tatsächliches Ergebnis und ein Status[15] enthalten. Weiterhin enthält es ein endgültiges Ergebnis, einen Status über alle Testschritte und eine tatsächliche Dauer.

### 2.2.3. Arbeitsweise in TSM

Beim Erstellen eines Testfalls gibt der Autor dem Tester[16] Anweisungen, wie dieser mit dem Prüfling zu interagieren hat.

Bei der Testfallausführung interagiert der Tester nach den Anweisungen im Testfall mit dem Prüfling und dokumentiert das tatsächliche Ergebnis. Hierzu bedient er sich einem Ampelsystem, um für jeden Testschritt einen Status zu vergeben.[17] Zusätzlich hat er die Möglichkeit, ein Ergebnis als Text zu dokumentieren.

Die Übersicht zeigt in einer Tabelle für jeden Testfall in jeder ausgeführten Revision den Status des Testfallprotokolls in den Farben, die den Ampelfarben der Testfallausführung entsprechen.

Testfälle und Testfallprotokolle lassen sich in eine oder mehrere PDF-Dateien exportieren. Diese enthalten alle Daten in einem druckfähigen Layout.

[15] „bestanden“, „bestanden mit Anmerkungen“ und „fehlgeschlagen“
[16] Dieser kann mit dem Autor identisch sein.
[17] Hierbei kommt die Kodierung „grün“ – „bestanden“, „gelb“ – „bestanden mit Anmerkungen“ und „rot“ – „fehlgeschlagen“ zur Anwendung.

# 3. Datenerhebung und Implementierung

## 3.1. Probanden und ihre Vorkenntnisse

Die Probanden[18] sind Tester und Entwickler verschiedener Unternehmen aus verschiedenen Branchen. Zu Befragungsbeginn hatte die Mehrheit von Ihnen eine Einführung in die Funktionsweise von TSM bei einem zehnminütigen Vortrag erhalten. Nur zwei bis drei Probanden hatten zuvor bereits mit dem Testwerkzeug Justus[19] gearbeitet. Generell ist festzustellen, dass die Probanden stark unterschiedliche Hintergründe haben. Eine direkte Vergleichbarkeit der Probanden untereinander ist daher nicht gegeben. Andererseits sorgt diese Vielfältigkeit aber auch für viele unterschiedliche Sichtweisen durch verschiedene Testprozesse.

## 3.2. Befragungen und andere Erhebungen

In diesem Abschnitt wird ein Überblick über die verwendeten Befragungs- und Erhebungsmethoden gegeben und auf ihre Probleme eingegangen.

- Online-Fragebogen (Rogator)
- Telefonate
- Diskussion beim Arbeitskreis Software-Qualität und Fortbildung e. V. (ASQF)
- E-Mails von Anwendern

### 3.2.1. Fragebogen

Bei einem kleinen Teil der Probanden wurden Erfahrungen und Nutzungsverhalten mittels eines Online-Fragebogens[20] erfasst. Von einer weiteren Erfassung dieser Art wurde abgesehen, da es zu besorgen war, dass durch die vorgegebenen Fragen wichtige Aspekte nicht beachtet werden würden. Dabei war weniger die Art oder der Inhalt der Fragen ausschlaggebend, als dass sie vielmehr nur Defizite vorhandener Funktionen abfragten und eine hohe Wiederholungsrate hatten. Aufgrund der

[18] Zur Wahrung der Anonymität wird unabhängig vom Geschlecht die männliche Form verwendet.
[19] Justus ist ein älteres Werkzeug zur Testfallverwaltung und Testdurchführungen, das als Projektarbeit an der Universität Stuttgart entstanden ist. Siehe `http://justus.tigris.org/`, abgerufen am 16.12.2014.
[20] Rogator – Siehe `http://www.rogator.de/`, abgerufen am 16.12.2014.

geringen Probandenanzahl werden die Ergebnisse des Fragebogens nicht gesondert ausgewiesen, um eine Deanonymisierung der Probanden zu verhindern.

### 3.2.2. Vorträge mit Diskussionen

TSM wurde in drei Vorträgen einem größeren Kreis von möglichen Probanden vorgestellt.

#### 3.2.2.1. Anwendertreffen

Der erste Vortrag wurde im Mai 2014 beim Anwendertreffen der STL GmbH gehalten. Dem Vortrag schloss sich eine halbstündige Diskussion mit den ungefähr 30 Teilnehmern an. Diese waren Entwickler und Tester von Firmen aus der Versicherungs- und Finanzdienstleistungsbranche.

#### 3.2.2.2. Arbeitskreis Software-Qualität und Fortbildung e. V. (ASQF)

Der zweite Vortrag fand im Rahmen einer Veranstaltung des Arbeitskreises Software-Qualität und Fortbildung e. V. im September 2014 statt.[21] Auch diesem Vortrag schloss sich eine einstündige Diskussion mit den acht Teilnehmern an. Diese kamen aus unterschiedlichen Industriebereichen und waren Entwickler, die mit Testaufgaben betraut sind. Bei anschließenden Einzelgesprächen wurden weitere Probleme deutlich. So berichteten einzelne Teilnehmer, dass ihre Tester meistens keinen gleichwertigen technischen Hintergrund wie Entwickler haben und daher schon mit – für Entwickler – einfache Aufgaben Probleme haben.[22]

#### 3.2.2.3. Stuttgarter Firma

Ein dritter Vortrag mit acht Teilnehmern fand am 22. Oktober 2014 bei einer mittelständischen Stuttgarter Firma mit über 370 Mitarbeitern statt. Im Anschluss fand eine einstündige Diskussion mit den Teilnehmern statt. Diese hatten alle technische Hintergründe als Entwickler. Neben allgemeingültigen Vorschlägen kamen hier auch sehr firmenspezifische Wünsche auf. Die Auflistung der Befragungsergebnisse enthält jeweils eine Kennzeichnung, wenn es sich um solche Wünsche handelt.

[21] Rainer Schmidberger: *Manuelles Testen mit dem Werkzeug TSM.* Vortrag vom 18. September 2014. – Siehe `http://bw-test.org/index.php?title=18._September_2014&oldid=1324`, abgerufen am 25.09.2014.

[22] Als Beispiel wurde der Umgang mit Versionskontrollsystemen oder einer Kommandozeile genannt.

### 3.2.3. Telefonbefragungen

Zusätzlich wurde ein einzelner Proband noch telefonisch befragt, um einen Eindruck von der Verwendung von TSM zu erhalten. Dieser Proband hatte schon intensiv mit TSM gearbeitet und zuvor schon mit Justus. Bei ihm war TSM schon mehrere Wochen im produktiven Einsatz, bevor er das erste Mal Kontakt mit dem Autor hatte. Er schilderte eine problemlose Einarbeitung, nur zu einigen einzelnen Punkten hätte er das Handbuch konsultieren müssen. Das Testen habe Spaß gemacht.

### 3.2.4. Analyse echter Arbeitsdaten

Ein Proband stellte eine Kopie seiner Arbeitsdaten zur Analyse bereit. Die Instanz war auf einem Netzlaufwerk installiert und wurde von mehreren Testern benutzt.

In den Daten waren sieben Projekte mit insgesamt 41 Paketen. Insgesamt waren 84 Testfälle mit 81 Testfallprotokollen enthalten. Der Großteil der Testfälle war nur einmal ausgeführt worden.

In keinem der Testfälle wurde Textformatierung verwendet; endgültige Resultate wurden nur in sehr wenigen Fällen verwendet. Eingefügte Bilder waren in einem großen Teil der Testfälle vorhanden. Teilweise wurden sie als Ersatz für beschreibenden Text verwendet – insbesondere für das erwartete Ergebnis.

## 3.3. Probleme der Evaluierung

In diesem Abschnitt wird auf die Probleme der Evaluation der vorgenommenen Änderungen eingegangen.

### 3.3.1. Wenige „Wiederholungsprobanden“

Während bei der ersten Erfassung ein großer und heterogener Probandenkreis einbezogen werden konnte, fanden sich für eine zweite Datenerhebung nur wenige Probanden. Dies ist unter anderem darin begründet, dass ein Großteil der Probanden der ersten Erfassung Teilnehmer von Vorträgen waren und anschließend in den meisten Fällen TSM nicht mehr eingesetzt haben.

