Masterarbeit am Institut für Informatik, Arbeitsgruppe Software Engineering
FG Modellbasierte Entwicklung und Qualitätssicherung
Die Arbeit entsteht in Zusammenarbeit mit der BIOTRONIK SE & Co. KG.

Masterarbeit

Untersuchung und Verbesserung der Testqualität in den Bereichen Testeffektivität und Wartbarkeit für die automatischen HMSC3-Val-Tests

vorgelegt von

Thea Schröter

Matrikelnummer: 4491806

thea.schroeter@fu-berlin.de

zur Erlangung des akademischen Grades

Master of Science (M.Sc.)

Erstprüfer: Prof. Dr.-Ing. I. Schieferdecker

Zweitprüfer: Prof. Dr. L. Prechelt

Berlin, 7. Juli 2013

Eidesstattliche Erklärung

Ich versichere hiermit an Eides Statt, dass diese Arbeit von niemand anderem als meiner Person verfasst worden ist. Alle verwendeten Hilfsmittel wie Berichte, Bücher, Internetseiten oder ähnliches sind im Literaturverzeichnis angegeben, Zitate aus fremden Arbeiten sind als solche kenntlich gemacht. Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungskommission vorgelegt und auch nicht veröffentlicht.

Berlin, 7. Juli 2013 .....

Thea Schröter

Zusammenfassung

Manuelles Bewerten von Testergebnissen ist bei einer erhöhten Testanzahl fehleranfällig und zeitaufwändig. Auch die Erstellung logischer Testfälle in einem Test-Management-Tool und die davon losgelöste manuelle Erstellung von Testskripten führen zu Defiziten in der Testqualität, da die Nachverfolgbarkeit von Anforderungen nicht sichergestellt werden kann. Mängel in der Testqualität sind dabei insbesondere in den Bereichen Testeffektivität und Wartbarkeit festzustellen.

In der Validierungsgruppe des Home Monitoring Systems bei Biotronik werden für verschiedene automatisierte System-Tests diese manuellen Verfahren angewandt. Da die System-Tests regelmäßig ausgeführt werden, besteht ein großes Fehlerpotential. Es ist das Ziel die Testqualität in den genannten Schlüsselbereichen zu verbessern und so einen Leistungsgewinn zu erwirken.

In dieser Arbeit wird durch eine Kombination aus statischer Analyse und der Goal Question Metric die Testqualität der automatisierten Tests untersucht. Hierbei werden durch den Einsatz der Goal Question Metric die zu messenden Eigenschaften gezielt eingegrenzt. Durch die Anwendung quantitativer Metriken können leicht verständliche und vergleichbare Daten erhoben werden. Auf der Basis festgestellter Defizite werden konkrete Zielvorgaben zur Verbesserung und zum Erhalt der Testqualität aufgestellt. Im Rahmen dieser Arbeit wird in Verbindung mit den Zielvorgaben ein neues Muster für Testspezifikationen und Testskripte vorgestellt. Des Weiteren soll die Entwicklung eines Tools zur Übertragung von Testergebnissen die Nachverfolgbarkeit von Anforderungen werkzeugtechnisch sicherstellen.

Danksagung

Die Erstellung dieser Arbeit wäre ohne die Unterstützung zahlreicher Personen nicht möglich gewesen.
Ich bedanke mich bei Frau Professor Dr. Ing. Schieferdecker, dass sie das Entstehen dieser Arbeit unterstützte.

Die Mitarbeiter der Firma Biotronik waren mir bei der Einführung in die Thematik und der Erfassung technischer Zusammenhänge eine große Hilfe. Mein Dank gilt hierbei vor allem Herrn Hans Greiner, der als persönlicher Betreuer vor Ort meinen Fragen stets aufgeschlossen und interessiert gegenüberstand sowie Herrn Uwe Katt, der sich engagiert um die organisatorischen Abläufe kümmerte.

Ein besonderer Dank gilt meiner Mutter Dörte Schröter und meinem Freund Christian Milkus, die mich beim Lektorat der Arbeit unterstützten. Für die weitere Hilfe in dieser Zeit möchte ich einen Dank an meine weitere Familie und meine Kommilitonen aussprechen.

Inhaltsverzeichnis

Zusammenfassung **III**

Abkürzungsverzeichnis **IX**

1 Einleitung **1**
1.1 Motivation . . . 1
1.2 Problemstellung . . . 2

2 Grundlagen **3**
2.1 Begriffserklärungen . . . 3
2.1.1 International Software Testing Qualifications Board (ISTQB) . . . 3
2.1.2 Testqualität und Qualitätsmerkmale . . . 4
2.1.3 Weitere Begriffserklärungen . . . 6
2.2 Systemlandschaft . . . 8
2.2.1 System under Test – HMSC . . . 8
2.2.2 Test-Tool – JavaTT und HMSC3-Val-Tests . . . 8
2.2.3 Testmanagementtool – QC/ALM . . . 11
2.3 Begriffsgegenüberstellungen . . . 13

3 Statische Analyse **15**
3.1 Ermittlung der Metriken mit der Goal Question Metric . . . 15
3.1.1 Testeffektivität . . . 16
3.1.2 Wartbarkeit . . . 18
3.2 Auswertung der statischen Analyse . . . 23
3.2.1 Auswertung der Metriken . . . 23
3.2.2 Einhaltung der Testeffektivität oder Wartbarkeit . . . 33
3.3 Zielvorgaben für die Verbesserung der Testqualität . . . 33
3.3.1 Zielvorgaben für die Verbesserung der Testeffektivität . . . 33
3.3.2 Zielvorgaben für die Verbesserung der Wartbarkeit . . . 34

4 Implementierung **36**
4.1 Aufbau der Musterstruktur . . . 36
4.1.1 Konzept . . . 36
4.1.2 Umsetzung am Test „Patient Option Template“ . . . 38
4.2 Tracing durch Werkzeugsteuerung . . . 39
4.2.1 Ablauf . . . 40

4.2.2 Voraussetzungen 41
4.2.3 Implementierungsdetails 44

5 Bewertung der erreichten Verbesserungen 52

6 Zusammenfassung 58
6.1 Fazit 58
6.2 Ausblick 59

Tabellenverzeichnis XI

Abbildungsverzeichnis XII

Listingsverzeichnis XIII

Literaturverzeichnis XIV

Anhang XVI
A Erläuterung einzelner Kenngrößen zur Bestimmung der Metriken XVI
A.1 Metriken 1.3.1 bis 1.3.3 – Anzahl Scenarios, Scenario-Aufrufe und TestSteps XVI
A.2 Metriken 2.3.4–2.3.6 und 2.3.7–2.3.9 – Metriken über die Komplexität der Scenarios XVI
A.3 Metrik 2.4.1 – Kommentaranteil XVII
A.4 Metrik 3.2.1 – Bewertung der Namensgebung der TestDesigns . . . XIX
A.5 Metrik 3.4.1 – Codeduplizierung XXI
B Implementierungsdetails XXII
B.1 Klassendiagramm des Reporting Framework (RepFram) XXII
B.2 Ablaufdiagramm/Sequenzdiagramm XXIV

Abkürzungsverzeichnis

ALM	HP Application Lifecycle Management [hier: Version ALM-11]
CRT	Kardiale Resynchronisationstherapie *(Cardiac Resynchronization Therapy)*
CSV	Comma-Separated Values
GQM	Goal Question Metric
HMSC	Home Monitoring Service Center
HMSC3-Val-Tests	Automatisierte Tests zur Validierung des HMSC
HTTP	Hypertext Transfer Protocol
ICD	Implantierbarer Defibrillator *(Implantable Cardioverter-Defibrillator)*
INI	Initialisierungsdatei
ISTQB	International Software Testing Qualifications Board
LOG	Ereignisprotokolldatei
QC	HP Quality Center [hier: Version QC-10]
RepFram	Reporting Framework
REST	Representational State Transfer
Sc.	Scenario
SCN	Scenario-Dateiendung
SUT	System Under Test
TCS	Testcase (Action-Dateiendung)
TMT	Test-Management-Tool
TSU	TestSetup-Dateiendung
URL	Uniform Resource Locator
XLS	Excel Spreadsheet
XML	Extensible Markup Language

1 Einleitung

1.1 Motivation

Mit dem Biotronik Home Monitoring® Service können Patienten mit implantierbaren Herzschrittmachern und Defibrillatoren (ICD) oder Systemen zur kardialen Resynchronisationstherapie (CRT) fernbetreut werden. Die Daten werden zeit- und ereignisgesteuert über eine im Implantat integrierte Antenne an ein mobilfunkfähiges Patientengerät – den CardioMessenger® – gesendet. Über diesen werden die Daten an das Home Monitoring Service Center (HMSC) weitergeleitet, wo sie vollautomatisch analysiert werden und dem behandelnden Arzt als Kardiobericht auf einer geschützten Internetseite zur Verfügung stehen [2].

Dieser Ablauf ist in Abbildung 1 grafisch dargestellt.

Abbildung 1: Home Monitoring [2]

Regulatorische Vorgaben erfordern das vollständige Testen aller Anforderungen an das Medizinprodukt HMSC. Dazu werden in einem Test-Management-Tool (TMT) logische Testfälle erstellt. Auf dieser Basis werden manuell konkrete Testfälle implementiert. Diese Testfälle können sowohl zur manuellen als auch automatischen Ausführung bestimmt sein. Das Ergebnis eines automatischen Testlaufs ist derzeit eine Datei (z.B. im Format XLS, XML, CSV). Alle Ergebnisse werden manuell bewertet und anschließend mit Hilfe des TMT – derzeit Quality Center (QC) – verwaltet und dokumentiert. Dieses Verfahren ist bei einer erhöhten Testanzahl durch die vermehrte Ausführung manueller Test- und Dokumentationsschritte fehleranfällig und zeitaufwändig. Des Weiteren ist durch die Erstellung der logischen Testfälle im Quality Center und die manuelle Erstellung der Testskripte keine Nachverfolgbarkeit der Anforderungen sichergestellt. Durch dieses Vorgehen sind Defizite in der Testqualität, insbesondere in den Bereichen der Testeffektivität und Wartbarkeit zu erwarten. Deshalb soll mit der Umstellung von QC (Version 10) auf die nächste Version (Version 11) bezeichnet mit HP Application Lifecycle Management (ALM) in diesen Bereichen die Testqualität verbessert werden.

1.2 Problemstellung

Die Arbeit wird die automatisierten Tests und deren Ergebnisse in Bezug auf die Testeffektivität und Wartbarkeit auf Qualitätsmängel untersuchen und Möglichkeiten zur Qualitätsverbesserung finden. Das Ziel der Arbeit ist es, insbesondere das Testen aller logischen Testfälle aus QC durch die Testskripte sicherzustellen und die Wartbarkeit der Tests zu verbessern.

Als Basis der weiteren Untersuchungen werden im ersten Schritt der Arbeit Begriffsklärungen und Definitionen erarbeitet. Auf dieser Grundlage wird anschließend die Herangehensweise beschrieben, nach der die Qualität in den Bereichen Testeffektivität und Wartbarkeit gemessen wird. Dies beinhaltet das Ermitteln quantitativer Metriken.

Im nächsten Schritt wird die aktuelle Qualität der automatisierten Tests untersucht. Es erfolgt eine statische Analyse der HMSC3-Val-Tests anhand der Metriken. Hierbei werden verschiedene Fragestellungen bewertet, wie etwa:

- „Testen die Testskripte genau die logischen Testfälle, die zuvor in QC erstellt wurden?“,
- „Wie wartbar sind die Tests?“ und
- „Wie werden die Tests bezüglich der Modularität, der Veränderbarkeit und des Aufwandes der Wartung beurteilt?“.

Anschließend wird ein Konzept zur Qualitätsverbesserung erarbeitet, welches die gefundenen Defizite lösen soll. Hierbei werden klare Zielvorgaben für die Verbesserung herausgearbeitet.

Im Implementierungsteil der Arbeit wird eine neue Musterstruktur für die Testspezifikationen und die Testskripte der automatisierten Tests entwickelt. Anschließend wird beispielhaft ein Tool entwickelt, welches anhand des neuen Musters nach jedem Testlauf automatisch die Ergebnisse der Testläufe nach ALM sendet. Durch dieses Tool und das zu Grunde liegende Muster wird die Nachverfolgbarkeit der Anforderungen werkzeugtechnisch sichergestellt. Sowohl die beispielhafte Implementierung der Musterstruktur als auch das entwickelte Tool sind auf der beiliegenden CD einzusehen.

Im letzten Schritt der Arbeit wird untersucht, ob die Zielvorgaben für die Qualitätsverbesserung erreicht wurden und es erfolgt eine kritische Bewertung der Arbeit.

2 Grundlagen

2.1 Begriffserklärungen

2.1.1 International Software Testing Qualifications Board (ISTQB)

Das International Software Testing Qualifications Board bietet eine standardisierte Ausbildung für professionelle Softwaretester. Die standardisierten Begriffe aus dem Glossar der ISTQB [9] werden in dieser Arbeit für ein einheitliches Verständnis zu Grunde gelegt. Die in Bezug auf diese Arbeit wichtigsten Begriffe aus dem Glossar [9] werden im Folgenden näher betrachtet.

Test

Ein Test ist eine Menge von einem oder mehreren Testfällen [5]. Der Test wird in einer **Testspezifikation** dokumentiert. Diese besteht aus der Testentwurfspezifikation, der Testfallspezifikation und/oder der Testablaufspezifikation. Die Testentwurfsspezifikation beschreibt die Testbedingungen, die detaillierte Testvorgehensweise und die logischen Testfälle.

Testziel/Testzweck

Das Testziel oder der Testzweck ist der Grund oder die Absicht zu dem ein Test entworfen und ausgeführt wird.

Testbare Anforderungen

Testbare Anforderungen sind Anforderungen, die so formuliert sind, dass Testbedingungen festgelegt werden können. Aus diesen Testbedingungen werden logische Testfälle formuliert. Bei der Durchführung der Testfälle lässt sich feststellen, ob die Anforderungen erfüllt sind (nach [4]).

Testbedingung

Eine Testbedingung kann ein Qualitätsmerkmal oder ein strukturelles Element einer Komponente oder eines Systems sein, welches durch einen oder mehrere Testfälle verifiziert wird.

Testfall

Ein Testfall umfasst folgende Angaben:

- die für die Ausführung notwendigen Vorbedingungen,
- die Menge der Eingabewerte (ein Eingabewert je Parameter des Testobjekts),
- die Menge der vorausgesagten Ergebnisse sowie

- die erwarteten Nachbedingungen.

Testfälle werden im Hinblick auf ein bestimmtes Ziel bzw. auf eine Testbedingung entwickelt.

Abstrakter Testfall
Ein abstrakter oder logischer Testfall ist ein Testfall ohne konkrete Ein- und Ausgabewerte für Eingabedaten und vorausgesagte Ergebnisse. Er verwendet logische Umschreibungen der Testabläufe, weil die konkreten noch nicht definiert oder verfügbar sind.

Konkreter Testfall
Ein konkreter Testfall ist ein Testfall mit konkreten Werten für Eingaben und vorausgesagte Ergebnisse. Logische Umschreibungen aus den abstrakten Testfällen werden durch konkrete Testabläufe ersetzt.

Testskript
Ein Testskript umfasst eine Folge von Schritten zur Testausführung. Die Dokumentation der Schritte wird üblicherweise als Testablaufspezifikation bezeichnet (nach [5]).

Teststep
Ein Teststep ist ein bestimmter Teil eines Testskripts. Dabei kann ein Teststep im Testskript genau einem abstrakten Testfall in der Testspezifikation entsprechen. Ein solcher TestStep würde konkreter Testfall heißen. Dennoch müssen nicht alle Teststeps notwendigerweise Testfälle sein. Sie können auch einfach die Bedingungen für den nächsten Teststep vorbereiten. Diese Begriffsbeschreibung wird aus [10] entnommen, da es in der ISTQB keine Definition des Teststeps gibt, diese für die Beschreibung dieser Arbeit jedoch benötigt wird.

2.1.2 Testqualität und Qualitätsmerkmale

An dieser Stelle werden die für diese Arbeit benötigten Begrifflichkeiten der Testqualität und deren Bedeutung im vorliegendem Zusammenhang erläutert[1].

Testqualität/Testgüte
Nach der IEEE 610 [4] ist Qualität der Grad, zu dem eine Komponente, ein System oder ein Prozess die formulierten Anforderungen und/oder die Kundenwünsche und -erwartungen erfüllt.
Da es keine Definitionen für die Testqualität gibt, gilt für diese Arbeit folgende hergeleitete Annahme: Testqualität ist der Grad, zu dem der Test die Anforderungen an den Test erfüllt.

[1] Alle Begrifflichkeiten stammen aus den referenzierten Quellen und wurden für diese Arbeit eigenhändig übersetzt.