### 3.3.2. Ausfall der Rückmeldungen

Die Probanden der ersten Erfassung, die TSM weiterhin benutzten, waren zusätzlich zu den oben beschriebenen Telefonbefragungen bereit. Bedauerlicherweise gab es von ihnen keine Rückmeldungen zu den Anpassungen. Dies war mutmaßlich durch berufliche Inanspruchnahme bedingt.

### 3.3.3. Prognose der Auswirkungen der Änderungen

Mangels Rückmeldungen können die Auswirkungen der umgesetzten Änderungen auf die Benutzerführung nur mittels Erfahrungen und unter Einbeziehung der DIN-Norm zu den Grundsätzen der Dialoggestaltung (siehe hierzu Abschnitt 3.4.6 *„Grundsätze der Dialoggestaltung“*) prognostiziert werden.

## 3.4. Ergebnisse der Befragungen

Die Änderungsvorschläge (aufgelistet in 3.5 *Änderungsvorschläge und deren Umsetzung*) lassen sich in mehrere Kategorien aufteilen. Im Folgenden werden diese näher erläutert. Die einzelnen Änderungsvorschläge sind in zweifacher Weise gekennzeichnet. So sind sie zum einen in die hier gefundenen Kategorien eingeteilt (im Folgenden „TSM-Kategorien“), als auch in die Kategorien der DIN EN ISO 9241-110 (siehe hierzu 3.4.6 *„Grundsätze der Dialoggestaltung“*, im folgenden „ISO-Kategorien“). Hierbei erfolgte die Einteilung in die TSM-Kategorien aus „TSM-Sicht“, d. h. ein Änderungsvorschlag, der mit „Aufgabenangemessenheit“ kategorisiert wurde, ist aus TSM-Sicht eine „Erweiterung“, da hierfür das Programm erweitert werden muss.

### 3.4.1. Fehlende Eclipse-Vorkenntnisse (Eclipse-spezifisch)

Ein kleinerer Anteil der Änderungsvorschläge lässt sich auf die fehlenden Eclipse-Vorkenntnisse einiger Probanden zurückführen. So arbeitet Eclipse immer Workspace-bezogen, d. h. Daten müssen erst mittels Assistent ex- oder importiert werden, bevor sie weiterverarbeitet werden können. Damit eng verbunden ist eine Baumansicht, die sich „Navigator“ nennt. Zwar sind die meisten Funktionen auch über Menüs verfügbar, doch für einige ist die Interaktion mit dem Navigator erforderlich.

### 3.4.2. Nicht-intuitive Benutzerführung (nicht-intuitiv)

Ein Teil der Änderungsvorschläge bezog sich auf schlecht gewählte Texte auf der Benutzeroberfläche oder fehlende Schaltflächen.

### 3.4.3. Vereinfachen der Oberfläche (Vereinfachung)

Mehrfach wurde eine vereinfachte oder vereinfachbare Oberfläche für Tester vorgeschlagen, da diese oft nicht die nötige Zeit haben, um sich detailliert in die Oberfläche einarbeiten zu können oder keinen technischen Hintergrund wie ein Entwickler haben. Die Tester haben sehr unterschiedliche Hauptbetätigungsfelder (Kundendienst, Entwicklung, Vertrieb, …) und sollen daher eine möglichst einfache Benutzeroberfläche erhalten. Hierzu soll beim Programmstart nur eine Oberfläche angezeigt bekommen, die stark vereinfacht ist. Tester mit mehr Erfahrung sollen die „erweiterte“ Oberfläche in den Einstellungen reaktivieren können.

### 3.4.4. Erweiterte Funktionalität (Erweiterung)

Erweiterungsvorschläge in Bezug auf die Funktionalität bezogen sich auf die Anbindung an die Infrastruktur und auf weitere Berichtsmöglichkeiten für die nächsthöhere Organisationsebene. So war es ein großer Wunsch, die Versionierung mittels Versionsverwaltungssystem halb- oder vollautomatisiert vornehmen zu lassen. Ebenfalls wichtig war es, dass Daten mit Kollegen oder anderen Standorten ausgetauscht werden können.

### 3.4.5. Datenhandhabbarkeit

Insbesondere von Führungskräften wurde eine bessere Handhabbarkeit der Daten gewünscht. Hierunter ist neben dem Umgang mit den Testdaten (Testfälle und Testfallprotokolle) auch deren Speicherung im Hinblick auf verteilte Zusammenarbeit und Historie wichtig. So soll möglichst zu jedem Zeitpunkt eine Übersicht über den aktuellen Teststand verfügbar sein, ohne dass die Tester hierzu mitwirken müssen. Die Daten sollen stunden- bis minutenaktuell sein. Des Öfteren wurde der Wunsch nach einer Datenbankanbindung genannt, da die Verwaltung einzelner XML-Dateien in der lokalen Installation als schwierig beschrieben wurde.

### 3.4.6. „Grundsätze der Dialoggestaltung“

Zusätzlich zu den oben definierten Kategorien werden die einzelnen Änderungsvorschläge nach Kriterien der Norm DIN EN ISO 9241-110:2008-09 kategorisiert. Dies dient der Ergänzung und als Möglichkeit zur Vergleichbarkeit mit anderen Änderungsvorschlägen oder Projekten.

Die Norm DIN EN ISO 9241 trägt den Titel „Ergonomie der Mensch-System-Interaktion“ und beschäftigt sich mit den Anforderungen an Arbeitsplatz, Hardware und Software um die Nutzer vor Gesundheitsschäden zu bewahren und die Arbeit angenehm zu gestalten.

Teil 110 der Norm trägt den Titel „Grundsätze der Dialoggestaltung“ und geht genauer auf die Ausgestaltung von Dialogen ein. „Dialog“ wird dabei definiert als „Interaktion zwischen einem Benutzer und einem interaktiven System in Form einer Folge von Handlungen des Benutzers (Eingaben) und Antworten des interaktiven Systems (Ausgaben), um ein Ziel zu erreichen“[23].

Die folgenden Grundsätze werden dabei empfohlen.

[23] [DIN08], Nr. 3.2.

#### 3.4.6.1. Aufgabenangemessenheit

„Ein interaktives System ist aufgabenangemessen, wenn es den Benutzer unterstützt, seine Arbeitsaufgabe zu erledigen, d. h., wenn Funktionalität und Dialog auf den charakteristischen Eigenschaften der Arbeitsaufgabe basieren, anstatt auf der zur Aufgabenerledigung eingesetzten Technologie.“[24]

#### 3.4.6.2. Selbstbeschreibungsfähigkeit

„Ein Dialog ist in dem Maße selbstbeschreibungsfähig, in dem für den Benutzer zu jeder Zeit offensichtlich ist, in welchem Dialog, an welcher Stelle im Dialog er sich befindet, welche Handlungen unternommen werden können und wie diese ausgeführt werden können.“[25]

#### 3.4.6.3. Erwartungskonformität

„Ein Dialog ist erwartungskonform, wenn er den aus dem Nutzungskontext heraus vorhersehbaren Benutzerbelangen sowie allgemein anerkannten Konventionen entspricht.“[26]

#### 3.4.6.4. Lernförderlichkeit

„Ein Dialog ist lernförderlich, wenn er den Benutzer beim Erlernen der Nutzung des interaktiven Systems unterstützt und anleitet.“[27]

#### 3.4.6.5. Steuerbarkeit

„Ein Dialog ist steuerbar, wenn der Benutzer in der Lage ist, den Dialogablauf zu starten sowie seine Richtung und Geschwindigkeit zu beeinflussen, bis das Ziel erreicht ist.“[28]

#### 3.4.6.6. Fehlertoleranz

„Ein Dialog ist fehlertolerant, wenn das beabsichtigte Arbeitsergebnis trotz erkennbar fehlerhafter Eingaben entweder mit keinem oder mit minimalem Korrekturaufwand seitens des Benutzers erreicht werden kann. Fehlertoleranz wird mit den Mitteln erreicht: Fehlererkennung und -vermeidung (Schadensbegrenzung); Fehlerkorrektur oder Fehlermanagement, um mit Fehlern umzugehen, die sich ereignen.“[29]

[24] [DIN08], Nr. 4.3.
[25] [DIN08], Nr. 4.4.
[26] [DIN08], Nr. 4.5.
[27] [DIN08], Nr. 4.6.
[28] [DIN08], Nr. 4.7.
[29] [DIN08], Nr. 4.8.

#### 3.4.6.7. Individualisierbarkeit

„Ein Dialog ist individualisierbar, wenn Benutzer die Mensch-System-Interaktion und die Darstellung von Informationen ändern können, um diese an ihre individuellen Fähigkeiten und Bedürfnisse anzupassen.“[30]

## 3.5. Änderungsvorschläge und deren Umsetzung

In diesem Abschnitt werden die Änderungsvorschläge aus den Befragungen präsentiert und deren Umsetzung erläutert. Die Auswirkungen der Umsetzung der Änderungsvorschläge wird prognostisch evaluiert. Ebenfalls wird erläutert, warum einige Vorschläge nicht umgesetzt wurden.

### 3.5.1. Selektiver Export nach Status und Priorität (umgesetzt)

**TSM-Kategorien:** *Erweiterung*
**ISO-Kategorien:** *Steuerbarkeit*
Der Export von Testfallprotokollen in PDF-Dateien soll nach Status und Priorität gefiltert werden können.

**Umsetzung:** Der Export von Testfallprotokollen wurde angepasst, so dass eine Filterung nach Status und Prioritäten möglich ist.

**Prognose:** Die Funktionalität dürfte für viele Benutzer einen Vorteil bringen, da die so erzeugten PDF-Dateien weniger umfangreich sind und eine Durchsicht einfacher machen und bei der Archivierung auf Papier Platz sparen. Nachteile sind wenn überhaupt nur in sehr geringem Umfang in Form von einem mehr an Optionen auf der Benutzeroberfläche zu erwarten. Hierzu muss der Benutzer sich allerdings aktiv für die Erweiterung der Optionen entscheiden.

### 3.5.2. Export und Import von Testfällen (teilweise umgesetzt)

**TSM-Kategorien:** *Erweiterung, Eclipse-spezifisch*
**ISO-Kategorien:** *Erwartungskonformität*
Testfälle sollen aus einem Projekt exportiert und in ein anderes Projekt importiert werden können.

**Umsetzung:** Das Exportieren und Importieren von Testfällen konnte umgesetzt werden, da das Eclipse-Framework diese Funktionalität bereits mitlieferte. Der Ex- und Import von Bildern, die mit Testfällen verbunden sind, ist dabei allerdings nicht möglich. Dessen technische Umsetzung war im Rahmen dieser Arbeit nicht möglich. Für die Endanwender wurde untersucht, welche Probleme dabei in der Praxis auftreten können und die Funktion mit einer separaten Benutzeranleitung dokumentiert.

[30] [DIN08], Nr. 4.9.

**Prognose:** Die Funktionalität wird nur für diejenigen Benutzer von Vorteil sein, die keine Bilder verwenden oder auf diese beim Ex- und Importieren verzichten können. Mit dem Ex- und Import von Bildern würde die Funktionalität alle Benutzer unterstützen.

### 3.5.3. Überblick über die Entwicklung eines einzelnen Testfalls (umgesetzt)

**TSM-Kategorien:** *Erweiterung, nicht-intuitiv*
**ISO-Kategorien:** *Steuerbarkeit*
Beim PDF-Export soll es möglich sein, für jeden Testfall zu sehen, wie er sich im Laufe der Zeit entwickelt hat. Dazu sollen alle oder eine bestimmte Anzahl von alten Testfallprotokollen nacheinander sortiert exportiert werden.

**Umsetzung:** Der Export von Testfallprotokollen einzelner oder mehrere Testfälle in allen Revisionen des Testfalls war bereits vorhanden, da der Export auf Testfallprotokolle beschränkt werden konnte. Zusätzlich wurde die Möglichkeit implementiert, mehrere Revisionen oder einen Revisionsbereich angeben zu können.

**Prognose:** Die Funktionalität dürfte vor allem für Führungskräfte oder für Benutzer die an solche berichten einen Vorteil bringen, da es damit möglich ist, genauer zu steuern, welche Daten exportiert werden sollen.

### 3.5.4. Dokumentation früherer Versionsstände eines Testfalls (umgesetzt)

**TSM-Kategorien:** *nicht-intuitiv, Datenhandhabbarkeit*
**ISO-Kategorien:** *Erwartungskonformität*
Es wird gewünscht, dass die vorherigen Versionsstände eines Testfalls aufbewahrt werden sollen.

**Umsetzung:** Die Speicherung früherer Versionsstände eines Testfalls war bereits in Form der Testfallprotokolle vorhanden. Diese beinhalten nicht nur die Ergebnisse der Testdurchführung sondern auch den Zustand des Testfalls zum Zeitpunkt der Testdurchführung. Um dies deutlicher zu machen, wird jedem Protokoll im TSM-Navigator die Revisionsnummer nachgestellt. Da so gespeicherte alte Zustände eines Testfalls nicht bearbeitet oder ausgeführt werden können, bietet sich die Benutzung eines Versionsverwaltungssystems an.

**Prognose:** Die Änderung dürfte dazu führen, dass deutlicher wahrgenommen wird, wo alte Versionsstände eines Testfalls zu finden sind. Da diese allerdings in Form eines Testfallprotokolls abgelegt sind, hilft die Änderung nicht, wenn tatsächlich alte Versionsstände eines Testfalls ausgeführt werden sollen.

### 3.5.5. Feld mit Versionsnummer oder -text zusätzlich zur Revisionsnummer (umgesetzt)

**TSM-Kategorien:** *Erweiterung*
**ISO-Kategorien:** *Erwartungskonformität*

Zusätzlich zur Erfassung der Revisionsnummer als Zahl wird ein Feld für die Versionsbezeichnung vorgeschlagen, dass als Freitextfeld agieren soll.

**Umsetzung:** Bei der Testfallausführung ist es nun möglich eine Versionsbezeichnung als Freitext zu hinterlegen. Diese ist dann im Testfallprotokoll und beim Export ersichtlich.

**Prognose:** Diese Funktionalität dürfte vor allem zu einer besseren Auswertbarkeit der Testfallprotokolle führen, da so nicht nachvollzogen werden muss, wie die Versionsbezeichnung des Prüflings in einer bestimmten Revision lautet.

### 3.5.6. Begriff „Dauer“ bei Testfall bearbeiten nicht eindeutig (umgesetzt)

**TSM-Kategorien:** *nicht-intuitiv,*
**ISO-Kategorien:** *Erwartungskonformität, Selbstbeschreibungsfähigkeit*
Es wird bemängelt, dass die Bezeichnung „Dauer“ beim Bearbeiten eines Testfalls nicht eindeutig erkennen lässt, dass der Ersteller dort die *geschätzte* Testdauer eintragen kann.

**Umsetzung:** Beim Bearbeiten eines Testfalls ist das Feld zur Eingabe der vom Ersteller geschätzten Durchführungsdauer nun mit „geschätzte Dauer“ beschriftet.

**Prognose:** Die geänderte Bezeichnung dürfte dazu führen, dass häufiger eine geschätzte Dauer vom Testfallersteller verwendet wird.

### 3.5.7. Testzeit bearbeitbar (umgesetzt)

**TSM-Kategorien:** *Erweiterung*
**ISO-Kategorien:** *Fehlertoleranz, Steuerbarkeit*
Es wird gewünscht, die tatsächliche Testdauer, die mittels Stoppuhr in TSM gemessen wurde, bei Eingabe des endgültigen Ergebnisses noch einmal bearbeiten zu können. Dies ist insbesondere dann wichtig, wenn die tatsächliche Dauer als geschätzte Dauer in den Testfall übernommen werden soll.