Qualitätsmerkmal

Nach der IEEE 610 [4] ist ein Qualitätsmerkmal eine Fähigkeit oder Eigenschaft, welche die Qualität eines Elements beeinflusst. Übernimmt man diese Definition zusammen mit einer Defintion der ISO 9126 [11], dann sind Qualitätsmerkmale eine Menge von Eigenschaften eines Tests, anhand derer die Qualität der Tests beschrieben und beurteilt werden kann. Ein Qualitätsmerkmal kann über mehrere Stufen in Teilmerkmale verfeinert werden. Beispiele für Qualitätsmerkmale sind Testeffektivität und Wartbarkeit (vgl. [14], [13]). Die Qualität eines Tests wird erst durch die Strukturierung des Qualitätsbegriffes in Qualitätsmerkmale quantitativ messbar.

Testeffektivität

In [14] beschreiben die Autoren die Testeffektivität als die Fähigkeit des Tests, einen gegebenen Testzweck zu erfüllen. Testeffektivität kann mittels folgender Attribute beschrieben werden:

- **Testabdeckung:** Die Testabdeckung stellt ein Maß für die Testvollständigkeit dar und kann auf verschiedenen Ebenen gemessen werden [14]. Zur Verdeutlichung dieser Ebenen sind folgende zwei Fragestellungen möglich:
  - „Werden alle Anforderungen in der Testspezifikation berücksichtigt?“ und
  - „Werden alle Schritte der Testspezifikation mit dem Testskript implementiert?“.

  Für diese Arbeit ist vor allem der zweite Fall von Bedeutung.

- **Testkorrektheit:** Die Eigenschaft Testkorrekheit bezeichnet die Korrektheit der Testspezifikation in Bezug auf die Systemspezifikation oder den Testzweck. Des Weiteren ist eine Testablaufspezifikation nur korrekt, wenn sie immer korrekte Testergebnisse zurückgibt und wenn sie erreichbare Endzustände hat [14].

- **Fault-Revealing-Capability:** Die Fault-Revealing-Capability ist die Fähigkeit eines Tests Fehler aufzudecken. Je mehr Testfälle gefunden werden können oder je höher die Varianz der Testdaten ist, desto höher ist die Wahrscheinlichkeit, dass Fehler durch diese Tests gefunden werden [14].

Wartbarkeit

Die Wartbarkeit beschreibt den Grad, zu dem eine Testspezifikation oder ein Testskript geändert werden kann. Sie ist eine wichtige Eigenschaft, wenn Fehlerkorrekturen, Verbesserungen oder Anpassungen aufgrund von Änderungen der Umgebung oder Anforderungen notwendig werden. Zur Untersuchung der Wartbarkeit werden folgende Eigenschaften herangezogen [14]:

- **Analysierbarkeit:** Der Analysierbarkeitsaspekt befasst sich mit dem Grad, zu dem Mängel in einer Testspezifikation oder einem Testskript gefunden werden können. Zum Beispiel sollten Testspezifikationen und Testskripts klar strukturiert

sein, um eine schnelle Überprüfung zu ermöglichen. Testarchitektur, Styleguides, Dokumentation und generell klar strukturierter Code sind Elemente, die Einfluss auf den Grad dieser Eigenschaft haben [14].

- **Veränderbarkeit:** Die Eigenschaft Veränderbarkeit beschreibt den Grad, zu dem die Testspezifikation und das Testskript die Umsetzung notwendiger Änderungen ermöglichen. So haben sowohl klar strukturierter Code als auch leicht und weitreichend erweiterbare Architektur positiven Einfluss auf die Qualität der Veränderbarkeit [14].
- **Stabilität:** Die Stabilität ist ein Maß für die Wahrscheinlichkeit, dass bei einer Änderung unerwartete Ergebnisse auftreten. Je weniger Auswirkungen durch eine Änderung entstehen, desto besser ist die Qualität in diesem Bereich. Insbesondere haben globale Variablen schlechten Einfluss auf die Stabilität [14].

Einhaltung der Testeffektivität oder Wartbarkeit

Die Eigenschaften „Einhaltung der Testeffektivität“ oder auch „Einhaltung der Wartbarkeit“ sind weitere Untereigenschaften der Attribute Effektivität und Wartbarkeit. Diese Eigenschaften beschreiben jeweils den Grad, zu dem sich die Testspezifikation und das Testskript an bestehende Standards, Regeln oder Abmachungen halten. Die Standards betreffen jeweils die Testeffektivität oder die Wartbarkeit und sind meist abhängig vom Unternehmen oder Projekt [14]. In diesem Zusammenhang wird untersucht, ob es solche Regeln für die HMSC3-Val-Tests gibt und ob diese eingehalten werden. Des Weiteren wird in einem nächsten Schritt geprüft, ob die Erstellung solcher Regeln den Grad der Testeffektivität oder Wartbarkeit erhöhen können.

2.1.3 Weitere Begriffserklärungen

An dieser Stelle werden weitere Begriffe, Abkürzungen und Definitionen erläutert.

System-Test

System-Tests sind (Software-) Tests, die auf dem gesamten System durchgeführt werden, um die Einhaltung der spezifizierten Anforderungen für das System zu überprüfen [4]. Bei System-Tests benötigt der Tester kein Wissen über das innere Design, den Code oder die Logik des Systems. Bei diesen Tests wird neben dem Design auch das Verhalten des Systems und die angenommenen Erwartungen des Kunden getestet.

Regressionstest

Nach der IEEE 610 [4] ist der Regressionstest das gezielte wiederholende Testen eines Systems oder einer Komponente. Der Test soll sicherstellen, dass durch vorherige Änderungen keine unbeabsichtigten Auswirkungen entstanden sind und das System oder die Komponente immer noch seine spezifizierten Anforderungen erfüllt.

Tracebility/Tracing

Tracebility ist die Nachverfolgbarkeit der Anforderungen, sodass sichergestellt ist, dass die Anforderungen getestet werden [12]. Durch die Nachverfolgung (Tracing) ist sowohl eine Umsetzung der Anforderungen in logischen Testfällen in der Testspezifikation sichergestellt als auch die Umsetzung der logischen Testfälle zu einzelnen Teststeps. Durch ein konsequentes Tracing der Anforderungen kann die Testabdeckung nachgewiesen oder bei fehlender Testabdeckung Verbesserungspotential erkannt werden (vgl. Abschnitt 2.1.2 auf Seite 5).

Representational State Transfer (REST)

Das REST-Prinzip steht für einen Programmierstil für Webanwendungen. Dabei stellt eine Uniform Resource Locator (URL) genau einen Seiteninhalt als Ergebnis einer Operation auf dem Server dar. REST-konforme Dienste bieten insbesondere die Operationen GET, POST, PUT und DELETE an, die ein Verändern aller Ressourcen des Dienstes ermöglichen (vgl. [8]).

Goal Question Metric (GQM)

Die Goal Question Metric ist eine Methode mit Top-Down-Ansatz, um Metriken zu finden. Durch systematische Vorgehensweise wird als Erstes das Ziel der Messung aufgestellt. Aus diesem Ziel werden Fragen abgeleitet. Die Antworten auf diese Fragen liefern eine Bewertung für die Erreichung des zuvor aufgestellten Ziels. Anschließend werden für jede Frage Metriken festgelegt. Diese Metriken tragen zur Beantwortung der Fragen bei. Weitere Schritte sind die Messung und Datensammlung der festgelegten Maße, die Validierung der Messwerte und die Interpretation der Messergebnisse [7].

Metrik

Nach der IEEE 610 [4] ist eine Metrik ein quantitatives Maß für den Grad, zu dem ein System, eine Komponente oder ein Prozess ein gegebenes Qualitätsmerkmal besitzt. Auch der Begriff der Qualitätsmetrik wird in diesem Zusammenhang verwendet.

Statische Analyse

Die statische Analyse (vgl. [12]) umfasst die Überprüfung von Dokumenten und ausführbaren Dateien ohne Ausführung der Prüfobjekte mit folgenden Zielen:

- Die Aufdeckung vorhandener Fehler oder fehlerträchtiger Stellen in einem Dokument, in diesem Fall den Testspezifikationen und Testskripten.
- Die Ermittlung von Messgrößen oder Metriken, um eine Qualitätsbewertung durchführen und somit Qualität messen und nachweisen zu können.

In dieser Arbeit wird insbesondere auf die Erfüllung des zweiten Ziels eingegangen, um quantitative Aussagen über die Qualität der betrachteten Dokumente zu gewinnen. Die Analyse wird meist durch Werkzeuge automatisiert vorgenommen. Da die Anzahl und Größe der zu untersuchenden Objekte überschaubar ist, kann in dieser Arbeit die statische Analyse teilweise ohne Werkzeugunterstützung durchgeführt werden.

Modularisierung, Modul und Modularität
Die Modularisierung ist die Gliederung eines Programms – hier der Testskripte – in einzelne wiederverwendbare Bausteine/Module. Jeder Baustein ist ein in sich geschlossener Teil des Gesamtsystems (hier der Testskripte). Ein gut modularisiertes System ist einfach zu verstehen und Änderungen beschränken sich meist auf ein Modul (vgl. [3]).

Das IEEE-Glossar der Software Engineering Terminologie (IEEE 610 [4]) definiert Modularität als den Grad, zu dem ein System oder ein Computerprogramm aus separaten Komponenten besteht, sodass eine Änderung an einer Komponente minimale Auswirkungen auf andere Komponenten hat.

2.2 Systemlandschaft

Die Testlandschaft der HMSC-Validierungsgruppe besteht aus einem System Under Test (SUT), einem Test-Tool und TMT. Das SUT wird durch die automatisierten Tests zur Validierung des HMSC – im Folgenden HMSC3-Val-Tests – getestet. Die Testläufe der HMSC3-Val-Tests werden durch das Test-Tool ausgeführt und anschließend im TMT verwaltet und bewertet.

2.2.1 System under Test – HMSC

Das HMSC ist eine zentrale Serveranwendung, welche die Daten der implantierten Geräte empfängt. Im HMSC werden die Daten vollautomatisch analysiert und dem behandelnden Arzt in übersichtlicher Form als Kardiobericht auf einer geschützten Internetseite zur Verfügung gestellt.

Diese geschützte Internetseite, auch Plattform des HMSC genannt, ist der von den automatisierten Tests betrachtete Teil des HMSC. Die HMSC-Validierungsgruppe prüft das Gesamtsystem mittels anwendungsfallbasierter Tests.

2.2.2 Test-Tool – JavaTT und HMSC3-Val-Tests

Die HMSC3-Val-Tests werden durch das Test-Tool JavaTT ausgeführt. Im Folgenden wird die Gruppe der HMSC3-Val-Tests vorgestellt. Im Anschluss wird der Aufbau der Tests, wie er durch das Test-Tool vorgeschrieben ist, beschrieben.

2.2.2.1 HMSC3-Val-Tests

Es handelt sich bei den HMSC3-Val-Tests um System-Tests, wie sie in der Validierungsabteilung für das HMSC durchgeführt werden.

- Die Regressionstests bestehen momentan aus elf Testabläufen. In jedem Testablauf werden die Hauptfunktionalitäten der Plattform des HMSC3 geprüft, wie z.B.

Benutzerverwaltung und Datenexport. Insbesondere sollen Auswirkungen von Änderungen, die durch Wartung oder Korrektur entstanden sind, in bereits getesteten Teilen des HMSC aufgedeckt werden.

- Die Gruppe der Legacytests testet neben der Plattformfunktionalität die Plugins für Implantate des HMSC3. Diese Plugins ermöglichen die Verarbeitung von Nachrichten der jeweiligen Implantate an das HMSC3. Hierfür muss die korrekte Behandlung der Nachrichten geprüft werden. Für jedes neue Implantatsplugin wird ein Systemtest mit Nachrichten echter Implantate in Echtzeit durchgeführt. Für bereits integrierte Plugins, die sich geändert haben, sind wiederum Regressionstests erforderlich.

- Die dritte Gruppe besteht aus einem einzelnen Test, welcher die kontextsensitive Online-Hilfe des HMSC auf die regulativ relevanten (Sicherheits-)Hinweise überprüft.

2.2.2.2 JavaTT

JavaTT [6] ist ein Test-Tool für Java-Anwendungen. Es liest Testdaten aus einer Reihe von XML-Dateien und verteilt sie an einen oder mehrere Zielcomputer, welche die Testdaten ausführen. Die Ergebnisse werden zurückgesendet, um eine Ergebnisdatei vom Typ XLS, XML oder CSV zu generieren.

Im Listing 1 ist ein Testskript aus der Gruppe der Regressionstests abgebildet, welches den Aufbau der Testskripte verdeutlichen soll.

```
1  <TestSetup Name="AckPost">
2    <SetupEntry>
3      <!-- Login -->
4      <Scenario Name="SCN_HMSC3_Platform/Login">
5        <!-- Parameters which are necessary for the login -->
6        <Parameter Name="UserGroup"     Value="System"/>
7        <Parameter Name="UserName"      Value="PreviewCSS"/>
8        <Parameter Name="Password"      Value="test12"/>
9        <Parameter Name="url" Value="www.testsystem-homemonitoring.int"/>
10     </Scenario>
11     <!-- Deactivate Monitoring -->
12     <Scenario Name="SCN_PatientAdministration/PA_DeactivateMonitoring">
13       <Parameter Name="Patient" Value="Val-60400025" />
14     </Scenario>
15   </SetupEntry>
16 </TestSetup
```

Listing 1: Ausschnitt eines Testskripts

Das oberste Element in einem JavaTT-Skript ist das **TestSetup**. Das TestSetup enthält den gesamten Testablauf und wird als TSU-Datei abgespeichert. Außerdem werden aus dieser Datei die **Scenarios** aufgerufen. Jedes Scenario ist in einer weiteren Datei im Format SCN abgespeichert. Der Aufbau eines Scenarios ist in Listing 2 abgebildet.

```
1  <Scenario Name="PA_DeactivateMonitoring">
2    <TestCase Name="Click Menu All Patient">
3      <FollowLink Name="Click menu 'All patient'" LinkText="All patient"/>
4      <!-- Check if we can see all patient -->
5      <HTMLTagVerificationPoint Name="Check if step is executed correctly">
6        <HTMLTag Name="input">
7          <HTMLTagAttribute Name="value" Value="Extended view"/>
8        </HTMLTag>
9        <Verifier Name="Exists" Type="java.lang.Boolean"  Value="true"/>
10     </HTMLTagVerificationPoint>
11   </TestCase>
12   <TestCase Name="Actions/Act_Select_Patient"/>
13   <TestCase Name="Deactivate Home Monitoring">
14     <FollowLink Name="Click 'Patient profile'" LinkText="Patient profile"/>
15     ...
16   </TestCase>
17 </Scenario>
```

Listing 2: Ausschnitt eines Scenarios

Das Scenario enthält einen bis beliebig viele **TestCases**. Anstatt eines TestCases können im Scenario auch beliebig viele **Actions** aufgerufen werden. Ein solcher Aufruf ist im Listing 2 in Zeile 12 bei der Action „Action/Act_Select_Patient“ zu sehen (Aufruf der Action im TestCase). Die Action[2] ist wiederum in einer separaten Datei vom Typ TCS abgespeichert und kann nur einen TestCase enthalten.

In jedem TestCase gibt es verschiedene **TestItems**. Jedes TestItem entspricht einer ausgeführten Operation und führt jeweils zu einem Ergebnis. Ein mögliches TestItem ist der **VerificationPoint**. Der VerificationPoint stellt einen Punkt dar, an dem Testinformationen oder Zustände verifiziert werden.

Zusammenfassend kann ein **Testskript** als eine Reihe von XML-Dateien – TestSetup, Scenarios und Actions – definiert werden.

Alle soeben beschriebenen Elemente lassen sich in der Ergebnisdatei, die durch JavaTT generiert wird, wiederfinden.

[2] Auf die Abbildung der Action wird an dieser Stelle verzichtet, da sie im Aufbau einem TestCase im Scenario entspricht.