**Umsetzung:** Am Ende der Testfallausführung kann die benötigte Zeit, die mittels einer Stoppuhr festgestellt wurde, nun durch den Tester bearbeitet werden.

**Prognose:** Diese Funktionalität dürfte zu einer höheren Akzeptanz der Zeiterfassung bei der Testfallausführung führen, da sie korrigierbar ist und somit auch bei vergessener Pausierung für die Zukunft verwendbar ist. Damit geht aber auch ein geringes Manipulationsrisiko einher, da so Tester eine höhere oder geringere Leistung angeben könnten. Allerdings sollten die tatsächliche Dauer der Testfalldurchführung niemals als Bewertungskriterium für die Arbeitsqualität eingesetzt werden.

### 3.5.8. Integration der Versionsverwaltung (umgesetzt)

**TSM-Kategorien:** *Erweiterung, Datenhandhabbarkeit*
**ISO-Kategorien:** *Aufgabenangemessenheit*
Um den Aufwand für die Tester gering zu halten und regelmäßig einen Überblick über den Stand der Tests zu erhalten, soll bei Abschluss einer Testfallausführung das Ergebnis direkt automatisiert dem Versionsverwaltungssystem übergeben werden.

**Umsetzung:** Das Versionsverwaltungssystem Subversion[31] wurde über das Kommandozeilenprogramm integriert. Von einer Verwendung der Java-Anbindung JavaHL[32] oder der Bibliothek SVNKit[33] wurde aus lizenzrechtlichen Gründen abgesehen, da diese nicht zusammen mit TSM ausgeliefert werden dürfen. Diese müssten von Benutzern nachinstalliert werden. Die Verwendung des Kommandozeilenprogramms ist aus Administratorensicht zu bevorzugen, da es automatisiert installiert werden kann.

**Prognose:** Die Funktionalität dürfte insbesondere bei vernetzten Benutzern und Benutzern mit erhöhten Anforderungen an die Dokumentierung von Tests Anklang finden. Aufgrund der Installation eines externen Programms könnte die Einrichtung der Funktionalität – insbesondere auf Windows-Systemen – die Mitwirkung eines Administrators erforderlich machen.

### 3.5.9. Übersicht über bereits bestandene Testfälle (firmenspezifisch, umgesetzt)

**TSM-Kategorien:** *Erweiterung, nicht-inutitiv*
**ISO-Kategorien:** *Erwartungskonformität*
Testfälle, die bereits in früheren Testdurchführungen als „bestanden“ gekennzeichnet wurden, werden in der Firma später nicht mehr erneut getestet. Daher soll in der Übersicht in der ersten Spalte eine Markierung erfolgen, wenn die letzte Ausführung dieses Testfalls den Status „bestanden“ hatte.

**Umsetzung:** In der Übersicht nimmt die erste Spalte nun die Status-Farbe der letzten Ausführung des Testfalls an, so dass nur die erste Spalte betrachtet werden muss, um einen Überblick über den aktuellen Teststand zu erhalten.

**Prognose:** Diese Funktionalität dürfte dafür sorgen, dass schnell ein Überblick über den aktuellen Teststand erhalten werden kann.

### 3.5.10. Speichern-Schaltfläche (umgesetzt)

**TSM-Kategorien:** *Erweiterung, nicht-intuitiv*
**ISO-Kategorien:** *Erwartungskonformität*
In der Werkzeugleiste soll eine Speichern-Schaltfläche verfügbar sein.

[31] Apache Subversion – Siehe: `https://subversion.apache.org/`, abgerufen am 31.10.2014.
[32] JavaHL – Siehe: `https://svn.apache.org/repos/asf/subversion/trunk/subversion/bindings/javahl/README`, abgerufen am 31.10.2014.
[33] SVNKit – Siehe `http://svnkit.com/`, abgerufen am 31.10.2014.

**Umsetzung:** In der Werkzeugleiste existiert nun eine Speichern-Schaltfläche.

**Prognose:** Diese Funktionalität dürfte für eine geringe Zeitersparnis sorgen, da zum Speichern nicht mehr das Datei-Menü aufgeklappt werden muss. Außerdem entspricht die Schaltfläche üblichen Konventionen für Benutzeroberflächen.

### 3.5.11. Benutzeroberfläche vereinfachen/Ansicht über Einstellungen einstellen (umgesetzt)

**TSM-Kategorien:** *Vereinfachung*
**ISO-Kategorien:** *Individualisierbarkeit*
Bei der vereinfachten Oberfläche wurde vorgeschlagen, dass beispielsweise der Filter und die Schnellansicht entfallen. Zusätzlich wäre es denkbar, dass der Tester nur die Testfälle angezeigt bekommt, die ihm zugewiesen wurden.

**Umsetzung:** Die Benutzeroberfläche ist nun beim Programmstart vereinfacht. So sind Schnellansicht und Filter ausgeblendet. Sie können mittels Schaltfläche in der Werkzeugleiste wieder eingeblendet werden. Die Testfälle eines einzelnen Testers können über den Filter selektiert werden.

**Prognose:** Die Änderung dürfte zu einer leichteren Einarbeitung in TSM führen, da der Benutzer nur die absolut notwendigen Elemente sieht. Trotzdem hat er die Möglichkeit, bei Bedarf weitere Informationen und Steuerungsmöglichkeiten zu aktivieren.

### 3.5.12. Revisionsnummer für Ausführungen auch über Neustart hinweg bewahren (umgesetzt)

**TSM-Kategorien:** *Erweiterung*
**ISO-Kategorien:** *Erwartungskonformität*
Es wurde gewünscht, bei der Ausführung von Testfällen die dort angegebene Revision des Prüflings auch über den Neustart hinweg verwenden zu können. Bisher war nach einem Neustart von TSM bei der Ausführung die Revisionsnummer „0“ gesetzt.

**Umsetzung:** Bei der Ausführung wird nun die höchste Revisionsnummer verwendet, die in den zu dem Testfall zugehörigen Testfallprotokollen gefunden wird.

**Prognose:** Die Funktionalität bringt Vorteile in der Benutzerführung, da sie dem Benutzer einen sinnvollen Standardwert anbietet.

### 3.5.13. Erstellen und Bearbeiten von Testfällen während der Ausführung (umgesetzt)

**TSM-Kategorien:** *Vereinfachung*
**ISO-Kategorien:** *keine Entsprechung*
Von Betreuerseite wurde vorgeschlagen, die bei der Testfallausführung vorhandenen Schaltflächen

für das Erstellen und Bearbeiten eines Testfalls zu entfernen, da die Funktionalität bereits mehrfach vorhanden sei.

**Umsetzung:** Bei der Testfallausführung vorhandenen Schaltflächen für das Erstellen und Bearbeiten eines Testfalls wurden entfernt, da die Funktionalität bereits mehrfach vorhanden war und die Schaltflächen nur aus historischen Gründen vorhanden waren.

**Prognose:** Die Änderung dürfte keinen Einfluss auf die Benutzerführung haben, da die Schaltflächen außerhalb der erwarteten Stellen platziert waren. Es ist nicht damit zu rechnen, dass der Wegfall der Schaltflächen von den Benutzern bemerkt wird.

### 3.5.14. Text für „veraltete Protokolle“ (umgesetzt)

**TSM-Kategorien:** *Vereinfachung*
**ISO-Kategorien:** *Aufgabenangemessenheit*
Von Betreuerseite wurde vorgeschlagen, den Hinweis „Veraltetes Protokoll“ bei der Ansicht eines Testfallprotokolls zu ändern. Der Hinweis erscheint, wenn der Testfall, auf den sich das Protokoll bezieht seit der Ausführung, die im Testfallprotokoll festgehalten wurde, geändert wurde. Das Testfallprotokoll ist aber nicht veraltet, sondern gibt den zum Zeitpunkt der Testfallausführung aktuellen Testfallzustand wieder.

**Umsetzung:** Der Text wurde in „Von diesem Testfall liegt eine neuere Version vor.“ geändert.

**Prognose:** Diese Änderung dürfte einen allgemeinen Mehrwert mit sich bringen, da sie eine Meldung verständlicher macht.

### 3.5.15. Erweitern-Schaltfläche (umgesetzt)

**TSM-Kategorien:** *Erweiterung*
**ISO-kategorien:** *Aufgabenangemessenheit, Erwartungskonformität*
Der Autor schlug eine Erweitern-Schaltfläche für den TSM-Navigator vor, der alle Elemente aufklappt.

**Umsetzung:** Der TSM-Navigator verfügt nun über eine Erweitern-Schaltfläche, mit der alle Element aufgeklappt werden können.

**Prognose:** Die Funktionalität wird in bestimmten Situationen eine Erleichterung für Benutzer mit sich bringen. So wird vermieden, dass der Benutzer alle Elemente einzeln aufklappen muss, wenn diese – durch das Eclipse-Framework bedingt - automatisch zugeklappt wurden.

### 3.5.16. Benutzerhandbuch (umgesetzt)

**TSM-Kategorie:** *nicht-dokumentiert*
**ISO-kategorien:** *keine Entsprechung*
Die Erweiterung des Benutzerhandbuchs wurde nicht vorgeschlagen, ist aber Standard in der Softwaretechnik, da sich der Funktionsumfang geändert und erweitert hat.

**Umsetzung:** Das Benutzerhandbuch wurde an geänderte und erweiterte Funktionalität angepasst.

**Prognose:** Ein angepasstes Benutzerhandbuch ermöglicht dem Benutzer die Möglichkeit, technisch umfangreichere Aufgaben korrekt umsetzen zu können.

### 3.5.17. Einfügen von Dateien in Testfälle

**TSM-Kategorien:** *Erweiterung*
**ISO-Kategorien:** *Aufgabenangemessenheit*
Es sollen neben Bildern weitere Dateien in einen Testfall eingefügt oder referenziert werden können. Dies wäre beispielsweise für Testdaten denkbar.

**Keine Umsetzung:** Eine Umsetzung hätte erhebliche Eingriffe in den Richtexteditor nötig gemacht. Da dessen Wartbarkeit sehr gering ist, wäre der Zeitaufwand für die Umsetzung unverhältnismäßig hoch gewesen.

### 3.5.18. Erweiterte Berichte

**TSM-Kategorien:** *Erweiterung*
**ISO-Kategorien:** *Aufgabenangemessenheit*
Von Führungskräften wurde insbesondere Wert auf erweiterte Berichtsmöglichkeiten gelegt.

Die bereits in TSM enthaltene Übersichtsseite wird generell begrüßt, ihr Umfang und Inhalt könnte noch mehr Informationen enthalten. Die Informationen könnten durch Diagramme oder Grafiken weiter aufbereitet werden. Eine abstrakte Darstellung ähnlich einem Projektmanagementwerkzeug wird dabei bevorzugt, um auch Benutzern, die nicht am Testen beteiligt sind, schnell einen Überblick über den Teststatus und -fortschritt zu verschaffen.

Die Übersichtsseite soll ebenfalls per PDF-Export exportiert werden können, um Berichte für z. B. Besprechungen verfügbar zu haben. Sofern möglich, sollen dabei die abstrakten Darstellungen ebenfalls exportiert werden können.

Welche Informationen hierbei wichtig sind oder gewünscht werden, ist den Probanden selbst noch unklar.

**Keine Umsetzung:** In Ermangelung konkreter Vorstellungen der erweiterten Berichte erfolgte keine Umsetzung.

### 3.5.19. Bugtracker-Anbindung

**TSM-Kategorien:** *Erweiterung*
**ISO-Kategorien:** *Aufgabenangemessenheit*
Zur einfachen Dokumentation der beim Testen gefundenen Fehler wird eine Anbindung innerhalb von TSM gewünscht, die es ermöglicht, Fehlerberichte direkt in ein Bugtrackingsystem einzugeben. Favorisiert wird hierbei die Anbindung von TSM an Mylyn[34] welches wiederum über Anbindungen zu gängigen Bugtrackingsystemen verfügt.[35]

**Keine Umsetzung:** Da die Dokumentation von Mylyn nicht ausreichend war, um mit einem verhältnismäßigen Zeitaufwand eine Anbindung umzusetzen, erfolgte keine Umsetzung dieses Vorschlags.

### 3.5.20. Rückschluss von Bugtracker-Ticket zu Testfall

**TSM-Kategorien:** *Erweiterung*
**ISO-Kategorien:** *keine Entsprechung*
Zusätzlich zur Bugtracker-Anbindung wurde eine „Rückwärtsverbindung“ von einem Ticket im Bugtracker zum dazugehörigen Testfall vorgeschlagen. So soll man von einem Ticket im Bugtracker herausfinden können, in welchem Testfall der Fehler gefunden wurde.

**Keine Umsetzung:** Dieser Änderungsvorschlag ist als ursprünglich firmenspezifisch anzusehen, ist bei näherer Betrachtung allerdings auch durchaus für die Allgemeinheit nützlich. Da bei der Entwicklung von TSM mit Absicht keine Identifikationsnummern für Testfälle – für den Benutzer sichtbar – verwendet wurden, würde eine Umsetzung dieses Vorschlags größere Änderungen im Benutzungskonzept und Datenmodell erforderlich machen.

### 3.5.21. Installationsprogramm für TSM

**TSM-Kategorien:** *Erweiterung, nicht-intuitiv*
**ISO-Kategorien:** *Erwartungskonformität, Fehlertoleranz*
Für die alleinstehende Variante von TSM soll ein Installationsprogramm zur Verfügung gestellt werden, dass TSM entpackt und z. B. Verknüpfungen auf dem Windows-Desktop oder -Startmenü auf die „eclipse.exe“ erstellt.

**Keine Umsetzung:** Da TSM als Zip-Datei ausgeliefert wird, muss es nur noch ausgepackt werden und ist direkt lauffähig. Ein Installationsprogramm würde die Größe der ausgelieferten Version erhöhen und keinen großem Mehrwert mit sich bringen.

[34] Bei Mylyn handelt es sich um ein Framework für Eclipse. Siehe: `http://www.eclipse.org/mylyn/`, abgerufen am 23.09.2014.

[35] Derzeit unter anderem Bugzilla, Trac, Jira und über den Generic Web Templates Connector Google Code Hosting, IssueZilla, GForge, Sourceforge, Mantis, ChangeLogic, OTRS, phpBB und vBulletin. Siehe: `http://wiki.eclipse.org/Mylyn_User_Guide#Task_Repository_Connectors`, abgerufen am 24.09.2014.

### 3.5.22. Automatische Generierung von Revisionsnummern

**TSM-Kategorien:** *Erweiterung*
**ISO-Kategorien:** *Aufgabenangemessenheit*
Es wurde der Wunsch nach einer automatischen Generierung von Revisionsnummern genannt. Was genau damit gemeint war, wurde nicht klar und konnte aufgrund der Anonymisierung der Fragebögen auch nicht ermittelt werden.

**Keine Umsetzung:** Da der Änderungsvorschlag unklar war, konnte er nicht umgesetzt werden.

### 3.5.23. Zusätzliches Feld zur Erfassung der Ticketnummer des Bugtrackers (firmenspezifisch)

**TSM-Kategorien:** *Erweiterung*
**ISO-Kategorien:** *Selbstbeschreibungsfähigkeit*
Im Testfall oder bei der Ausführung eines Testfalls soll die Ticketnummer eines Bugtracker erfasst werden können.

**Keine Umsetzung:** Dieser Vorschlag wurde nicht umgesetzt, da er zu firmenspezifisch war und keinen allgemeinen Mehrwert mit sich bringt.

### 3.5.24. Bessere Führung bei Testfallausführung durch das Ausgrauen der nicht aktiven Testschritte

**TSM-Kategorien:** *Erweiterung, nicht-intuitiv*
**ISO-Kategorien:** *Selbstbeschreibungsfähigkeit*
Bei der Testfalldurchführung soll immer nur ein Testschritt aktiv sein, die anderen sollen ausgegraut sein.