Tabelle 1: Ausschnitt aus der Ergebnisdatei des TestSetups „AckPost“

Scenario Name	TestCase Name	TestStep Name	Result Value	Result Message
Login	Actions/ Act_Fill_Login_CSS	Load start URL	INFO	Loaded URL https://www.testsystem-homemonitoring.int/
Login	Actions/ Act_Fill_Login_CSS	Type 'User group'	INFO	FormParameter set: SignInForm:userGroup=[System]
Login	Actions/ Act_Fill_Login_CSS	Type 'User name'	INFO	FormParameter set: SignInForm:userName=[PreviewCSS]
Login	Actions/ Act_Fill_Login_CSS	Type 'Password'	INFO	FormParameter set: SignInForm:password=[test12]
Login	Actions/ Act_Fill_Login_CSS	Click button 'Sign in'	INFO	Sent request InputName=SignInForm:signIn InputValue=null.
Login	Actions/ Act_Fill_Login_CSS	Check for correct login into HMSCIII as 'System', 'PreviewCSS'	FINE	Verification point Check for correct login into HMSCIII as 'System', 'PreviewCSS' passed. All properties are as they should be.
PA_ DeactivateMonitoring	Click Menu All Patient	Click menu 'All patient'	INFO	Clicked link text='All patient' and URL='null'
PA_ DeactivateMonitoring	Click Menu All Patient	Check if step is executed correctly	FINE	Verification point Check if step is executed correctly passed. All properties are as they should be.
PA_ DeactivateMonitoring	Actions/ Act_Select_Patient	Input search-string 'Val-60400025'	INFO	FormParameter set: PatientsOverview: searchString=[Val-60400025]
PA_ DeactivateMonitoring	Actions/ Act_Select_Patient	Select 'All monitoring statuses'	ERROR	Form nr. 0 doesn't exist.
PA_ DeactivateMonitoring	Actions/ Act_Select_Patient	Click button 'Search'	INFO	Sent request InputName=PatientsOverview: submitSearchButton InputValue=null.
PA_ DeactivateMonitoring	Actions/ Act_Select_Patient	Click patient 'Val-60400025'	INFO	Clicked link text='Val-60400025' and URL='null'
PA_ DeactivateMonitoring	Actions/ Act_Select_Patient	Wait	INFO	Waited 2000 milliseconds.
PA_ DeactivateMonitoring	Actions/ Act_Select_Patient	Check if Patient 'Val-60400025' is select	FINE	Verification point Check if Patient 'Val-60400025' is select passed. All properties are as they should be.
PA_ DeactivateMonitoring	Deactivate Home Monitoring	Click 'Patient profile'	INFO	Clicked link text='Patient profile' and URL='null'

Die Tabelle 1 zeigt einen Ausschnitt aus der Ergebnisdatei, in der alle Scenarios aus dem Listing 1 mit ihren Ergebnissen entnommen werden können. Jede ausgeführte Operation (TestItem) in einem Scenario führt, wie in der Ergebnisdatei zu erkennen ist, zu einem Ergebnis. Sowohl bei einer manuellen als auch automatischen Auswertung der Ergebnisse ist darauf zu achten, dass ein Testlauf nur mit „Passed“, das heißt ohne Beanstandungen, gewertet werden kann, wenn unter den Ergebnissen keine Meldung vom Typ „Error“ oder „Warning“ war.

2.2.3 Testmanagementtool – QC/ALM

Das Quality Center bzw. Application Lifecycle Management ist ein Test-Management-Tool zur Verwaltung und Organisation von Tests, wobei ALM die Nachfolgeversion von QC ist[3]. Im weiteren Verlauf der Arbeit wird während der theoretischen Betrachtungen weiterhin der Name QC verwendet. Erst ab Kapitel 4 wird von den Umsetzungen in ALM die Rede sein.

[3] QC /ALM kann auch zur Steuerung der Ausführung von Tests verwendet werden. Diese Funktion wird aber im vorliegenden Kontext nicht genutzt.

In QC gibt es mehrere Strukturebenen, genannt **Module**, die das Verwalten der Tests ermöglichen. Zwei dieser Ebenen sind für diese Arbeit von Bedeutung und werden im Folgenden mit ihren enthalten Objekten erläutert.

In der ersten Strukturebene in QC, dem **TestDesign Module**, liegt das **TestDesign**. Das TestDesign dient zur Spezifikation des Tests im Quality Center. Es besteht aus mehreren **TestSteps**, die neben den Angaben zur Testausrüstung detaillierte Anweisungen zur Testausführung enthalten. Der TestStep kann die auf der Anwendung auszuführenden Aktionen, die Eingabewerte oder die erwarteten Ausgaben enthalten. Weiterhin können an das TestDesign durch den Test abzudeckende Anforderungen angehängt werden.

Die zweite wichtige Strukturebene im QC ist das **TestLab Module**. In dieser Ebene liegt das **TestSet** zur Planung von Testläufen. Die TestSets sind hierarchisch in einer Ordnerstruktur, den **TestSetFoldern**, angeordnet. Jedes TestSet enthält eine Menge von Testinstanzen (**TestInstance**). Eine TestInstance gehört immer zu genau einem TestDesign. Dagegen können aus jedem TestDesign mehrere Testinstanzen gebildet werden. Für jede TestInstance gibt es einen verantwortlichen Tester. Dieser verantwortliche Tester erzeugt einen **TestRun** in der TestInstance. Durch die Verknüpfung der TestInstance mit dem TestDesign finden sich im TestRun die TestSteps aus dem TestDesign als **RunStep** wieder. Jeder RunStep kann ein Ergebnis haben.

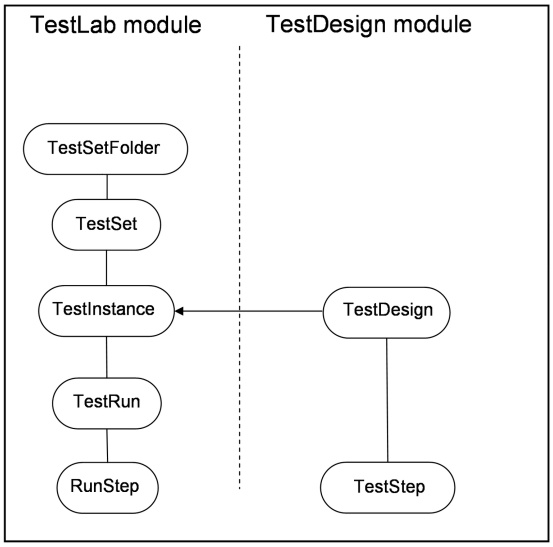

Abbildung 2: Übersicht der Objekte aus QC

Nachfolgend ist ein TestSet mit der TestInstance „Ack_Postpone“ abgebildet. Wie zuvor beschrieben, liegt das TestSet in einem TestSetFolder. Die TestInstance ist mit dem gleichnamigen TestDesign verknüpft.

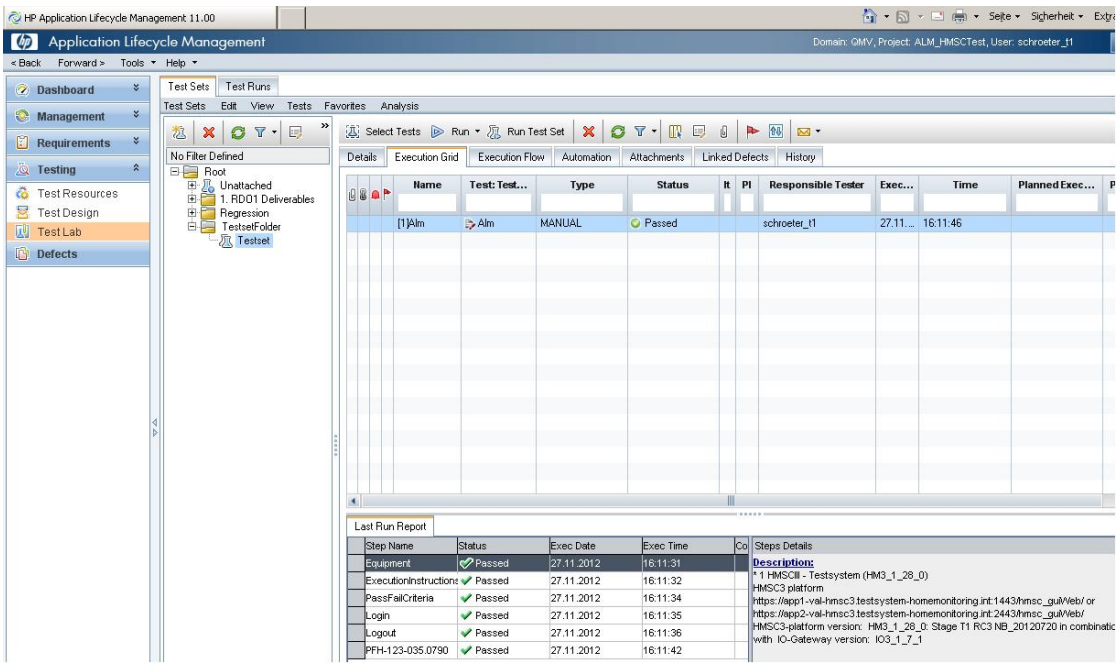

Abbildung 3: Ausschnitt aus ALM

2.3 Begriffsgegenüberstellungen

Die wichtigsten Elemente aus ISTQB, JavaTT und QC werden nun in einen direkten Zusammenhang gebracht und grafisch veranschaulicht. Diese Gegenüberstellung dient in den folgenden Kapiteln als Grundlage, auf deren Basis auch die Metriken ermittelt werden.

In der Abbildung 4 ist die Grafik so aufgebaut, dass Elemente, die sich inhaltlich auf einer logischen Ebene befinden, auch grafisch in einer Ebene dargestellt sind. So gibt es eine logische Ebene zwischen Testspezifikation, Testskript, TestSetup, TestInstance sowie TestDesign. Das TestDesign entspricht nach der Definition einer Testspezifikation und das TestSetup entspricht einem Testskript.

Ein weiterer Zusammenhang besteht zwischen den logischen und konkreten Testfällen sowie dem TestStep der Definition, dem Scenario aus JavaTT und dem RunStep und TestStep aus QC. Für diese Arbeit wird angenommen, dass der Umfang eines Scenarios in JavaTT einem TestStep in QC entspricht. Dabei deckt sich ein TestSteps in QC mit einem logischen Testfall und die Scenarios in JavaTT mit einem konkreten Testfall. Diese Annahme ist derzeit nicht sichergestellt, wird jedoch für spätere Zielvorgaben so vorausgesetzt.

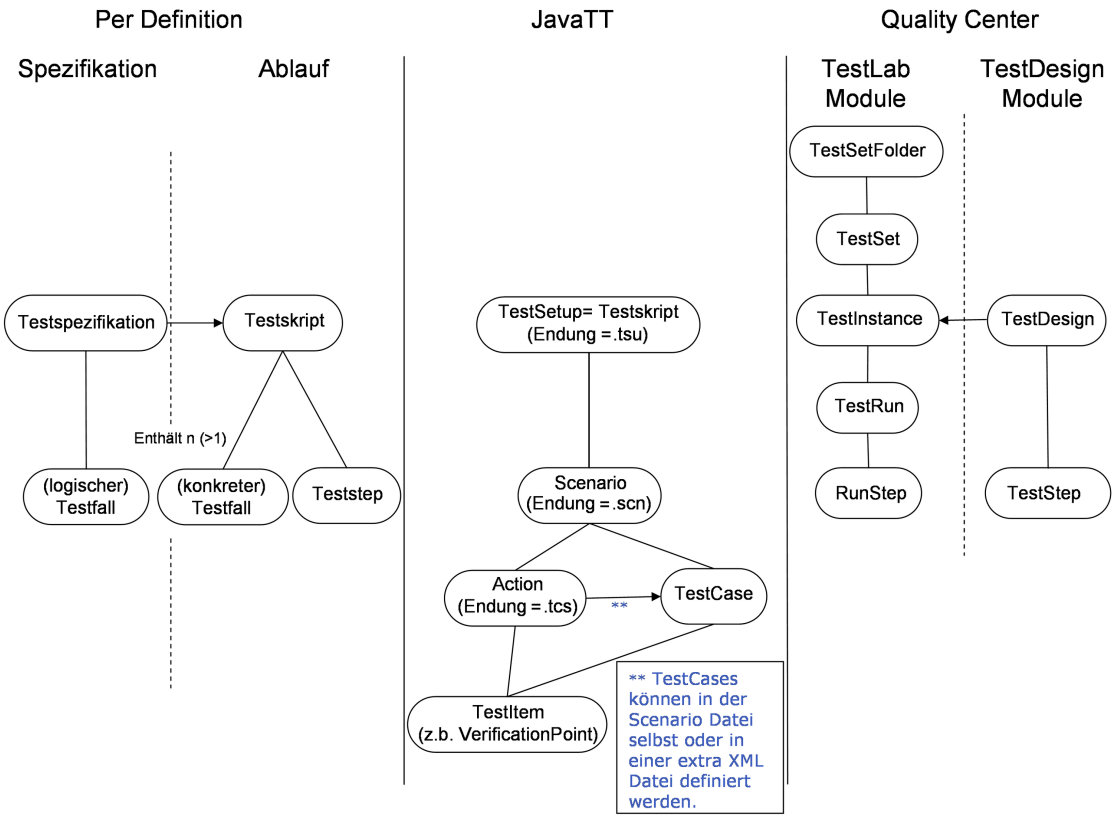

Abbildung 4: Gegenüberstellung der Begriffe in ISTQB, JavaTT und QC

3 Statische Analyse

In diesem Kapitel wird die Testqualität hinsichtlich Testeffektivität und Wartbarkeit untersucht. Dazu werden die Testskripte und TestDesigns der HMSC3-Val-Tests betrachtet.

Dieses Kapitel besteht aus zwei Abschnitten, in denen sich mit den beiden Qualitätsmerkmalen befasst wird. Sowohl für die Testeffektivität als auch die Wartbarkeit werden zunächst diverse Metriken mittels GQM ermittelt. Anschließend werden alle gefundenen Metriken auf die Regressionstests angewendet, die einen großen Teil der HMSC3-Val-Tests ausmachen. Es wird erwartet, dass dabei repräsentative Ergebnisse erzielt werden, da bei den Regressionstests fast alle Schritte automatisiert mit JavaTT ablaufen. Auf die Auswertung der Legacytests und den Onlinehilfetest wird wegen des hohen Anteils manueller Ausführungsschritte an dieser Stelle verzichtet. Abschließend erfolgt eine Auswertung der Ergebnisse, bei der vorhandene Defizite der HMSC3-Val-Tests aufgedeckt werden.

3.1 Ermittlung der Metriken mit der Goal Question Metric

Für die Bestimmung aller Metriken in diesem Abschnitt wird das Verfahren der Goal Question Metric angewendet (Begriffserklärung und Vorgehen im Abschnitt 2.1.3 auf Seite 7).

Der Blickwinkel bei der Bildung der Ziele geht stets von einer rein technischen Perspektive aus. Die Fragestellungen sind daher aus der Sicht des Testers zu bewerten. Bei der Aufstellung der Metriken werden vorrangig quantitative Metriken berücksichtigt. Diese ermöglichen bei der nachfolgenden Auswertung eine bessere Vergleichbarkeit und Verarbeitung der Daten. Da die betrachtete Datenmenge sehr klein ist, können die Ergebnisse dieser Analyse jedoch nicht auf andere Tests verallgemeinert werden. Auf die Erstellung qualitativer Metriken wird in dieser Arbeit trotz des Vorteils der Anwendung bei kleineren Stichproben weitestgehend verzichtet, da sie sehr zeitaufwändig und schwerer auszuwerten sind. Dabei spricht vor allem die schlechte Vergleichbarkeit der Ergebnisse gegen eine Verwendung qualitativer Metriken.

Die Metriken sind so bezeichnet, dass das jeweilige Ziel und die gestellte Frage erkennbar sind. Die Nummerierung erfolgt dazu nach dem Schema „Ziel.Frage.Metrik“ – so ist die Metrik 1.1.1 die erste Metrik, zur ersten Frage des ersten Ziels.

3.1.1 Testeffektivität

Das Qualitätsmerkmal Testeffektivität wird laut dem Testqualitätsmodell im Wesentlichen durch folgende Teilmerkmale näher bestimmt: Testkorrektheit, Fault-Revealing-Capability und Testabdeckung.

Die Testkorrektheit – Korrektheit der Systemspezifikation – wird in dieser Arbeit nicht betrachtet, da der Fokus der Arbeit nicht auf diesem Merkmal liegt. Die Korrektheit der Testablaufspezifikation (Testskript) hinsichtlich gültiger Endzustände kann bei den vorliegenden Tests angenommen werden. Diese Annahme ist möglich, da in JavaTT derzeit nur sequentielle Abläufe umsetzbar sind und es keine Verzweigungen geben kann.

Für die Fault-Revealing-Capability scheint die Varianz der Testdaten von Bedeutung zu sein. Auf die Untersuchung dieser Eigenschaft wird hier jedoch verzichtet, weil die HMSC-Validierungsgruppe System-Tests ausführt, bei denen der Varianz weniger Bedeutung beigemessen werden kann. Zudem wird die Variabilität bereits auf niedrigeren Testebenen in anderen Testgruppen berücksichtigt.

Das Hauptaugenmerk in dieser Arbeit wird auf die Testabdeckung gelegt, da hier insbesondere durch den Bruch zwischen der Erstellung logischer Testfälle und der Testskripterstellung ein Qualitätsdefizit erwartet wird.