**Keine Umsetzung:** Wenngleich der Vorschlag hervorragend zur Führung des Benutzers durch den Dialog geeignet ist, konnte er aufgrund von konzeptionellen und auch technischen Gründen nicht umgesetzt werden. So ist insbesondere unklar, wie erkannt werden soll, welcher Testschritt aktiv ist oder aus Benutzersicht aktiv sein soll.

### 3.5.25. Kopieren und Einfügen von Word

**TSM-Kategorien:** *Erweiterung*
**ISO-Kategorien:** *Erwartungskonformität*
Es hat sich gezeigt, dass beim Kopieren und Einfügen von Microsoft Word in TSM HTML-Steuerzeichen eingefügt werden, da es es sich um bereits formatierten Text handelt. Die Steuerzeichen sollen beim Einfügen nicht erscheinen.

**Keine Umsetzung:** Dieser Vorschlag wurde nicht umgesetzt, da der technische Aufwand nicht verhältnismäßig gewesen wäre.

### 3.5.26. Übersicht über weggeworfene Versionen (firmenspezifisch)

**TSM-Kategorien:** *Erweiterung*
**ISO-Kategorien:** *keine Entsprechung*
So fern Testfälle für neue Versionen so erzeugt werden, dass die bereits vorhandenen kopiert und modifiziert werden, soll es möglich sein, nachzuvollziehen, welche Testfälle dabei gelöscht worden sind.

**Keine Umsetzung:** Dieser Vorschlag wurde nicht umgesetzt, da er zu firmenspezifisch war und keinen allgemeinen Mehrwert mit sich bringt. Die gewünschte Funktionalität ist mittels Versionsverwaltung umsetzbar.

### 3.5.27. Abnehmer für Testfälle und -daten (firmenspezifisch)

**TSM-Kategorien:** *Erweiterung*
**ISO-Kategorien:** *keine Entsprechung*
Die Testfälle werden in der Firma von einer Person abgenommen, bevor sie den Testern zur Ausführung zur Verfügung gestellt werden. In TSM sollte es daher möglich sein, eine Person als Abnehmer bereits bei der Erstellung des Testfalls festlegen zu können.

**Keine Umsetzung:** Dieser Vorschlag wurde nicht umgesetzt, da er zu firmenspezifisch war und keinen allgemeinen Mehrwert mit sich bringt.

### 3.5.28. Prioritäten anders benennen (firmenspezifisch)

**TSM-Kategorien:** *Erweiterung, nicht-intuitiv*
**ISO-Kategorien:** *Individualisierbarkeit*
Die Prioritäten die einem Testfall zugewiesen werden können sind derzeit mit „niedrig“, „mittel“ und „hoch“ bezeichnet. Diese sollten gegebenenfalls den Bezeichnungen des Unternehmens angepasst werden.

**Keine Umsetzung:** Dieser Vorschlag wurde nicht umgesetzt, da er zu firmenspezifisch war und keinen allgemeinen Mehrwert mit sich bringt.

### 3.5.29. Allgemeines-Hinweis-Feld im Testfall (firmenspezifisch)

**TSM-Kategorien:** *Erweiterung*
**ISO-Kategorien:** *Selbstbeschreibungsfähigkeit*
Es wurde der Wunsch geäußert, den Testern über ein Feld allgemeine Hinweise zu geben, die sich nicht direkt auf den Testfall beziehen. Als Beispiel wurde genannt, dass die Tester auch auf andere (Fehler-)Meldungen achten sollen, die nicht direkt mit dem gerade ausgeführten Testfall zu tun haben.

**Keine Umsetzung:** Dieser Vorschlag wurde nicht umgesetzt, da er zu firmenspezifisch war und keinen allgemeinen Mehrwert mit sich bringt. Die gewünschte Funktionalität lässt sich mittels

zusätzlichen Testschritten oder Angaben in der Vorbedingung erreichen. Mit der Verwendung von zusätzlichen Testschritten ist zudem sichergestellt, dass der Tester explizit bestätigen muss, dass er beispielsweise auf andere (Fehler-)Meldungen geachtet hat; bei einem separaten Hinweisfeld wäre dies nicht gewährleistet.

### 3.5.30. Testsequenz/Auswahl ausführbarer Testfälle

**TSM-Kategorien:** *Erweiterung*
**ISO-Kategorien:** *Selbstbeschreibungsfähigkeit*
Es wird eine Möglichkeit zur Hintereinanderausführung mehrere Testfälle ähnlich einer Testsequenz gewünscht. Alternativ wäre auch denkbar, dass mehrere Testfälle vorausgewählt werden können und anschließend der Reihe nach ausgeführt werden.

**Keine Umsetzung:** Dieser Vorschlag wurde im Rahmen dieser Arbeit nicht umgesetzt, da bei der Entwicklung von TSM explizit auf das Konzept von Testsequenzen verzichtet wurde, da diese Komplexität der Daten, aber auch der Benutzerführung erhöhen. Trotz alledem sollte eine zukünftige Umsetzung in der Erwägung gezogen werden, da es bei einer guten Umsetzung in Bezug auf Benutzbarkeit dem Benutzer eine wichtige Hilfestellung sein kann. Zurzeit muss dieser nämlich noch selbst vor Ausführung eines Testfalls nachschauen, ob er diesen für die aktuelle Revision des Prüflings bereits ausgeführt hat oder dies noch tun muss.

### 3.5.31. Aktion und erwartetes Ergebnis nur lesbar (firmenspezifisch)

**TSM-Kategorien:** *Erweiterung*
**ISO-Kategorien:** *Steuerbarkeit, Fehlertoleranz*
Um unbeabsichtigter oder sogar mutwilliger Veränderung der Testfälle vorzubeugen, sollen die Felder „Aktion“ und „Erwartetes Ergebnis“ bei der Testfallausführung schreibgeschützt werden können.

**Keine Umsetzung:** Dieser Vorschlag wurde nicht umgesetzt, da TSM über keine Benutzer- oder Rechteverwaltung verfügt und daher ein solcher Schutz – zumindest gegen mutwillige Veränderungen – nicht wirksam wäre. Im Hinblick auf unbeabsichtigte Veränderungen sollten organisatorische Maßnahmen beim Benutzer getroffen werden, um dies zu verhindern.

### 3.5.32. Endgültiges Ergebnis (firmenspezifisch)

**TSM-Kategorien:** *Erweiterung*
**ISO-Kategorien:** *keine Entsprechung*
Es wird gewünscht, die Eingabe des endgültigen Ergebnisses weiter unterteilen zu können. Denkbar sind hier Unterteilungen wie „Allgemeiner Eindruck des Prüflings“ und „Änderungsvorschläge am Testfall“.

**Keine Umsetzung:** Dieser Vorschlag wurde nicht umgesetzt, da er firmenspezifisch war und keinen allgemeinen Mehrwert mit sich bringt. Dabei wurde insbesondere berücksichtigt, dass bei einem bestanden Testfalldurchführung es unwahrscheinlich ist, dass das Testfallprotokoll erneut gelesen

wird und somit Änderungsvorschläge am Testfall nicht zwingend beachtet werden. Zusätzlich ist es bereits möglich, einen Testfall direkt bei Ausführung zu modifizieren, so fern dies nötig ist.

### 3.5.33. Generierung von Testfällen aus Bugtracker-Tickets (firmenspezifisch)

**TSM-Kategorien:** *Erweiterung*
**ISO-Kategorien:** *keine Entsprechung*
Zum Testen von behobenen Fehlern ist es bisher Praxis bei einer der Firmen, Testfälle zu erstellen, die den Tester auffordern, das Problem aus einem bestimmten Ticket nachzuvollziehen und zu überprüfen, ob dieses behoben wurde. Dabei enthalten die Testfälle keine expliziten Testanleitungen sondern lediglich Anweisungen der Form „Prüfen Sie das Ticket mit der Nr. X“. Um die Erstellung solcher Testfälle zu vereinfachen wäre es wünschenswert, wenn man durch Auswahl von Tickets automatisch solche Testfälle erzeugen lassen könnte.

**Keine Umsetzung:** Dieser Vorschlag wurde nicht umgesetzt, da er zu firmenspezifisch war und keinen allgemeinen Mehrwert mit sich bringt.