Die Testabdeckung des TestDesigns zum TestSetup für die HMSC3-Val-Tests soll sichergestellt sein (Ziel 1).

Um dieses Ziel zu erreichen, muss es eine direkte Zuordnung vom TestDesign zum TestSetup geben. Es wird zunächst ermittelt, ob eine solche Zuordnung existiert und ob es Bereiche des TestDesigns gibt, die nicht durch das TestSetup abgedeckt sind. Durch die Beantwortung folgender Fragen kann eine existierende Zuordnung und die momentane Testabdeckung bewertet werden:

- ***Frage 1:*** Wie ist das Verhältnis der Scenarios im TestSetup zu den TestSteps im TestDesign?
- ***Frage 2:*** Gibt es im TestDesign spezifizierte TestSteps, die nicht als Scenario im TestSetup implementiert wurden?
- ***Frage 3:*** Gibt es Unterschiede in der Anzahl der TestSteps im TestDesign zur Anzahl der Scenarios sowie Scenario-Aufrufe im TestSetup?

Die erste Frage bezieht sich auf eine angenommene Zuordnung zwischen TestSteps im TestDesign und Scenarios (bei Bedarf abgekürzt mit Sc.) im TestSetup. Eine bestehende Zuordnung erleichtert die Feststellung der Testabdeckung des TestDesigns zum TestSetup und kann durch die ersten beiden Metriken beantwortet werden. Die zweite Frage bezieht sich auf die bestehende Testabdeckung und wird durch

zwei Metriken betrachtet. Die dritte Frage deckt die Unterschiede in der Anzahl der genannten Elemente auf. Zur Beantwortung dieser Frage werden drei Metriken aufgestellt, die auch zur Beantwortung der ersten Frage beitragen.

Zur Beantwortung der Fragen werden folgende Metriken festgelegt:

Metrik 1.1.1

$$\textit{Verhältnis Scenarios : TestSteps} := \frac{\textit{Anzahl der Scenarios}}{\textit{Anzahl der TestSteps}} \tag{1.1.1}$$

Diese Metrik zeigt das Verhältnis zwischen den Scenarios im TestSetup und den TestSteps im TestDesign. Bei der Anzahl der Scenarios handelt es sich um die Menge der Scenarios eines TestSetups, bei der Mehrfachaufrufe nicht betrachtet werden. Die Idee hinter dieser Metrik ist, dass Inhalt und Umfang eines Scenarios dem eines TestSteps entsprechen sollte. Da ein TestStep in QC nach der Definition einem abstrakten Testfall entspricht, gibt es keine Mehrfachaufrufe der TestSteps. Deshalb wird für das Verhältnis von Scenario zu TestStep wie oben beschrieben auch nur die Menge der Scenarios in der Metrik verwendet.

Wenn jeder TestStep genau durch ein Scenario umgesetzt wird, dann liegt der Wert der Metrik bei „1“. Liegt der Wert kleiner „1“, dann gibt es TestSteps, die nicht durch ein Scenario umgesetzt sind. Ein Wert kleiner „1“ würde damit auf fehlende Testabdeckung hinweisen. Liegt der Wert größer „1“, ist dies ein Hinweis für schlechtere Wartbarkeit und nicht für schlechte Testabdeckung[4] (mehr dazu im Abschnitt Wartbarkeit).

Metrik 1.1.2

$$\textit{Verhältnis Scenario-Aufrufe : TestSteps} := \frac{\textit{Anzahl aller Scenario-Aufrufe}}{\textit{Anzahl der TestSteps}} \tag{1.1.2}$$

Da für die Metrik 1.1.1 vorerst kein optimaler Wert von „1“ erwartet wird, dient das Verhältnis der Scenario-Aufrufe zu den TestSteps (Metrik 1.1.2) als ergänzende Metrik. Zudem ist die Anzahl der Scenario-Aufrufe zu berücksichtigen, wenn die Parametrisierung von Scenarios in JavaTT genutzt wird. Durch diese Parametrisierung ist es möglich, einen logischen Testfall (entspricht etwa einem TestStep in QC) durch mehrere konkrete Testfälle zu implementieren. Dabei würde jeder Scenario-Aufruf mit unterschiedlichen Parametern einem konkreten Testfall entsprechen.

[4] Es gibt Scenarios, für die zuvor kein TestStep spezifiziert wurde. Dadurch sind sehr komplexe TestSetups möglich. Dies kann auf schlechtere Verständlichkeit und damit schlechtere Wartbarkeit hinweisen.

Metrik 1.2.1. und 1.2.2:

Anzahl nicht implementierter TestSteps (1.2.1)

Anteil nicht implementierter TestSteps := (1.2.2)

$$\frac{\textit{Anzahl nicht implementierter TestSteps}}{\textit{Gesamtanzahl der TestSteps}} * 100$$

Die Metriken 1.2.1 und 1.2.2 betrachten die Anzahl und den Anteil (in %) der nicht implementierten TestSteps. Ist der Wert dieser Metriken hoch, ist dies ein Hinweis auf eine schlechte Testabdeckung. Da viele der TestSteps bestimmte Anforderungen testen, ist es wichtig, dass der Wert dieser Metriken gegen „0“ geht. Beide Metriken werden zum jetzigen Zeitpunkt durch inhaltliche Analyse der TestDesigns und TestSetups gemessen.

Metrik 1.3.1, 1.3.2 und 1.3.3.:

Anzahl der Scenarios im TestSetup (1.3.1)

Anzahl aller Scenario-Aufrufe im TestSetup (1.3.2)

Anzahl der TestSteps im TestDesign (1.3.3)

Um die vorigen Metriken – insbesondere die Metriken 1.1.1 und 1.1.2 – besser zu deuten, ist es hilfreich, die Anzahl der Scenarios (ohne Mehrfachaufruf) und die Anzahl der Scenario-Aufrufe in den TestSetups sowie die Anzahl der TestSteps in den TestDesigns zu kennen.

3.1.2 Wartbarkeit

Das Qualitätsmerkmal Wartbarkeit wird laut dem Testqualitätsmodell durch folgende Teilmerkmale näher bestimmt: Analysierbarkeit, Änderbarkeit und Stabilität. Auf die Untersuchung der Stabilitäts-Eigenschaft wird verzichtet, da keine globalen Variablen in den TestSetups verwendet werden und keine globalen Beziehungen zwischen den TestSetups bestehen. Damit sind keine Nebenwirkungen durch Änderungen an den TestSetups zu erwarten.

Die Analysierbarkeit der TestDesigns und TestSetups der HMSC3-Val-Tests soll verbessert werden (Ziel 2).

Um dieses Ziel zu erreichen, muss die bestehende Analysierbarkeit zunächst festgestellt werden. Dabei sind die Lesbarkeit und die Verständlichkeit der TestDesigns und TestSetups zu berücksichtigen. Die Verwendung von Kommentaren und wenig komplexe Testskripte ermöglichen in diesem Zusammenhang eine gute Verständlichkeit der

Testskripte. Durch die Beantwortung folgender Fragen kann die Analysierbarkeit bewertet werden:

- ***Frage 1:*** Ist das TestDesign gut lesbar?
- ***Frage 2:*** Ist der Ablauf im TestSetup anhand des TestDesigns erkennbar?
- ***Frage 3:*** Ermöglichen die TestSetups und alle enthaltenen Elemente durch ihre Struktur ein schnelles Verständnis des Aufbaus und der Inhalte der Testskripte?
- ***Frage 4:*** Ist der Code gut dokumentiert?

Die erste Frage beantwortet durch die Betrachtung der Lesbarkeit der TestDesigns auch die Analysierbarkeit. Die zweite Frage macht eine Aussage über die Verständlichkeit der TestDesigns und TestSetups. So ist ein schnelles Verstehen der Inhalte der TestSetups möglich, wenn der Ablauf im TestSetup mit dem im TestDesign übereinstimmt oder wenn im TestSetup Referenzen auf das TestDesign vorhanden sind (denkbar wäre eine Referenz durch Übereinstimmung der Namen des Scenarios und TestSteps). Bei der dritten Frage wird der Fokus insbesondere auf die Komplexität, den Umfang und den Aufbau der enthaltenen Elemente gelegt, um den Grad der Verständlichkeit zu quantifizieren. Die letzte Frage betrachtet die Dokumentation des Quellcodes. Diese wirkt sich ebenfalls auf die Verständlichkeit und bessere Lesbarkeit der Testskripte aus.

Zur Beantwortung der Fragen werden folgende Metriken festgelegt:

Metrik 2.1.1:

$$\textit{Fehlerquotient (FQ)} := \frac{\textit{Anzahl Rechtschreibfehler}}{\textit{Wortzahl des TestDesigns}} * 100 \tag{2.1.1}$$

Je weniger Rechtschreibfehler in einem Dokument enthalten sind, desto lesbarer und verständlicher ist es. Die Metrik ist vor allem im Hinblick auf die unterschiedlichen Englischkenntnisse der Tester relevant.

Tabelle 2: Bewertung des Fehlerquotienten

Note	15 (1^+)	14 (1)	13 (1^-)	12 (2^+)	11 (2)	10 (2^-)	9 (3^+)	8 (3)	7 (3^-)	6 (4^+)	< 5 (4)
FQ	0,0-0,3	0,4-0,7	0,8-1,0	1,1-1,3	1,4-1,7	1,8-2,0	2,1-2,3	2,4-2,7	2,8-3,0	3,1-3,3	> 3,4

Der Fehlerquotient berechnet sich aus der Anzahl der Rechtschreibfehler und der Zahl der Wörter im TestDesign. Zur Bestimmung dieser Werte werden die TestDesigns aus QC in ein Word-Dokument extrahiert, um die dortige Rechtschreibprüfung anzuwenden. Als Rechtschreibfehler werden alle von Word erkannten Rechtschreibfehler gezählt, ausgenommen sind Eigennamen oder technische Angaben. In der Metrik wird der Fehlerquotient angegeben. Die Bewertung erfolgt anschließend nach Abiturmaßstab [1] mit Schulnoten (siehe Tabelle 2).

Metrik 2.2.1. und 2.2.2:

$$\textit{Anzahl unspezifizierter Scenarios} \tag{2.2.1}$$

$$\textit{Anteil unspezifizierter Scenarios} := \frac{\textit{Anzahl unspezifizierter Scenarios}}{\textit{Gesamtanzahl der Scenarios}} * 100 \tag{2.2.2}$$

Die Metriken 2.2.1 und 2.2.2 betrachten die Anzahl und den Anteil (in %) unspezifizierter Scenarios. Ist der Wert dieser Metriken hoch, ist dies ein Hinweis auf eine schlechte Analysierbarkeit. Der Testablauf im TestSetup könnte in diesem Fall nicht anhand des TestDesigns erkannt werden. Hinzu kommt, dass ein TestSetup mit vielen unspezifizierten Scenarios unnötig komplex ist. Deshalb sollten nach Möglichkeit nur Scenarios getestet werden, die zuvor durch einen TestStep im TestDesign spezifiziert wurden.

Weitere Metriken über den Umfang und die Komplexität eines Scenarios:

min, max, mittlere Anzahl der Actions pro Scenario (2.3.1–2.3.3)

min, max, mittlere Anzahl der TestCases pro Scenario (2.3.4–2.3.6)

min, max, mittlere Anzahl der TestItems pro Scenario (2.3.7–2.3.9)

Diese Metriken stellen den Umfang und die Komplexität der Scenarios, die von dem jeweiligen TestSetup aufgerufen werden, dar. Nach Möglichkeit sollte ein Scenario viele Actions aufrufen, um TestCases auszulagern. Außerdem dürfen wegen der hohen Komplexität nicht zu viele TestCases innerhalb eines Scenarios vorhanden sein. Aber auch zu wenige TestCases können die Komplexität der Scenarios erhöhen. Dieser Fall tritt ein, wenn ein TestCase zu viele einzelne Schritte enthält. Um dies beurteilen zu können, ist die Anzahl der TestItems zu betrachten. Auch deren Anzahl sollte zur Begrenzung der Komplexität und zum besseren Verständnis der Scenarios beschränkt sein. Für die mittlere Anzahl der jeweiligen Größen wird der Median bestimmt, um den Wert robuster gegen „Ausreißer“ zu machen.

Metrik 2.4.1:

$$\textit{Kommentaranteil} := \frac{\textit{Anzahl Kommentare}}{\textit{Größe des TestSetups}} \tag{2.4.1}$$

Der Kommentaranteil berechnet sich aus der Anzahl der Kommentare und der Größe des TestSetups. Ein Kommentar beginnt jeweils mit „<!--“ und endet mit „-->“. Die Berechnung der Kommentaranzahl und die Bestimmung der Größe des TestSetups erfolgen anhand eines Beispiels im Anhang unter A.3.

Der Aufwand bei Änderungen der HMSC3-Val-Tests im TestDesign oder TestSetup soll verringert werden (Ziel 3).

Um dieses Ziel zu erreichen, muss die Erweiterbarkeit der TestDesigns und TestSetups untersucht werden. Hierbei sind die Eigenschaften Modularität und Codeduplizierung für die TestDesigns und TestSetups genauer zu betrachten. Durch die Beantwortung folgender Fragen kann der Aufwand bei Änderungen bewertet werden:

- ***Frage 1:*** Ist das TestSetup gut und leicht erweiterbar?
- ***Frage 2:*** Ist das TestDesign gut und leicht erweiterbar?
- ***Frage 3:*** Gibt es Stellen im TestDesign, die wiederholt auftauchen?
- ***Frage 4:*** Gibt es Codeduplizierung im TestSetup?

Die ersten beiden Fragen betrachten den Aufwand bei Änderungen hinsichtlich der Eigenschaft Erweiterbarkeit. Bei der dritten und vierten Frage werden Wiederholungen im TestDesign und TestSetup untersucht, um eine Aussage über den Aufwand bei Änderungen treffen zu können. Der Aufwand einer Änderung steigt mit der Anzahl der Wiederholungen.

Zur Beantwortung der Fragen werden folgende Metriken festgelegt:

Metriken für Modularität:

Anzahl der Fan-Ins einer Action (3.1.1)

Codeduplizierung (Betrachtung bei Frage 4) (3.1.2)

Anhand der Metriken 3.1.1 und 3.1.2 ist die Modularität der Tests erkennbar. Je modularer ein Test aufgebaut ist, desto besser ist er erweiterbar. Für die vorliegenden Tests und das Test-Tool JavaTT wird ein Modul genau einer Action entsprechen. Deshalb zeigt die Metrik 3.1.1 wie viele Scenarios die vorhandenen Actions aufrufen. Die Metrik 3.1.2 betrachtet die Codeduplizierung. Jeder duplizierte Ablauf innerhalb der Testskripte weist auf geringe Modularität hin. Die Bestimmung dieser Metrik wird bei Frage 4 betrachtet.

Metrik 3.2.1:

Bewertung der Namensgebung der TestDesigns (3.2.1)

Die Metrik 3.2.1 kann als Maß für die Erweiterbarkeit des TestDesigns herangezogen werden. Je passender und eindeutiger der Name für ein TestDesign ist, desto besser kann auf den Inhalt eines Tests anhand des Namens geschlossen werden und desto leichter ist die Zuordnung neuer Testfälle zu einem TestDesign. Die Bewertung erfolgt nach drei Kriterien:

- **(Teil 1)** Passt der Name des TestDesigns zum Inhalt des Tests?
- **(Teil 2)** Kann man vom Namen des TestDesigns gut auf das TestSetup schließen?
- **(Teil 3)** Sind Teile des Namens des TestDesigns irreführend oder gibt es sonstige Auffälligkeiten, die eine Erweiterung des TestDesigns erschweren?

Jedes Kriterium wird mit „+“ oder „-“ bewertet. Bei Teil 1 und 2 gibt es Abstufungen von „++“, „+“ über „- “ bis „--“. Bei Teil 3 wird nur eine negative Bewertung vorgenommen. Das heißt, dass für jeden irreführenden Teil im Namen und jede sonstige Auffälligkeit, die die Qualität des Namens des TestDesigns negativ beeinflusst, ein „-“ in diesem Teil vermerkt wird. Des Weiteren kann hier die Modularität hinsichtlich der TestDesigns geprüft werden. Wenn der erste und zweite Teil jeweils positiv gewertet wurde, ist von einem gut abzugrenzenden Testinhalt und einer eindeutigen Namensgebung auszugehen. Dabei würde ein TestDesign genau einem Modul entsprechen. Es handelt sich hier um eine qualitative Metrik. Die Bewertung erfolgt rein subjektiv durch den Autor. Durch die angewandte Wertung mit Ordinaldaten wird jedoch eine Quantifizierung der Metrik erreicht.