### 3.5.34. Statistikseite nicht genutzt

**TSM-Kategorien:** *nicht-intuitiv*
**ISO-Kategorien:** *Selbstbeschreibungsfähigkeit*
Es wurde – auf Nachfrage – berichtet, dass die Statistikseite unbekannt sei und daher ungenutzt geblieben sei. Dies ist vermutlich darauf zurückzuführen, dass die Seite sehr versteckt als zweite Registerkarte implementiert ist. Die dazugehörigen Registerreiter sind dabei atypisch am unteren Seitenrand zu finden.

**Keine Umsetzung:** Die Sichtbarkeit der Statistikseite wurde nicht geändert, da ihr Inhalt aufgrund der Darstellungsform von geringem Mehrwert ist und der Zeitaufwand dazu nicht im Verhältnis stehen würde.

## 3.6. Schwierigkeiten bei der Umsetzung

Im Folgenden wird ein Überblick über die technischen Probleme der Umsetzung gegeben.

### 3.6.1. TSM-spezifische Probleme

#### 3.6.1.1. Quelltextqualität

Der während des Studienprojekts entstandenen Quelltext wurde von acht Personen mit unterschiedlicher Programmierweise erstellt und nur minimaler Qualitätssicherung unterzogen. Dementsprechend niedrig war seine Qualität zu Beginn der Bachelorarbeit. Die Verbesserung des Quelltextes war zwar

nicht Inhalt der Bachelorarbeit aber notwendiges „Übel“[36] um Anpassungen vornehmen zu können. Im Folgenden wird auf die dabei gefundenen Hauptprobleme eingegangen.

**Bezeichner** Viele Bezeichner wie Klassen- und Methodennamen, Objekte und Variablen waren nicht „sprechend“ benannt.[37] Oft handelte es sich um Bezeichner, die nur ein bis zwei Buchstaben lang waren.

**Dokumentation** Die vorhandene Dokumentation beinhaltete vor allem die separate Dokumentation in Form eines Handbuchs, welches sich an Endanwender richtet. Die integrierte Dokumentation in Form von (Javadoc[38]-)Kommentaren war teilweise unvollständig, unverständlich und sogar vereinzelt widersprüchlich.

**Komplexität** Die Komplexität vieler Klassen ist durch ihre Länge bedingt sehr groß. In der McCabe-Metrik[39] gemessen[40] war der Höchstwert 229 von empfohlenen 10. Nahe an diesen Wert kamen weitere Klassen[41] mit den Werten 214, 207, 174 und 140.
Eine umfassende Komplexitätsreduzierung war im Rahmen dieser Arbeit nicht möglich, da es hierzu umfangreicher Umbauarbeiten der Architektur bedurft hätte.

**Programmierrichtlinien** Die Oracle-Programmierrichtlinie für Java[42] wurde nur bedingt eingehalten. So waren beispielsweise an vielen Stellen Konstrukte zu finden, die zwar grammatikalisch korrekt waren, aber deren Lesbarkeit gering war und die bei Umstrukturierungen zu Problemen führen können.[43]

[36] Wartung sollte nie als Übel angesehen werden – hebt sie doch im Idealfall die Qualität – , in der Praxis stellt sie sich doch zuweilen als sehr mühsam dar.

[37] Bezeichnungen die so gewählt sind, dass sie direkt erkennen lassen, welche Funktion sie erfüllen.

[38] Javadoc ist ein Werkzeug zur Generierung von HTML-Dokumentation aus speziell formatierten Kommentaren. Siehe `http://www.oracle.com/technetwork/java/javase/documentation/javadoc-137458.html`, abgerufen am 07.10.2014.

[39] Eine von Thomas J. McCabe 1976 entworfene Metrik die anhand der Verzweigungen eines Quelltextes berechnet wird. Siehe [McC76].

[40] Gemessen mit SonarQube in der Version 3.5.1 für die Klasse „DataModel“ in der Revision 15 des Hauptzweiges „trunk“ auf Sourceforge. Siehe zur Komplexitätsmetrik in SonarQube `http://docs.codehaus.org/display/SONAR/Metric+definitions`, abgerufen am 07.10.2014.

[41] „Richtext“, „ExportPDF“, „BreadcrumbViewer“ und „Overview“ in der hier genannten Reihenfolge, jeweils in der Revision 15 des Hauptzweiges „trunk“ auf Sourceforge.

[42] Oracle: *Code Conventions for the Java Programming Language.* 1999. `http://www.oracle.com/technetwork/java/codeconvtoc-136057.html`

[43] Beispielhaft seien hier If-Anweisungen mit Einrückungen aber ohne geschweifte Klammern zu nennen. Hier ist nur die darauf folgende Zeile Bestandteil des If-Blocks. Wird die Einrückung aber geändert oder nicht korrekt eingehalten, so kann dies dazu führen, dass der Block als größer angesehen wird.

#### 3.6.1.2. Architektur

Die Architektur von TSM wurde schon zur Entwicklungszeit nicht vollständig geplant und dokumentiert, da diese sich an der Architektur des Eclipse-Frameworks orientieren musste und von diesem keine genaue Vorstellung herrschte.

### 3.6.2. Eclipse-Framework

#### 3.6.2.1. Dokumentation

Zwar verfügt das Eclipse-Framework über eine umfangreiche Dokumentation, diese bietet allerdings – je nach Thema – zu viel oder zu wenig Detailtiefe. Die verfügbaren Beispiele sind meistens veraltet oder gar nicht mehr im Internet verfügbar.

#### 3.6.2.2. Erstellen der alleinstehenden Version

Die Erstellung der alleinstehenden Version von TSM erfordert den Export aus Eclipse heraus, bei dem Abhängigkeiten berücksichtigt werden und TSM mit den benötigten Eclipse-Plugins als „Paket“ gepackt wird. Hierbei kommt es regelmäßig zu erheblichen Problemen, da erforderliche Eclipse-Plugins nicht verfügbar sind oder in einer falschen Version vorliegen oder trotz Vorhandenseins als fehlend markiert werden.

# 4. Zusammenfassung und Ausblick

Schon zum Ende des Studienprojekts im Jahre 2012 war ersichtlich, dass TSM – trotz der damaligen Schwierigkeiten – auf dem richtigen Weg war, um ein nützliches Werkzeug beim manuellen Testen zu werden. Diese Bachelorarbeit hat ebenfalls gezeigt, dass TSM als Werkzeug für die Praxis akzeptiert wird und es oft nur noch kleinerer Verbesserungen der Benutzerführung bedarf, um die Akzeptanz bei den Benutzern zu erhöhen. Vielfältige Rückmeldungen aus der Industrie haben gezeigt, dass verbesserungswürdige Stellen eher in der Interaktion mit der Software und Infrastruktur um TSM herum als in Kernfunktionalitäten besteht. So waren öfters geäußerte Wünsche Daten zu ex- und importieren und Überblicke über den aktuellen Zustand der Tests zu exportieren. Bedauerlicherweise setzt hier die Architektur von TSM gewisse Grenzen, die zuvörderst durch das Eclipse-Framework bestimmt sind.

## Ausblick

Die im Rahmen dieser Arbeit entstandenen Funktionalitäten stehen jedermann über die Sourceforge-Plattform zur Verfügung und finden vielleicht auch an anderer Stelle Verwendung ohne dass man dies jemals erfahren würde.[44] [45] Die erhobenen Wünsche und Anforderungen sind nicht zwingend an die aktuelle Implementierung gebunden, sondern sind in der Mehrheit Anforderungen an die Benutzeroberfläche. Zur Verschiebung der technischen Grenzen, die durch das Eclipse-Framework vorgegeben werden, ist eine Neuimplementierung mittels anderer Technologie denkbar. Bei Übernahme der bekannten Benutzungskonzepte dürften die Umstellungsschwierigkeiten für die Benutzer gering sein.