Metrik 3.3.1:

Wiederholungen im TestDesign (3.3.1)

Die Metrik 3.3.1 betrachtet die Wiederholungen im TestDesign. Dazu wird das TestDesign auf Formulierungen oder technische Angaben, die in den Beschreibungen der TestSteps wiederholt verwendet werden, untersucht. Diese Metrik wird mit der Suche in Microsoft Word durchgeführt. Die TestDesigns werden wiederum zuvor als Word-Dokumente aus QC exportiert. Gesucht wird nach einer Liste mit bekannten Formulierungen wie:

- **(a)** der Versionsnummer des HMSC3 oder anderen technischen Details,
- **(b)** Benutzernamen/-gruppen und Seriennummern von Implantaten sowie

- **(c)** Bezeichner von Anforderungen in den TestStep-Beschreibungen.

Alle genannten Formulierungen sollten nicht im TestDesign verwendet werden. Derart konkrete Angaben machen eine ständige Änderung des TestDesigns erforderlich. Der Bezeichner einer Anforderung – z.B. „R-UC-TEMPLATE-010“ – darf zwar als TestStep-Name bestehen, sollte aber in der Beschreibung des TestSteps nicht auftauchen. Die Zuordnung einer Anforderung an den TestStep erfolgt in QC lediglich als Verlinkung. Für diese Metrik gibt es keine Gesamtbewertung. Die jeweiligen Formulierungen werden für jede Kategorie gezählt und jeweils bewertet.

Metrik 3.4.1:

$$\textit{Code-Duplizierung} := \frac{\textit{Anzahl duplizierte TestCases}}{\textit{Gesamtanzahl TestCases}} \tag{3.4.1}$$

Für die Bewertung der Code-Duplizierung wird das Tool Simian[5] verwendet. Simian kann neben Java- und C- auch XML-Quellcode sowie normale Text-Dateien auf Duplikate untersuchen. Dabei kann angegeben werden, ab welcher Anzahl übereinstimmender Zeilen (mindestens 2 Zeilen) eine Meldung durch Simian erfolgen soll.

Für die Betrachtung der Codeduplizierung in den vorliegenden Scenarios wird nach mindestens drei übereinstimmenden Zeilen gesucht. Da Simian auch schließende Tags der im XML-Format gespeicherten Scenarios findet, muss eine inhaltliche Analyse der gefundenen Wiederholungen erfolgen. Von allen gefundenen Wiederholungen werden dabei nur die wiederkehrenden Abläufe gezählt, deren Auslagerung in eine Action sinnvoll ist. Es wird jeder wiederkehrende Ablauf sowohl im selben als auch in anderen Scenarios gezählt. Das detaillierte Vorgehen bei der Bestimmung dieser Metrik erfolgt im Anhang unter A.5.

Die Metrik 3.1.2 betrachtet die Codeduplizierung als Maß der Modularität. Je häufiger Abläufe innerhalb der Scenarios wiederholt werden, statt sie in eine Action auszulagern, desto schlechter ist die Modularität.

3.2 Auswertung der statischen Analyse

3.2.1 Auswertung der Metriken

In diesem Abschnitt werden alle festgelegten Metriken auf die Gruppe der Regressionstests angewendet und anschließend bewertet. Soweit bei der Metrik nicht anders angegeben, werden alle Größen bezogen auf das TestSetup mittels XPath

[5] Simian – Similarity Analyser: `http://www.harukizaemon.com/simian/`

ermittelt. Die Größen zu den TestDesigns werden, falls nicht anders beschrieben, mittels inhaltlicher Analyse ermittelt.

Die Metriken werden pro Ziel in einer einzelnen Tabelle betrachtet und anschließend bewertet. In der folgenden Tabelle 3 sind alle Werte zu den Metriken, die für das Ziel 1 ermittelt wurden, abgebildet:

Tabelle 3: Auswertung der Metriken für das Ziel 1
Metrik 1.1.1 – Verh. Sc./TestSteps, Metrik 1.1.2 – Verh. aller Sc.-Aufrufe/TestSteps, Metrik 1.2.1 – Anz. nicht impl. TestSteps, Metrik 1.2.2 – Anteil nicht impl. TestSteps, Metrik 1.3.1 – Anz. Sc. , Metrik 1.3.2 – Anz. aller Sc.-Aufrufe, Metrik 1.3.3 – Anzahl TestSteps

Ziel 1	Performance Batchprint (AllPatients)	Performance Batchprint (PDF with many Episodes)	Patient Administration	Patient Option Template	User Administration	Acknowledge and Postpone	Multiple User Access	Implant and Feature Administration	Performance Transmitter Overview	Performance Print PDF	Performance Download Monitoring Data
Metrik 1.1.1	0,7	1,5	0,7	0,1	1,4	1,7	0,7	0,7	0,2	0,7	3,5
Metrik 1.1.2	0,9	2,5	1,3	0,9	2,5	8,9	1,5	2,1	0,2	2	18
Metrik 1.2.1	3	3	5	7	0	6	5	2	4	6	2
Metrik 1.2.2 (in %)	27,3	75	20,8	5,88	0	37,5	20,8	5,4	100	100	100
Metrik 1.3.1	8	6	18	13	19	28	18	26	1	4	7
Metrik 1.3.2	10	10	31	110	35	143	37	79	1	12	36
Metrik 1.3.3	11	4	24	119	14	16	24	37	4	6	2

Die Metrik 1.1.1 sollte, wie bereits beschrieben, bei guter Testabdeckung und einer bestehenden Zuordnung zwischen Scenario und TestStep einen Wert von „1“ haben. Dieses Ergebnis wird von keinem der Tests erreicht. Die meisten Werte für diese Metrik lassen schlechte Testabdeckung vermuten, da die Ergebnisse kleiner „1“ sind. Dies bedeutet, es gibt bei diesen Tests mehr TestSteps als Scenarios. Für den anderen Teil der Tests ist der Wert deutlich größer als „1“. Somit gibt es bei diesen Tests mehr Scenarios als TestSteps. Die Analysierbarkeit wird dadurch erschwert und verursacht damit schlechtere Wartbarkeit.

Die Metrik 1.1.2 macht deutlich, dass sich durch die Betrachtung der Scenario-Aufrufe das Verhältnis zu den TestSteps in den meisten Fällen verbessert. Bei einigen Tests führt die Betrachtung der Scenario-Aufrufe zu annähernd vollständiger Testabdeckung. Dies

bestätigt die Annahme, dass durch die Parametrisierung der Scenarios die Umsetzung mehrerer TestSteps erfolgen kann. Bei vielen Tests steigt der Wert für das Verhältnis aber auch hier deutlich über „1“ an. Die TestSetups sind offenbar sehr komplex.

Bei der inhaltlichen Betrachtung der TestDesigns und TestSetups kann festgestellt werden, dass die Werte größer „1“ durch komplexe TestSteps bedingt sind. In diesen Fällen ist es notwendig, dass mehrere Scenarios verwendet werden, um einen TestStep umzusetzen. Darüber hinaus treten diverse Fälle auf, in denen mehrere TestSteps durch ein einzelnes sehr umfangreiches Scenario getestet wurden.

Da die Werte der ersten beiden Metriken sehr unterschiedlich sind und bei inhaltlichen Betrachtungen festgestellt werden kann, dass diese Metriken derzeit wenig Aussagekraft besitzen, wird durch weitere Metriken die tatsächliche Testabdeckung betrachtet. Die Werte für die Metriken 1.2.1 und 1.2.2 werden durch inhaltliche Analyse der TestDesigns und TestSetups gebildet. Damit ist sichergestellt, dass die Ergebnisse unabhängig von der Annahme einer Zuordnung von Scenario zu TestStep ermittelt werden.

Bei der inhaltlichen Analyse zeigt sich, dass es diverse TestSteps gibt, zu denen keine Implementierung vorliegt. Zum einen gibt es TestSteps, die inhaltlich die jeweils nachfolgenden TestSteps zusammenfassen und zum anderen gibt es TestSteps, die als manuelle TestSteps gekennzeichnet wurden. Aus diesen Gründen kann für keinen dieser TestSteps eine fehlende Testabdeckung festgestellt werden. Zu den manuellen TestSteps zählen solche, die das Ausführen des Skripts und Prüfen der CSV-Datei verlangen oder auch TestSteps, die das Einhalten von Performance-Werten prüfen. Bei der Metrik 1.2.2 wurden TestDesigns gefunden, bei denen der Anteil nicht implementierter TestSteps bei 100% lag. Dies resultiert ausschließlich aus TestSteps, die die beschriebenen Inhalte aufweisen. Somit kann in diesem Fall nicht von einer fehlenden Testabdeckung ausgegangen werden. Vielmehr weist dies auf eine schlechte Analysierbarkeit hin.

Die Metriken 1.3.1–1.3.3 zeigen die Unterschiede in der Anzahl der TestSteps im TestDesign zur Anzahl der Scenarios sowie Scenario-Aufrufe im TestSetup. Zur besseren Veranschaulichung der Werte in der Tabelle wurden sie zusätzlich in einem Balkendiagramm visualisiert (siehe Abbildung 5).

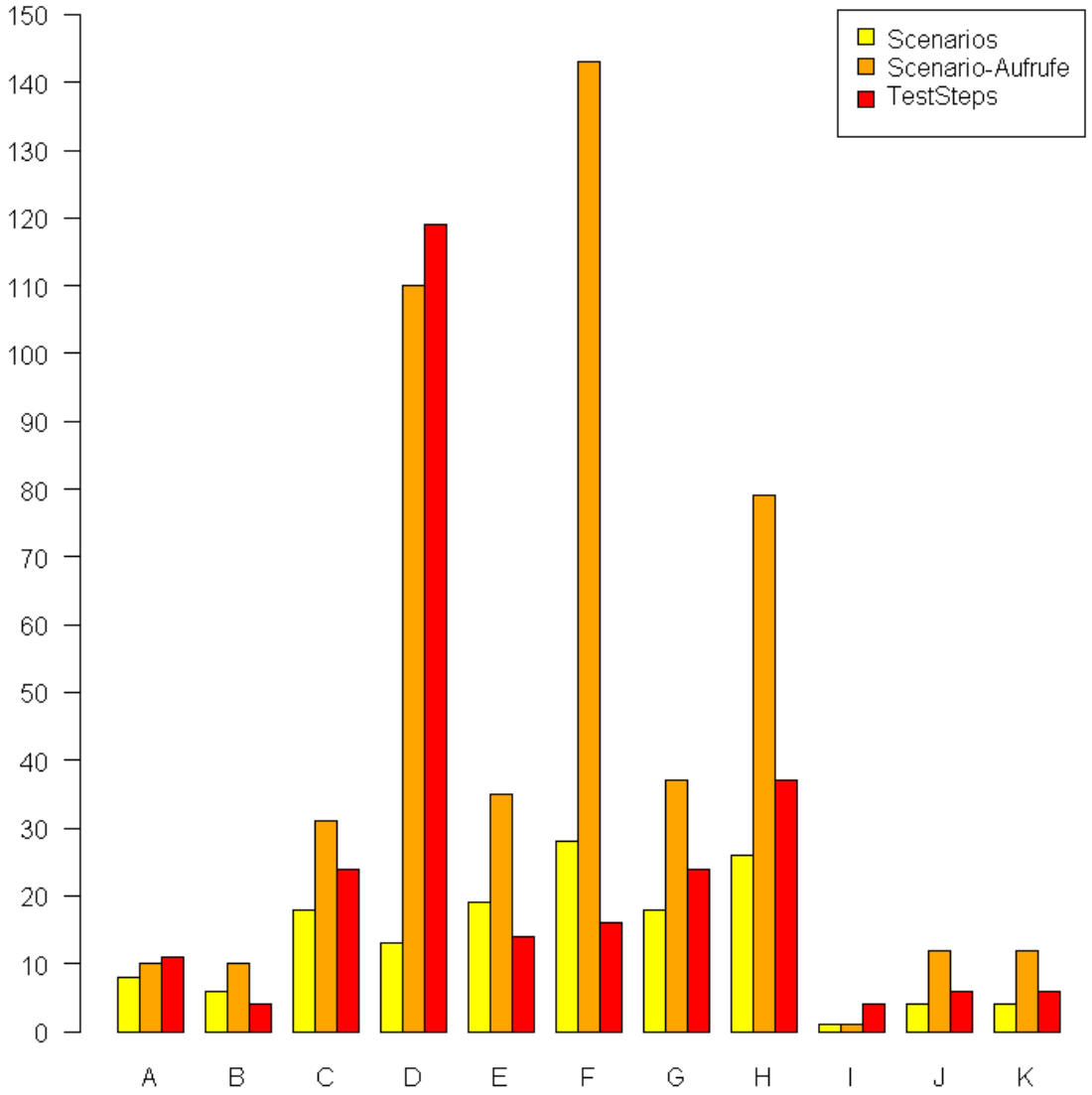

Abbildung 5: Anzahl der Scenarios, Scenario-Aufrufe und TestSteps
A – Performance Batchprint (All Patients),
B – Performance Batchprint (PDF with max. Episodes),
C – Patient Administration,
D – Patient Option Template,
E – User Administration,
F – Acknowledge and Postpone,
G – Multiple User Access,
H – Implant and Feature Administration,
I – Performance Transmitter Overview,
J – Performance Print PDF,
K – Performance Download Monitoring Data

Im Balkendiagramm wird deutlich, wie groß der Unterschied zwischen der Spezifikation im TestDesign und den implementierten Testskripten allein in der Anzahl der Elemente ist. Zudem ist in diesem Diagramm erkennbar, wie sich die Verhältnisse aus den Metriken 1.1.1 und 1.1.2 darstellen.

In der folgenden Tabelle 4 sind alle Werte zu den Metriken, die für das Ziel 2 ermittelt wurden, abgebildet:

Tabelle 4: Auswertung der Metriken für Ziel 2
Metrik 2.1.1 – Fehlerquotient,
Metrik 2.2.1 – Anzahl nicht spez. Sc., Metrik 2.2.2 – Anteil nicht spez. Sc.,
Metrik 2.3.1–2.3.3 – min, max, mittlere Anz. Actions pro Sc.,
Metrik 2.3.4–2.3.6 – min, max, mittlere Anz. TestCases pro Sc.,
Metrik 2.3.7–2.3.9 – min, max, mittlere Anz. TestItems pro Sc.,
Metrik 2.4.1 – Kommentaranteil

Ziel 2	Performance Batchprint (AllPatients)	Performance Batchprint (PDF with many Episodes)	Patient Administration	Patient Option Template	User Administration	Acknowledge and Postpone	Multiple User Access	Implant and Feature Administration	Performance Transmitter Overview	Performance Print PDF	Performance Download Monitoring Data
Metrik 2.1.1	1,01	0,87	0,9	0,9	0,99	1,1	0,5	0,43	0,43	0,72	1,13
Metrik 2.2.1	2	0	2	3	5	11	1	6	0	4	7
Metrik 2.2.2 (in %)	25	0	11,11	23,08	26,32	39,29	5,6	23,08	0	100	100
Metrik 2.3.1	0	0	0	0	0	0	0	0	2	0	0
Metrik 2.3.2	1	1	1	1	1	2	1	1	2	1	4
Metrik 2.3.3	0	0,5	1	0	0	0	0	0	2	0,5	1
Metrik 2.3.4	1	1	1	1	1	1	1	1	11	1	1
Metrik 2.3.5	2	4	6	9	5	5	5	3	11	4	10
Metrik 2.3.6	1	2,5	3	2	2	2	1	1	11	2,5	3
Metrik 2.3.7	0	0	0	0	0	0	1	0	45	0	0
Metrik 2.3.8	18	25	23	51	17	79	18	108	45	21	16
Metrik 2.3.9	1,5	6,5	10	12	9	16	2,5	8	45	4,5	10
Metrik 2.4.1 (in %)	34,44	22,92	39,04	35,81	37,3	26,61	48	19,05	8,57	22,92	28

Der Fehlerquotient (Metrik 2.1.1) ist ein Maß für die Anzahl der Rechtschreibfehler in den TestDesigns. Die Bewertung der Metrik erfolgt mittels Abiturmaßstab, bei dem anhand des Fehlerquotienten die Noten vergeben werden. Für die vorliegenden TestDesigns ergeben sich Schulnoten von „1“ bis „2+“ (siehe Notenübersicht in Tabelle 2).

Die Metriken 2.2.1 und 2.2.2 zeigen, dass es in den meisten TestSetups unspezifizierte Scenarios gibt. Bei zwei TestSetups liegt der Anteil sogar bei 100%. Dies weist eher auf ein unvollständiges TestDesign hin, als dass von zu komplexen TestSetups ausgegangen werden muss. Sowohl zu komplexe TestSetups als auch unvollständige TestDesigns führen dazu, dass der Ablauf des TestSetups nicht anhand des TestDesigns erkennbar ist und die TestSetups schwerer verständlich sind. Dies wirkt sich negativ auf die Analysierbarkeit aus.

Die Bewertung der Metriken über die Komplexität (Metriken 2.3.1–2.3.9) fällt sehr unterschiedlich aus. Anhand der minimalen und maximalen Werte sowie dem Median lassen sich insbesondere für die TestCases und TestItems nur schwer Aussagen zur Komplexität treffen. Lediglich für die Anzahl der Aufrufe einer Action kann festgestellt werden, dass diese zwischen null und vier Actions pro Scenario liegt – betrachtet für jeweils einen Test. Dabei ist der mehrfache Aufruf einer Action in dieser Wertung beinhaltet. Dies spricht für eine sehr geringe Nutzung der Actions, in deren Folge es komplexere Scenarios gibt. Die Komplexität der Scenarios wird anhand der TestCases und TestItems durch die zusätzliche Darstellung in einer Boxplot veranschaulicht. Mit Hilfe einer Boxplot ist die Verteilung und die Lage der Daten sehr gut darstellbar. Die Boxplot wird durch eine Stripchart ergänzt, die zusätzlich verdeutlicht, wie groß die jeweiligen betrachteten Datenmengen sind.

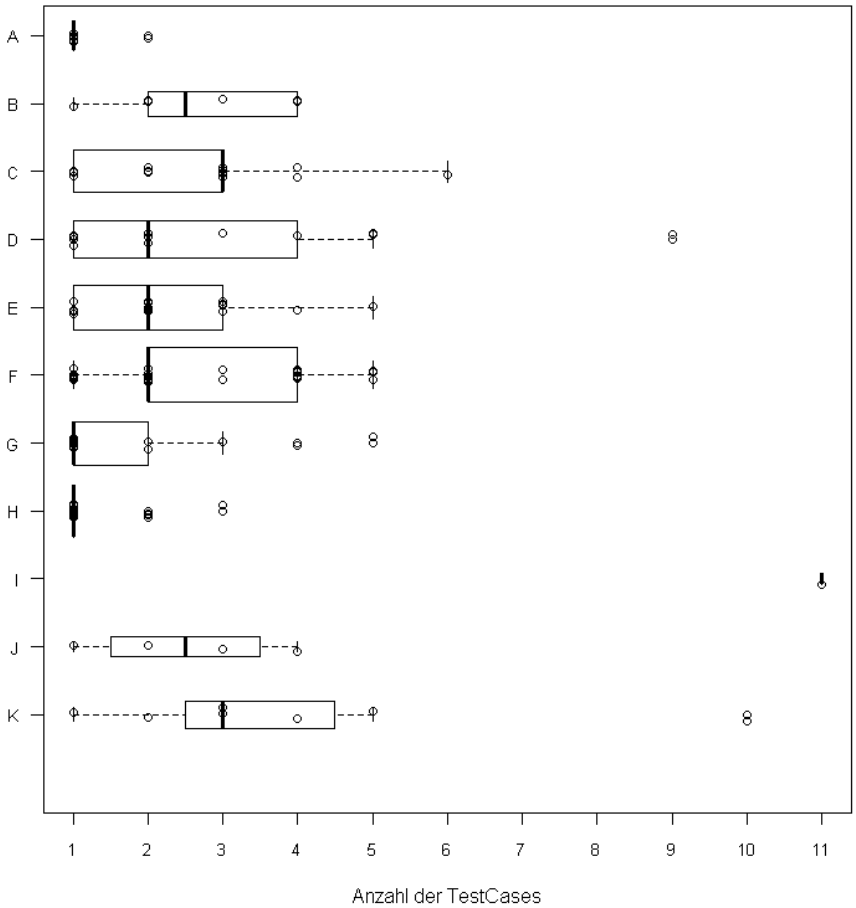

Abbildung 6: Anzahl der TestCases pro Scenario
A – Performance Batchprint (All Patients) (8),
B – Performance Batchprint (PDF with max. Episodes) (6),
C – Patient Administration (18),
D – Patient Option Template (13),
E – User Administration (19),
F – Acknowledge and Postpone (28),
G – Multiple User Access (18),
H – Implant and Feature Administration (26),
I – Performance Transmitter Overview (1),
J – Performance Print PDF (4),
K – Performance Download Monitoring Data (6)

Die Boxenhöhe ist abhängig von der Größe der Datenmenge. Diese ermittelt sich aus der Anzahl der Scenarios, die im jeweiligen Test aufgerufen werden (die Klammerangabe hinter dem Testnamen bezeichnet die Anzahl der Scenarios). So ist die Box beim Test „Acknowledge and Postpone“ durch 28 im TestSetup enthaltene Scenarios deutlich höher, als die Box beim Test „Performance Print PDF“, der nur vier Scenarios enthält. Für die Boxplot der TestCases in Abbildung 6 ist festzustellen, dass bei vielen Tests die Hälfte aller Scenarios weniger als drei TestCases enthalten. Außerdem gibt es selten mehr als vier oder fünf TestCases innerhalb eines Scenarios. Des Weiteren ist festzustellen, dass häufig nur ein TestCase im Scenario vorhanden ist. Die Anzahl der TestCases pro Scenario scheint unabhängig von der Anzahl der Scenarios im TestSetup (Größe der Datenmenge) zu sein. Das heißt, egal ob ein Test sehr viele oder sehr wenige Scenarios enthält, sind die Scenarios hinsichtlich der TestCases nicht mehr oder weniger komplex. Die Ausnahme davon bildet der Test I – „Performance Transmitter Overview“ – mit nur einem Scenario, welches die größte Anzahl an TestCases enthält (elf TestCases).

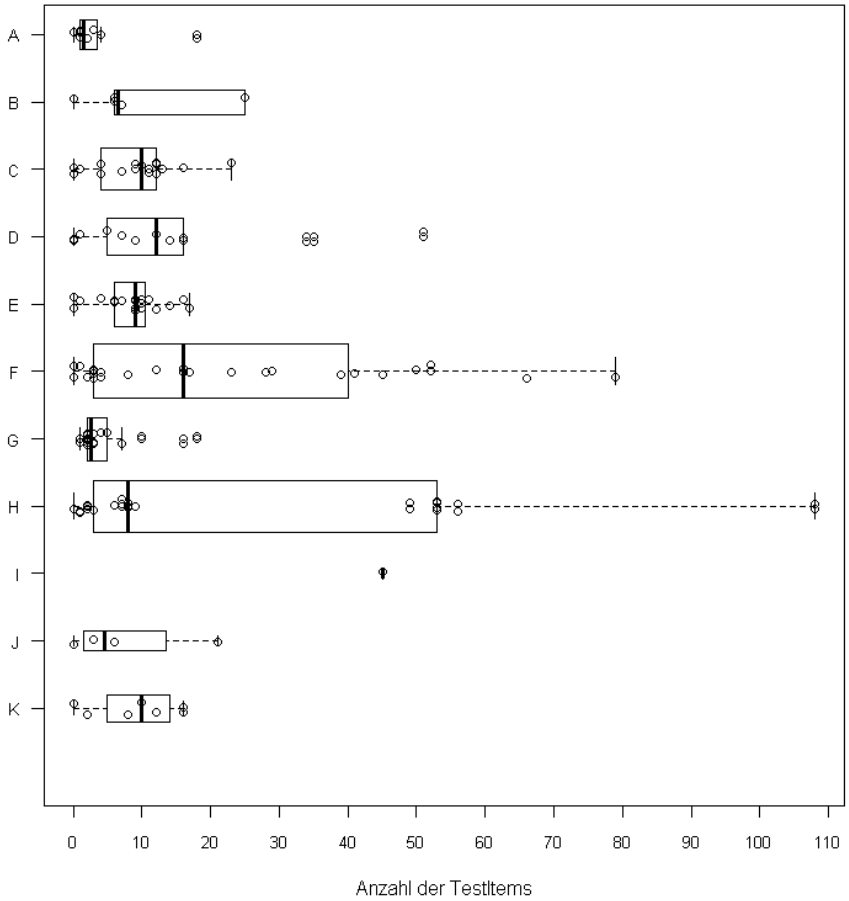

Abbildung 7: Anzahl der TestItems pro Scenario
A – Performance Batchprint (All Patients) (8),
B – Performance Batchprint (PDF with max. Episodes) (6),
C – Patient Administration (18),
D – Patient Option Template (13),
E – User Administration (19),
F – Acknowledge and Postpone (28),
G – Multiple User Access (18),
H – Implant and Feature Administration (26),
I – Performance Transmitter Overview (1),
J – Performance Print PDF (4),
K – Performance Download Monitoring Data (6)

Gleichfalls besteht keine Korrelation zwischen der Anzahl der TestItems im Scenario und der Anzahl der Scenarios im TestSetup. Die eingangs getroffene Annahme, dass TestSetups mit nur wenigen Scenarios automatisch sehr komplexe Scenarios enthalten, ist somit widerlegt. Weiterhin fällt bei der maximalen Anzahl der TestItems auf, dass sehr viele TestItems in den Scenarios enthalten sind. Dies kann eine weitere Ursache dafür sein, dass in Metrik 1.1.1 Werte kleiner „1“ auftreten. Bei einer hohen Anzahl an TestItems ist das Scenario so komplex, dass es meist mehrere TestSteps umsetzt.

Trotz der übersichtlichen Darstellung der Streuung der Daten lässt sich keine allgemeine Aussage über das Verhältnis von TestCases zu TestItems treffen. Auch wenn im Mittel auf etwa zwei bis drei TestCases 10–15 TestItems kommen – was für klar strukturierte und lesbare Scenarios spricht – zeigt sich bei der inhaltlichen Analyse, dass diese Werte häufig nicht mit dem tatsächlichen Verhältnis von TestCases zu TestItems übereinstimmen. So gibt es Scenarios, die sehr viele TestCases bei gleichzeitig sehr wenigen TestItems enthalten. Gleichzeitig gibt es Scenarios mit sehr wenigen TestCases – häufig sogar nur einem TestCase – bei hoher Anzahl an TestItems. Aus diesem Grund ist eine Wertung dieser Metriken hinsichtlich des Verhältnisses von TestCases zu TestItems nur mit unterstützender inhaltlicher Analyse sinnvoll. Aber auch mit inhaltlicher Analyse sind der Umfang und die Komplexität der Scenarios so verschieden, dass keine allgemein gültige Aussage bezüglich der Verständlichkeit getroffen werden kann.

Die Metrik 2.4.1 betrachtet den Kommentaranteil der TestSetups, Scenarios und Actions. Die Werte für den Kommentaranteil der einzelnen Tests liegen im Mittel bei 30% . Daher könnte man von einem ausreichendem Anteil an Kommentaren im Code ausgehen. Allerdings fällt bei detaillierter Betrachtung der einzelnen Tests auf, dass häufig auch sinnlose Kommentare vorhanden sind, sodass das Ergebnis relativiert werden muss.

In der Tabelle 5 sind alle Werte für die Metriken von Ziel 3 abgebildet:

Tabelle 5: Auswertung der Metriken für Ziel 3
Metrik 3.2.1 – Bewertung der Namensgebung,
Metrik 3.3.1 – Wiederholungen im TestDesign,
Metrik 3.4.1 (3.1.2) – Codeduplizierung

Ziel 3	Performance Batchprint (AllPatients)	Performance Batchprint (PDF with many Episodes)	Patient Administration	Patient Option Template	User Administration	Acknowledge and Postpone	Multiple User Access	Implant and Feature Administration	Performance Transmitter Overview	Performance Print PDF	Performance Download Monitoring Data
Metrik 3.2.1. (Teil 1)	++	++	++	--	++	++	++	++	++	++	++
Metrik 3.2.1. (Teil 2)	+	-	++	--	++	++	++	++	++	-	++
Metrik 3.2.1. (Teil 3)	-	-	--	-	-	--	-	-	/	/	/
Metrik 3.3.1. (a)	3	3	10	10	3	5	3	/	3	3	2
Metrik 3.3.1. (b)	15	15	3	3	/	10	7	/	9	/	3
Metrik 3.3.1. (c)	/	/	33	33	8	4	4	29	4	9	/
Metrik 3.4.1. (3.1.2)	0,5	0,42	0,6	0,55	0,55	0,2	0,5	0,27	0,23	0,33	0,51

Die Auswertung für die Metrik 3.1.1 ist in einer separaten Tabelle (siehe Tabelle 6) enthalten, da hier nicht die Anzahl der Actions pro TestSetup betrachtet wird, sondern die Anzahl der Aufrufe der jeweiligen Actions.

Die Metrik 3.2.1 wird mittels der Einzelbewertungen in den Teilen 1–3 untersucht. Die Namen der TestDesigns sowie die Namen der zugehörigen TestSetups befinden sich als Tabelle im Anhang unter A.4. Die Bewertung der einzelnen Teile wird anhand der ersten beiden Tests („Performance Batchprint (AllPatients)“ und „Performance Batchprint (PDF with many Episodes)“) näher erläutert. In Teil 1 erhalten beide Tests die Wertung „++“, da anhand des Namens des TestDesigns auf die Inhalte der Tests geschlossen werden kann. In Teil 2 erhält nur der Test „Performance Batchprint (AllPatients)“ eine teilweise positive Wertung, da der Name des TestSetups ebenfalls einen Hinweis auf den Inhalt des Tests gibt (Batchprinting). Da die beiden Tests in einem gemeinsamen TestDesign spezifiziert sind, ist dies eine Auffälligkeit, die im Teil 3 zu einem „-“ in der Wertung führt.

Die Metrik 3.3.1 unterteilt sich ebenfalls in drei Kategorien. Die erste Kategorie (a) untersucht das wiederholte Auftauchen der Versionsnummer des HMSC oder anderer technischer Details im TestDesign. Die zweite Kategorie (b) umfasst Wiederholungen im TestDesign in Bezug auf die Angabe von Benutzernamen oder -gruppen sowie

Seriennummern von Implantaten. Die dritte Kategorie (c) zählt die Wiederholungen der Bezeichner von Anforderungen in der Beschreibung der TestSteps. Wie bereits in der Metrik erklärt wird, sollte keine der genannten Angaben im TestDesign vorhanden sein. Deshalb werden alle vorkommenden Wiederholungen negativ gewertet. Die Tests, die in einem gemeinsamen TestDesign spezifiziert werden, erhalten zur Wertung die gleiche Anzahl an Wiederholungen.

Die Metrik 3.4.1 zeigt den Anteil duplizierter TestCases. Zu den duplizierten TestCases werden auch Abläufe, die bisher keine TestCases waren, gezählt, wenn sie aus drei oder mehr Zeilen bestehen (genauer nachzulesen im Anhang unter A.5). Tritt ein solcher Ablauf oder TestCase im selben oder in einem anderen Scenario noch einmal auf, gilt er als dupliziert. Der Anteil dieser duplizierten TestCases ist für alle Tests zu hoch. Die häufige Wiederholung von Abläufen führt neben erhöhter Komplexität der Scenarios zu schlechter Änderbarkeit der Tests. Bei einer notwendigen Änderung an einem Ablauf müssen meist drei bis vier weitere Scenarios angepasst werden. Weiterhin ist ein hoher Anteil an dupliziertem Code ein Anzeichen für schlechte Modularität. Jeder wiederholte Ablauf sollte in eine Action ausgelagert werden. Um die bisherige Nutzung der Actions zu verdeutlichen, eignet sich auch die Metrik 3.1.1, die in der folgenden Tabelle 6 näher betrachtet wird.

Tabelle 6: Auswertung der Metrik 3.1.1 für Ziel 3
Metrik 3.1.1 – Anzahl der Fan-Ins

Ziel 3	DownloadMonitoringData	DownloadMonitoringDataBatchPrint	Fill_Login	Fill_Login_CSS	MessageSendT3PwithCurrentSCTS	Select_Patient	Select_Patient_Search	SendAllMessagesT3P
Metrik 3.1.1.	1	1	10	1	1	15	1	1

Die Anzahl der Aufrufe einer Action liegt zwischen null und vier Actions pro Scenario je Test (siehe Tabelle 4 Metrik 2.3.1–2.3.3). Dabei ist der mehrfache Aufruf einer Action in dieser Wertung beinhaltet. Im Dateisystem der Regressionstests sind elf Actions vorhanden. Von diesen werden nur sieben Actions aufgerufen. Für diese sind in der Tabelle 6 die Gesamtanzahl der Aufrufe aus allen Scenarios dargestellt. Dabei rufen nur 24 von 122 Scenarios eine Action auf. Dies spricht für schlechte Modularität. Eine

beginnende Modularisierung ist an den Actions „Fill_Login“ und „Select_Patient“ erkennbar. Hier liegt die Anzahl der Aufrufe deutlich höher als bei den übrigen Actions.