Um hohe Akzeptanz bei Benutzern zu erreichen, ist es wichtig, dass auch noch so minimale oder ungewöhnliche Änderungswünsche von Benutzer nicht verworfen oder gar als unsinnig bezeichnet werden, sondern einer sorgfältigen Prüfung unterzogen werden. Gerade im Free/Libre-Open-Source-Bereich (FLOSS) sind hier häufig noch große Defizite festzustellen. So ergab eine Studie mit 72 FLOSS-Entwicklern aus dem Jahr 2012, dass für 70 % der Befragten Usability keine hohe Priorität hat.[46]

[44] Zum 07.01.2015 verzeichnete Sourceforge 1408 Downloads für das Projekt. Diese Zahl bezieht sich auf alle Dateien, die im Projekt zur Verfügung gestellt werden (Programm, Handbuch, Änderungsnotizen, …). Siehe `https://sourceforge.net/projects/tsmtest/files/stats/timeline?dates=2013-03-14+to+2015-01-07`, abgerufen am 07.01.2015.

[45] Ein Entwickler aus der Republik Südafrika verweist beispielsweise in einem Forumseintrag auf TSM. Siehe `http://www.dotnetfunda.com/forums/show/15445/free-testing-tool-for-windows-application`, abgerufen am 11.11.2014.

[46] [RC12]

Testwerkzeuge dürften in Zukunft weiter Zuspruch und Zulauf finden, da die Automatisierung des Systemtests nach einer Studie mehrerer Hochschulen aus dem Jahr 2011 nicht sehr hoch ist.[47] Als am meisten genannte Testaktivität für den Einsatz von Werkzeugen wurde dabei das Ausführen von Testfällen genannt, als zweithäufigste das Generieren von Testfällen und als dritthäufigste das Spezifizieren von Testfällen.[48]
Die Werkzeugunterstützung für das manuelle Testen wird also auch in Zukunft Bestandteil des Testprozesses sein und sollte eine gute Benutzerführung anbieten, um eine hohe Akzeptanz zu erreichen.

[47] [VSWVP13], S. 194.
[48] [VSWVP13], S. 196.

# A. Quelltextmetriken

Als „Nebenprodukt“ entstanden bei dieser Arbeit auch Quelltextmetriken über TSM. Diese werden hier kurz dargestellt. Hierbei ist zu beachten, dass es für objektorientierte Programmiersprachen wie Java besser geeignete Metriken gibt, um einen Eindruck vom Zustand einer Software zu erhalten. Da der Fokus der Arbeit nicht in der Erfassung der Metriken lag, wurden nur diejenigen erfasst, für die technische Unterstützung vorhanden war. Trotzdem können die hier verwendeten Metriken dazu genutzt werden, Vergleiche mit anderen Softwareprojekten zu erstellen.

## A.1. Metrikenerhebung

In diesem Abschnitt wird beschrieben wie über den Quelltext von TSM Metriken erhoben wurden.

### A.1.1. Werkzeuge

Als Werkzeug kam SonarQube[49] in der Version 3.5.1 zum Einsatz. Es wurden Analysen mit dem Quelltextstand vor Beginn dieser Arbeit und zum Stand gegen Ende dieser Arbeit erhoben.

### A.1.2. Ausgewählte Metriken

#### A.1.2.1. Lines of Code

Anzahl der physischen Quelltextzeilen, die mindestens ein Zeichen enthalten, das kein Leerzeichen, Tabulator oder Teil eines Kommentars ist.[50]

#### A.1.2.2. Komplexität

Die Komplexitätsmetrik entspricht der McCabe-Metrik[51]

[49] Siehe `http://www.sonarqube.org/`

[50] Siehe die Definition von SonarQube `http://docs.sonarqube.org/display/SONAR/Metric+definitions`, abgerufen am 29.12.2014.

[51] Eine von Thomas J. McCabe 1976 entworfene Metrik die anhand der Verzweigungen eines Quelltextes berechnet wird. Siehe [McC76]. Siehe zur Komplexitätsmetrik in SonarQube `http://docs.sonarqube.org/display/SONAR/Metric+definitions`, abgerufen am 29.12.2014.

#### A.1.2.3. Kommentare (%)

Die Dichte der Kommentarzeilen.

#### A.1.2.4. Duplizierte Zeilen (%)

Anteil der duplizierten Quelltextzeilen.

#### A.1.2.5. Dokumentierte öffentliche API (%)

Anteil der öffentlichen API, die dokumentiert ist.

### A.1.3. Vorher-Nachher-Vergleich

| **Metrik** | **Vorher** | **28.11.2014** |
|---|---|---|
| **Lines of Code** | 20 320 | 21 286 |
| **Komplexität** | 3787 | 4026 |
| **Kommentare (%)** | 19,1 % | 19,3 % |
| **Duplizierte Zeilen (%)** | 5,0 % | 4,1% |
| **Dokumentierte öffentliche API (%)** | 31,8 % | 36,6 % |

„Vorher“ bezieht sich auf den Zeitpunkt vor Beginn dieser Arbeit.

# B. LaTeX-Vorlage

## B.1. Verwendete Vorlage und deren Lizenz

Diese Arbeit wurde mit der Vorlage „uni-stuttgart-computer-science-template“[52] erstellt. Die Vorlage trägt den folgenden Urheber- und Lizenzvermerk in der Datei „LICENSE“, der keine Anwendung auf die vorliegende Arbeit findet:

„The MIT License (MIT)

Copyright (c) 2005-2008 Oliver Kopp
Copyright (c) 2009-2010 Oliver Kopp, Bastian Reitschuster
Copyright (c) 2012 Oliver Kopp, Kai Mindermann, Niklas Schnelle
Copyright (c) 2013 Oliver Kopp

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.“

[52] `https://github.com/latextemplates/uni-stuttgart-computer-science-template`, abgerufen am 07.01.2015.

# Literaturverzeichnis


[Boe77] B. W. Boehm. The High Cost of Software. *COMPSAC*, S. 4–15, 1977. (Zitiert auf Seite 9)

[DIN08] DIN EN ISO 9241-110:2008-09 – Ergonomie der Mensch-System-Interaktion – Teil 110: Grundsätze der Dialoggestaltung, 2008. (Zitiert auf den Seiten 21, 22 und 23)

[LL10] J. Ludewig, H. Lichter. *Software Engineering: Grundlagen, Menschen, Prozesse, Techniken.* dpunkt-Verl., Heidelberg, 2., überarb., aktualisierte und erg. aufl. Auflage, 2010. (Zitiert auf Seite 14)

[McC76] T. J. McCabe. A Complexity Measure. *IEEE Transactions on Software Engineering*, SE-2(4):308–320, 1976. URL `http://www.literateprogramming.com/mccabe.pdf`. (Zitiert auf den Seiten 35 und 39)

[RC12] A. Raza, L. Capretz. Do open source software developers listen to their users? *First Monday*, 17(3), 2012. URL `http://firstmonday.org/ojs/index.php/fm/article/view/3640`. (Zitiert auf Seite 37)

[Rie97] E. H. Riedemann. *Testmethoden für sequentielle und nebenläufige Software-Systeme.* Teubner, Stuttgart, 1997. URL `https://www-secse.cs.tu-dortmund.de/secse/pages/teaching/riedemann/index_de.shtml`. (Zitiert auf Seite 9)

[VSWVP13] K. Vosseberg, A. Spillner, M. Winter, K. Valbert-Polenske. Technischer Report zur Umfrage 2011: Softwaretest in der Praxis. Technischer Bericht SW-TU TR 13-1, Hochschule Bremerhaven, Hochschule Bremen und Fachhochschule Köln, 2013. URL `http://www.softwaretest-umfrage.de/TechnischerReport130317.pdf`. (Zitiert auf Seite 38)

**Erklärung**

Ich versichere, diese Arbeit selbstständig verfasst zu haben. Ich habe keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommene Aussagen als solche gekennzeichnet. Weder diese Arbeit noch wesentliche Teile daraus waren bisher Gegenstand eines anderen Prüfungsverfahrens. Ich habe diese Arbeit bisher weder teilweise noch vollständig veröffentlicht. Das elektronische Exemplar stimmt mit allen eingereichten Exemplaren überein.

---

Ort, Datum, Unterschrift