3.2.2 Einhaltung der Testeffektivität oder Wartbarkeit

Für die Einhaltung der Qualitätsmerkmale der Testeffektivität und Wartbarkeit sollte es Regeln und Vorschriften geben. Bei der Untersuchung der Regressionstests wird festgestellt, dass keine Regeln vorliegen. Das Fehlen von Regeln dürfte eine der Ursachen für die schlechte Wartbarkeit sein. Fehlende Struktur- und Namensvorgaben verursachen insbesondere Defizite für die Analysierbarkeit und Verständlichkeit der Tests.

3.3 Zielvorgaben für die Verbesserung der Testqualität

In diesem Abschnitt werden aus den festgestellten Defiziten Zielvorgaben für Verbesserungen der Testqualität erarbeitet. Diese teilen sich auf in Zielvorgaben, die innerhalb dieser Arbeit erreicht werden sollen und Regelvorgaben, durch deren Einhaltung zukünftig die Qualität in den Bereichen der Testeffektivität und Wartbarkeit erhalten wird. Die einzelnen Zielvorgaben und Regeln werden – soweit möglich – nach den Qualitätsmerkmalen unterschieden. Die Unterscheidung wird nach Gewichtung des Nutzens dem jeweiligen Qualitätsmerkmal zugeordnet. Die meisten Vorgaben bewirken darüber hinaus aber auch eine Veränderung im jeweils anderen Bereich.

3.3.1 Zielvorgaben für die Verbesserung der Testeffektivität

Bei der Testeffektivität sind für das betrachtete Merkmal Testabdeckung keine groben Mängel festzustellen. Dennoch wird die statische Analyse durch die fehlende Zuordnung zwischen den TestSteps in QC und den Scenarios im TestSetup erschwert.

Aus diesem Grund soll eine Zuordnung zwischen den TestSteps und den Scenarios hergestellt werden. Dazu ist es erforderlich, TestSteps, für die es keine Umsetzung im TestSetup gibt, gesondert zu behandeln (siehe Metrik 1.2.1 und 1.2.2). Diese TestSteps können entweder nachträglich implementiert werden, in ein rein manuelles TestDesign ausgelagert werden oder sie müssen deutlich als manuelle TestSteps gekennzeichnet werden. Die letzte Variante sollte nur umgesetzt werden, wenn keine der beiden anderen Möglichkeiten realisierbar ist. Dies betrifft z.B. TestSteps, die eine Angabe von Performance-Werten enthalten, da die automatisierte Überprüfung von Performance-Werten in JavaTT derzeit nicht möglich ist. Durch die deutliche Kennzeichnung dieser manuellen TestSteps kann die falsche Annahme einer schlechten Testabdeckung vermieden werden.

Ein Ziel in dieser Arbeit ist es, für den Bereich der Testeffektivität eine automatisierte Prüfung dieser Zuordnung zu ermöglichen. Dazu soll das RepFram entwickelt werden. Dieses Tool soll für jedes Scenario das ermittelte Ergebnis in den korrespondierenden

TestStep in QC schreiben. Dadurch werden alle TestSteps aufgedeckt, die kein zugehöriges Scenario besitzen (Metrik 1.2.1). Dies erlaubt eine sofortige Kontrolle der Testabdeckung. Außerdem ermöglicht das RepFram die Aufdeckung überflüssiger Scenarios im TestSetup, da diese keinen korrespondierenden TestStep im TestDesign besitzen. So kann durch das RepFram auch die Wartbarkeit hinsichtlich zu komplexer TestSetups kontrolliert werden. Bei der Zuordnung müssen zusätzlich alle Scenarios berücksichtigt werden, die das TestSetup vor- und nachbereiten. Da es für diese Scenarios derzeit keine TestSteps im TestDesign gibt, sollen zwei zusätzliche TestSteps „Precondition“ und „Postcondition“ angelegt werden. Alle Scenarios mit Inhalten zur Vor- und Nachbereitung des TestSetups werden zu einem derartigen Scenario zusammengefasst.

Insgesamt soll das Verhältnis zwischen den Scenarios und den TestSteps (Metrik 1.1.1) durch geeignete Strukturen und die Implementierung des RepFrams aussagekräftiger werden. Für die Fälle, in denen das Verhältnis der Scenario-Aufrufe zu den TestSteps besser anwendbar ist (siehe Metrik 1.1.2), muss bei der Entwicklung eine Lösung gefunden werden.

3.3.2 Zielvorgaben für die Verbesserung der Wartbarkeit

Nach Auswertung der Metriken zur Betrachtung der Wartbarkeit sind verschiedene Defizite in der Qualität erkennbar. Die TestSetups und Scenarios sind häufig sehr komplex, was die Verständlichkeit der Skripte erschwert. Außerdem gibt es keine einheitlichen Strukturen und Vorgaben, anhand derer man ohne detaillierte Betrachtung den Umfang eines TestSetups oder Scenarios erkennen kann. Zudem erschweren die Komplexität, die häufigen Codeduplizierungen sowie die fehlende Modularität der Testskripte die Erweiterbarkeit der Tests.

Zur Verbesserung der Analysierbarkeit und Erweiterbarkeit soll ein Konzept zur Modularisierung entwickelt werden. Bei der Modularisierung werden Actions als Module betrachtet. Hierbei wird stets eine lose Kopplung der Module erreicht, da es keinen gegenseitigen Aufruf von Actions gibt.

Darüber hinaus sollen alle nicht genutzten TestSetups, Scenarios und Actions gelöscht werden, um die Übersichtlichkeit im Dateisystem zu verbessern.

Die Qualität im Bereich der Wartbarkeit soll durch Einhaltung folgender Richtlinien dauerhaft erhalten werden:

- Einhaltung der Namenskonvention für die TestDesigns und TestSetups sowie die darin enthaltenen Elemente,
- keine Angabe von technischen Details, Nutzerdaten, Seriennummern von

Implantaten (o.ä.) sowie Bezeichnern für Anforderungen,

- Begrenzung der Anzahl der TestItems pro Scenario sowie
- eigene Menge an Scenarios für jedes TestSetup.

Eine bessere Zuordnung von TestDesign und TestSetup könnte durch Verwendung folgender Namenskonventionen erzielt werden:

- Die TestDesigns und TestSetups erhalten den gleichen Namen.
- Das Attribut `Name` im Scenario und der Dateiname des Scenarios erhalten die gleiche Bezeichnung.
- Der Bezeichnung des Scenarios wird der Name des TestSetups als Kürzel vorangestellt (z.B. wird beim Test „Patient Administration“ ein PA vorangestellt).

Die erste Namenskonvention hat direkten Einfluss auf die Metrik 3.2.1, da mit ihr vom Namen des TestDesigns unmittelbar auf das TestSetup geschlossen werden kann. Die zweite Konvention gilt grundsätzlich auch für die TestSetups und Actions – das Scenario steht dabei stellvertretend für die anderen Elemente. Diese Konvention trägt dazu bei, dass die jeweiligen Elemente schnell im Dateisystem gefunden werden können. Die dritte Konvention ermöglicht eine Zuordnung der Scenarios zum TestSetup.

Für die Angabe von technischen Details (z.B. Versionsnummer des HMSC), Nutzerdaten, Seriennummern von Implantaten sowie Bezeichnern für Anforderungen gelten die Vorgaben aus der Metrik 3.3.1. Diese Richtlinie hat somit auch direkten Einfluss auf die Werte in dieser Metrik.

Die Begrenzung der Anzahl der TestItems soll über das RepFram geprüft werden. Vorstellbar ist die Prüfung über die Anzahl der Einträge der Ergebnismengen aus JavaTT. Die Begrenzung der Anzahl der TestItems hat Einfluss auf die Werte der Metriken 2.3.7–2.3.9.

Die letzte Richtlinie, die eine eigene Menge an Scenarios für jedes TestSetup verlangt, bietet den Vorteil einer wiedererkennbaren Struktur im Dateisystem. Alle zu einem Test gehörenden Scenarios sollen in einem gemeinsamen Ordner abgelegt werden. Dies vereinfacht die Suche nach einem Scenario bei der Analyse oder Erweiterung eines Tests. Die Gefahr der Codeduplizierung kann durch die Nutzung von Actions vermieden werden.

Im nächsten Schritt werden die Vorgaben weitestgehend umgesetzt. Dabei wird insbesondere auf die Entwicklung eines Strukturmusters für die TestSetups und TestDesigns sowie die Implementierung des RepFrams eingegangen.

4 Implementierung

Auf Grundlage der theoretischen Betrachtungen und getroffenen Entscheidungen wird eine Musterstruktur für die TestSetups und TestDesigns entworfen. Diese Musterstruktur wird als Basis für die darauf folgende Umsetzung einer automatisierten Übertragung der Ergebnisse nach ALM mit dem RepFram benutzt.

4.1 Aufbau der Musterstruktur

4.1.1 Konzept

Im Mittelpunkt der Musterstruktur steht die Zuordnung vom Scenario zum TestStep. Bei dem Umbau der TestSetups und TestDesigns sind folgende Punkte zu beachten. Statt einer Nummerierung im Stepnamen erhalten die TestSteps den Namen des zugehörigen Scenarios. Alle manuellen TestSteps werden als solche sichtbar gekennzeichnet. Vor- und nachbereitende Scenarios des TestSetups werden jeweils in einem Scenario zusammengefasst und entsprechend der Namenskonvention bezeichnet.

Neben diesen formalen Änderungen sind bei der Umformung weitere Aspekte zu beachten. Wiederkehrende Scenarios wie z.B. „Login“, die nur zur Vorbereitung des nachfolgenden Scenarios dienen, werden aus dem TestSetup gelöscht. Der jeweils vorbereitende Ablauf wird in das nachfolgende Scenario integriert. Handelt es sich um einen wiederkehrenden Ablauf, wird er zukünftig als Action aufgerufen. Andernfalls werden die vorbereitenden TestItems nicht ausgelagert und innerhalb des Scenarios ausgeführt. Dieser Umformungsschritt reduziert die Komplexität der TestSetups, da nur noch Scenarios enthalten sind, die zuvor als TestStep spezifiziert wurden. Die Auslagerung von wiederkehrenden Abläufen in Actions vermeidet das „Aufblähen“ der Scenarios. Durch diese Änderung verbessern sich die Werte der Metrik 1.1.2, bei denen das Verhältnis der Scenario-Aufrufe zu den TestSteps deutlich größer als „1“ war.

Je nach Inhalt des Testfalls kann eine Umsetzung mehrerer konkreter Testfälle zu einem abstrakten Testfall erforderlich sein. In diesen Fällen wird ein TestStep durch mehrere Scenario-Aufrufe umgesetzt. Jeder Scenario-Aufruf entspricht dabei einem konkreten Testfall. Bei der Erstellung des TestDesings ist zu prüfen, ob eine solche Umsetzung sinnvoll ist. Alternativ könnte es sich anbieten, mehrere abstrakte Testfälle zu spezifizieren. Dabei wäre jeder Scenario-Aufruf in ein eigenständiges Scenario zu übertragen. Durch die Nutzung von Actions könnten auch in diesem Fall wiederkehrende

Abläufe vermieden werden. Je nach Umsetzung verbessern sich die Werte der Metriken 1.1.1 oder 1.1.2. Die Umsetzung mehrerer konkreter Testfälle zu einem abstrakten Testfall könnte im Test „Peformance Print PDF“ erfolgen. In diesem Test werden Kardioberichte für einen Patienten erstellt. Die Berichte werden nacheinander für eine, zwei und drei Langzeitepisoden ausgewählt. Der abstrakte Testfall würde die Erstellung des Kardioberichts enthalten, die konkreten Testfälle dagegen die Umsetzung für die unterschiedlichen Langzeitepisoden.

Ein weiterer Aspekt ist die Verbesserung der Modularität, die durch das vorgeschlagene Konzept zur Modularisierung umgesetzt wird. Eine optimale Modularität wäre durch die Erstellung kleiner Module (Actions), wie z.B.:

- „Click <Menu>“,
- „Click Button ‘Apply‘ on <Page>“,
- „Click Button ‘Confirm‘ on <Page>“ und
- „Add Parameter“,

zu erreichen. Diese Module müssten unter Mitgabe von Parametern aus den Scenarios heraus aufgerufen werden. Die Parameter enthalten dabei z.B. den Wert des zu öffnenden Menüs. Dieses Konzept ist jedoch mit JavaTT nicht umsetzbar, da die Parametrisierung nur aus dem TestSetup erfolgen kann. So könnte der Parameter <Menu> nur einmal pro Scenario-Aufruf gesetzt werden. Dies steht der Erstellung kleiner Module entgegen.

Die auf JavaTT angepasste Modularisierung der Testskripte ergibt sich wie folgt. Bei der Suche nach Codeduplizierung (siehe Metrik 3.4.1) werden alle wiederkehrenden Abläufe aufgedeckt. Diese werden jeweils in eine Action ausgelagert und stellen eigenständige Module dar. Diese 22 Actions enthalten kleinere Abläufe, wie:

- „Login“,
- „Logout“,
- „Select Patient“,
- „Click Menu ‘All Patients‘“,
- „Search and Select Patient“,
- „Deactivate Home Monitoring“,
- „Delete Patient“,
- „New User“ und

- „Check if Button ‘Apply‘ is enabled“

Die Komplexität der gewählten Module kann variieren und ist von der Größe des wiederkehrenden Ablaufs abhängig. Bei einer statischen Analyse der Testskripte kann die Nutzung der Actions als Maß für die Modularität mit der Metrik 3.1.1 überprüft werden.

4.1.2 Umsetzung am Test „Patient Option Template“

Die vorgestellte Musterstruktur wird beispielhaft am TestSetup „PatientOptionTemplate“ und dem zugehörigen TestDesign umgesetzt. Der Test „Patient Option Template“ befasst sich mit dem Erstellen, Verändern und Löschen von sogenannten „Option Templates“. Diese Templates speichern für jeden Befund eines Implantatstyps (Cylos DRT, Belos ApT, Kronos LVT, Lexos ApT, Philos 1 DRT, Stratos LVT, Lumax 540 DRT, Evia DRT) unterschiedliche Benachrichtigungsoptionen. Zum Beispiel kann eingestellt werden, dass bei allen Implantaten vom Typ Cylos DRT bei einer mittleren Herzfrequenz von mehr als 80 Schlägen pro Minute, der Status des Patienten auf „rot“ gesetzt wird und der Arzt eine extra Benachrichtigung erhält. Im Test wird geprüft, ob unterschiedliche Benutzerrollen – Arzt, Benutzeradministrator und Customer Service-Spezialist – die „Option Templates“ anlegen können und diese auch unabhängig vom Ersteller des Templates wieder löschen können. Für das weitere Verständnis dieses Tests sind zudem die Unterschiede zwischen Implantatstyp und Implantatsfamilie von Bedeutung. So gehören die Typen Belos ApT und Lexos ApT zur selben Implantatsfamilie.

Diese Auswahlentscheidung wurde auf Basis der schlechten Werte insbesondere im Bereich der Erweiterbarkeit getroffen. So wurde im gesamten Skript nur einmal eine Action aufgerufen (Metrik 2.3.1–2.3.3). Des Weiteren war dieser Test im Bereich der Codeduplizierung (Metrik 3.4.1) negativ auffällig. Der Anteil wiederkehrender Abläufe lag bei über 50%. Hinzu kommt, dass dieser Test gemeinsam mit dem Test „Patient Administration“ in einem TestDesign spezifiziert wird. Dies verursacht eine schlechte Bewertung der Namensgebung des TestDesings (Metrik 3.2.1). Insgesamt ist der Test sehr schwer verständlich, weil der Testablauf nicht anhand des TestDesigns erkennbar ist. Das TestDesign ist sehr kleinschrittig und komplex. Auch das TestSetup ist mit 110 Scenario-Aufrufen schlecht zu überschauen (Metrik 1.3.2 und 1.3.3).

Zunächst wurde das TestSetup auf inhaltlich zusammenhängende Abläufe untersucht, die zu einem konkreten Testfall zusammengefasst werden können. Das umfangreiche und kleinschrittige TestDesign bot sich nicht als Grundlage für den Umbau an. Hier konnten keine logischen Testfälle ermittelt werden, denen man konkret ein Scenario zuordnen konnte. In einem späteren Schritt wurde erkannt, dass es bereits TestSteps gibt, die inhaltlich die jeweils nachfolgenden TestSteps zusammenfassen. Diese TestSteps können im TestDesign als abstrakte Testfälle verwendet werden.

Es konnten zunächst folgende abstrakte Testfälle herausgearbeitet werden:

- Erstellen eines „Option Templates“ für verschiedene Implantatstypen als
  - Arzt,
  - Benutzeradministrator (UserAdmin) und
  - Customer Service-Spezialist (CssAdmin),
- Prüfen, dass andere Implantatstypen fremde „Option Templates“ nicht sehen und annehmen dürfen,
- Ändern eines „Option Templates“,
- Prüfen, ob gleicher Implantatstyp das eigene „Option Template“ sehen und annehmen darf und
- Löschen eines „Option Templates“ – jede Benutzerrolle kann unabhängig vom Ersteller des „Option Templates“ dieses löschen.

Nach einer Diskussion mit den Testern der Validierungsgruppe wurden die Testfälle zusätzlich besser strukturiert. In der alten Variante des TestSetups wurde eine Vielzahl von „Option Templates“ erstellt. Im Ergebnis der Neustrukturierung wurde die minimale Anzahl von „Option Templates“ verwendet, die alle möglichen Kombinationen der Testfälle „Erstellen“ und „Löschen“ eines „Option Templates“ erfasst.

Außerdem wurde ein zusätzlicher Testfall aufgenommen. Dieser verlangt die Überprüfung, ob ein Implantatstyp aus der selben Implantatsfamilie das erstellte „Option Template“ anderer Implantatstypen der Implantatsfamilie weder sehen noch annehmen kann.

Die vorgenommenen Änderungen sind auf der beiliegenden CD einsehbar.

4.2 Tracing durch Werkzeugsteuerung

Durch das vorgegebene Muster liegt die gleiche Struktur sowohl in den TestSetups als auch in den TestDesigns vor. Bei der exemplarischen Umsetzung des Tools RepFram macht sich das RepFram die entwickelte Musterstruktur zu Nutze und sendet nach jedem Testlauf die Ergebnisse der HMSC3-Val-Tests automatisch in das Test-Management-Tool (TMT). Diese automatisierte Übertragung der Testergebnisse in das TMT ist erst mit der Version 11, dem Application Lifecycle Management (ALM), möglich, da erst mit dieser Version die genutzte REST-Schnittstelle zur Verfügung steht. Deshalb wird ab diesem Kapitel statt QC der Begriff ALM verwendet. Das Ziel der Implementierung ist es, durch das RepFram und das zu Grunde liegende Muster, die Nachverfolgbarkeit der Anforderungen werkzeugtechnisch sicherzustellen. Im Folgenden wird der Ablauf

vorgestellt, wie er aktuell umgesetzt ist und es wird dargestellt, wie dieser nach der Implementierung geplant ist. Es werden die Begriffe aus JavaTT und QC/ALM verwendet, wie sie in Abschnitt 2.2.2.2 und 2.2.3 eingeführt wurden.

4.2.1 Ablauf

Wie bereits in Abschnitt 1.1 beschrieben, werden bei der Planung neuer Tests in ALM zunächst logische Testfälle in einem TestDesign erstellt. Die im XML-Format implementierten konkreten Testfälle werden mittels JavaTT regelmäßig ausgeführt. Dieser Ablauf ist in Abbildung 8 dargestellt. Der Tester lässt die HMSC3-Val-Tests mittels Test-Tool (JavaTT) durchlaufen. Das Ergebnis der automatisierten Testläufe ist jeweils eine Datei (z.B. im Format XLS, XML, CSV). Alle Ergebnisse werden manuell bewertet und anschließend mit Hilfe des TMT verwaltet und dokumentiert. Bereits diese manuelle Bewertung der Testergebnisse wirkt sich bei einer erhöhten Testanzahl negativ auf die Fehleranfälligkeit und den Zeitaufwand aus. Des Weiteren ist durch die Erstellung der logischen Testfälle in ALM und die manuelle Erstellung der TestSetups keine Nachverfolgbarkeit der Anforderungen sichergestellt. Um dies zu gewährleisten und die manuelle Bewertung der Ergebnisse zu vermeiden, wird in den bestehenden Ablauf das RepFram eingebunden (siehe Abbildung 9).

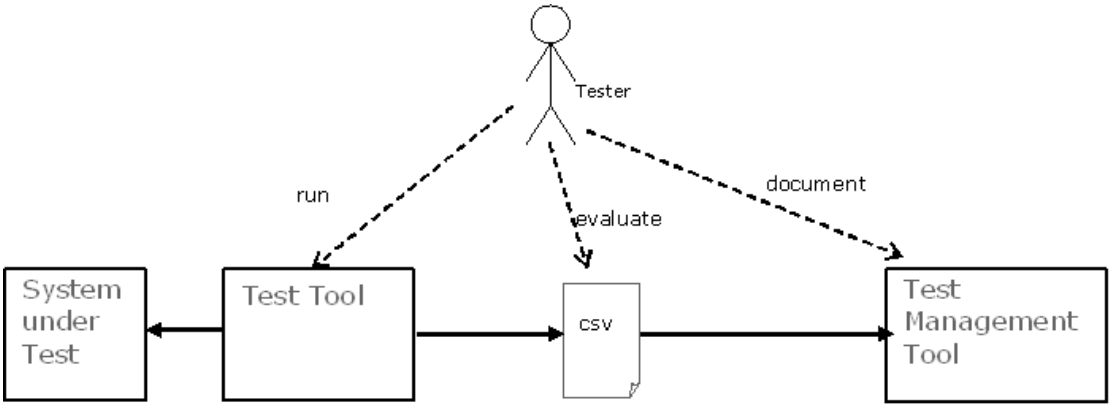

Abbildung 8: Momentaner Testablauf der HMSC3 Val-Tests

Im neuen Ablauf startet der Tester lediglich die HMSC3-Val-Tests über JavaTT. Der weitere Ablauf wird ohne manuellen Eingriff durchgeführt. Das Test-Tool ruft das RepFram über eine Javamethode auf. In diesem Aufruf werden neben weiteren Parametern die Ergebnismengen des aktuellen Testlaufs mitgegeben. Die Ergebnismengen werden im RepFram geeignet verarbeitet.

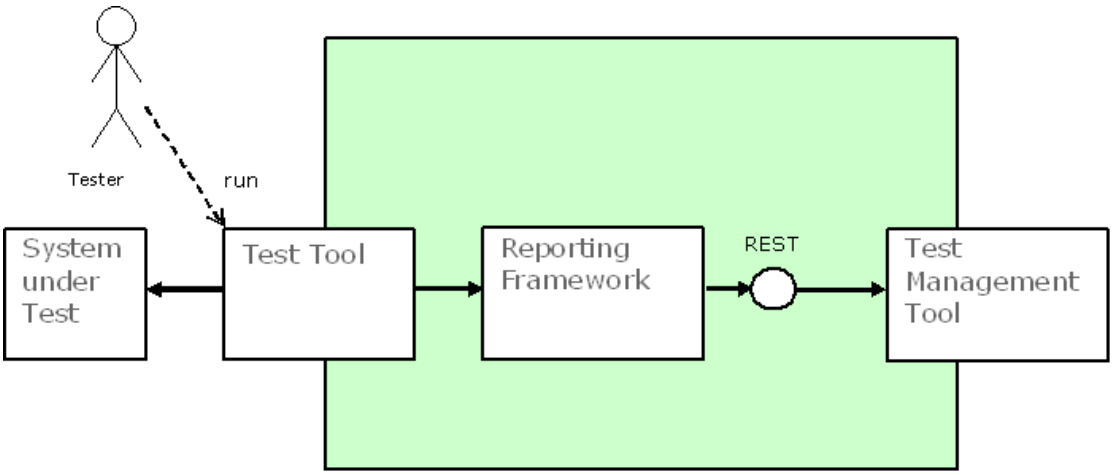

Abbildung 9: Ablauf mit integriertem RepFram

Das TMT – ALM – stellt eine REST-Schnittstelle zur Verfügung, um Operationen an den Elementen von ALM zu ermöglichen. Über diese Schnittstelle schickt das RepFram alle verarbeiteten Ergebnisse und dokumentiert sie in ALM.

Bei der Implementierung bestand der Wunsch der Validierungsgruppe, dass das RepFram unabhängig vom Test-Tool entwickelt wird. Die Nutzung anderer Test-Tools ist in der Umsetzung des RepFrams berücksichtigt[6]. In der Beschreibung der Voraussetzungen und den erläuterten Implementierungsdetails wird davon ausgegangen, dass das RepFram nur durch JavaTT genutzt wird.

Im nächsten Abschnitt werden zunächst die Voraussetzungen beschrieben, die für den Aufruf des RepFram und die Ergebnisübertragung nach ALM notwendig sind.

4.2.2 Voraussetzungen

4.2.2.1 Vorbereitungen durch den Tester

Bevor ein Testlauf mit dem eingebundenen RepFram gestartet werden kann, müssen folgende Vorbereitungen getroffen werden. Es müssen:

- der Name des TestSetups und Name des TestDesigns müssen übereinstimmen,
- ein Attribut `AlmPath` im TestSetup angelegt werden,
- die Namen der Scenarios und Namen der TestSteps im TestDesign müssen übereinstimmen und
- es muss eine ini-Datei angelegt werden.

[6] In einem ersten Ansatz der Validierungsgruppe kann das RepFram bereits mit Testskripten in Python gestartet werden.

Die ersten zwei vorbereitenden Änderungen sind notwendig, um den laufenden Test mit dem zugehörigen TestDesign in Verbindung zu bringen. Der Name des TestSetups – sowohl Datei- als auch Attributsname – muss dem Namen des TestDesigns entsprechen, wie im Listing 3 und der Abbildung 10 zu sehen ist. Dies ist notwendig, um die TestInstance zu identifizieren. Die TestInstance ist innerhalb von ALM mit dem TestDesign verknüpft und am gleichen Namen erkennbar.

```
1  <TestSetup Name="Alm"  AlmPath="TestsetFolder/Testset">
2    <SetupEntry>
3      <!-- Login -->
4      <Scenario Name="SCN_HMSC3_Platform/Login">
5        <!-- Parameters which are necessary for the login -->
6        <Parameter Name="UserGroup"     Value="System"/>
7        <Parameter Name="UserName"      Value="PreviewCSS"/>
8        <Parameter Name="Password"      Value="test12"/>
9        <Parameter Name="url" Value="www.testsystem-homemonitoring.int"/>
10     </Scenario>
11     <!-- Deactivate Monitoring -->
12     <Scenario Name="SCN_PatientAdministration/PA_DeactivateMonitoring">
13       <Parameter Name="Patient" Value="Val-60400025" />
14     </Scenario>
15   </SetupEntry>
16 </TestSetup
```

Listing 3: TestSetup mit Änderungen

Das Attribut `AlmPath` wird im TestSetup eingefügt (siehe Listing 3). Es spezifiziert den Pfad zum TestSet in ALM, in dem die TestInstance liegt, in der später ein Run erzeugt wird (siehe Abbildung 10). Die Angabe des Pfades ist zusätzlich zum Namen des TestDesigns notwendig, weil immer mehrere TestInstances mit einem TestDesign verknüpft sein können. Darüber hinaus sind die Namen der TestSets und TestSetFolder nicht immer eindeutig. Eine eindeutige Identifizierung der TestInstance ist daher nur über eine vollständige Pfadangabe möglich.

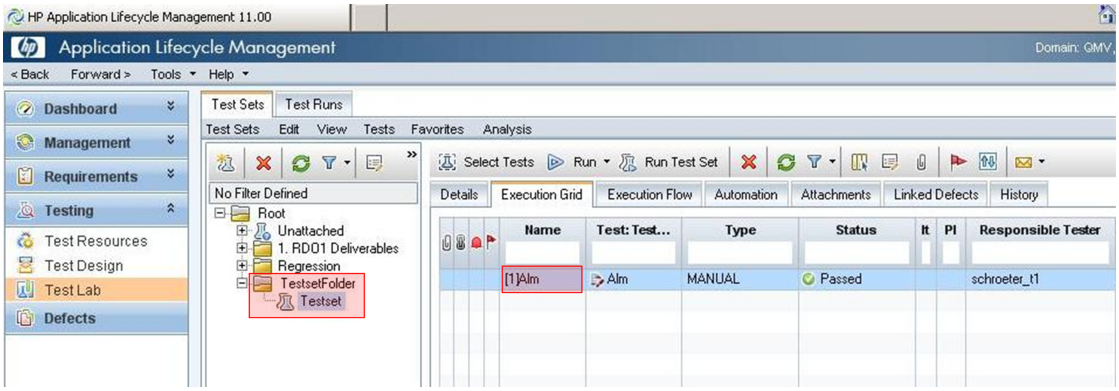

Abbildung 10: ALM – Pfad zum TestSet und Name des TestDesigns hervorgehoben

Die dritte vorbereitende Maßnahme wird bereits über das implementierte Muster der TestSetups und TestDesigns sichergestellt. Sollte die eindeutige Zuordnung vom Scenario zum TestStep über den Namen nicht umgesetzt sein, gibt es unterschiedliche

Fehlerbehandlungen des RepFrams, auf die später eingegangen wird.

Jeder Tester legt lokal eine INI-Datei in folgender Form an:

```
;alm
[alm]
tester= schroeter_t1
encUserPW = VXNlcm5hbWU6UGFzc3dvcmQ=
url =http://hpqc1-2:8080/qcbin
domain = QMV
project =ALM_HMSCTest
attachment = /attachments
```

Über diese Datei erhält das RepFram weitere benötigte Informationen. Dazu zählen neben dem Namen des Testers (`tester`) das mit Base64 kodierte `encUserPW` in der Form „Nutzername:Passwort“. Des Weiteren sind die Url (`url`), unter der das TMT zu finden ist und die Domäne (`domain`) sowie das Projekt (`project`) anzugeben[7]. Die letzte Angabe unter `attachment` spezifiziert den Pfad zum Speicherplatz für etwaige Anhänge an den TestRun in ALM.

Die Angabe des verschlüsselten Passwortes genügt einer ersten exemplarischen Umsetzung, sollte jedoch in einer nächsten Version durch eine geeignete Variante ersetzt werden. Nach Abstimmung mit den Verantwortlichen bei Biotronik wird das bestehende Sicherheitsrisiko als gering eingestuft. Die INI-Datei ist nur auf dem Arbeitsplatzrechner eines jeden Testers abgelegt. Gleichzeitig erfolgt der Vorgang der Authentifizierung im gesicherten Netzwerk des Unternehmens. Die gewählte Form der Kodierung verhindert das versehentliche Erfassen des Passwortes durch einen anderen Nutzer. Weiterhin ist eine direkte Weitergabe des Wertes an ALM möglich, da die verwendete Kodierung vom System erwartet wird.

4.2.2.2 Voraussetzungen durch das Test-Tool – JavaTT

Das RepFram benötigt vom Test-Tool die Ergebnismengen des Testlaufs, um diese zu verarbeiten, bevor sie ins ALM gesendet werden. Diese umfassen sowohl die Ergebniswerte als auch die -Nachrichten (siehe Tabelle 1 im Abschnitt 2.2.2.2).

Wie in dieser Tabelle zu sehen ist, sind die Ergebnisse eines Testlaufs sehr detailliert angegeben. Für jedes ausgeführte TestItem gibt es ein Ergebnis, das einen der folgenden Werte annehmen kann: INFO, FINE, FINER, FINEST, Min Value, Max Value, WARNING, ERROR, not initialized oder FATAL. Um diese Ergebnisse so zu strukturieren, dass sie später zu einem Ergebnis pro TestStep zusammengefasst

[7] In ALM sind Tests in Projekten zusammengefasst, die wiederum durch Domänen strukturiert sind.

werden können, muss eine geeignete Datenstruktur gefunden werden. Diese soll eine Zuordnung der einzelnen Ergebnisse zu den zugehörigen Scenarios enthalten. Die gewählte Datenstruktur ist eine `MultiValueMapArrayList`[8]. Dies ermöglicht eine Speicherung der Ergebnisse zu jedem Scenario in Ausführungsreihenfolge. Das heißt, pro Scenario können mehrere Ergebnisse als Liste abgespeichert werden. Nachfolgend ist ein Beispiel für die Speicherung der Ergebniswerte für das Scenario „Login“ aufgeführt :

```
Login=[INFO, ERROR, FINE, ERROR, ERROR, WARNING, WARNING, WARNING]
```

Ebenso wird vom Test-Tool die boolsche Variable `ReportingFlag` erwartet. Diese zeigt an, ob das RepFram die Ergebnisse nach ALM übertragen soll. Wird das ReportingFlag auf „false“ gesetzt, verarbeitet das RepFram die Ergebnismenge des Testlaufs ohne eine Verbindung mit ALM aufzubauen. Diese Variante kann bei Probeläufen eines Tests verwendet werden oder zur lokalen Prüfung der Ergebnismengen.

4.2.3 Implementierungsdetails

Im Rahmen dieser Arbeit wurden keine Tests zur Verifikation und Validierung des RepFrams erstellt, da Tests gemäß gängiger Praxis nicht vom Entwickler durchgeführt werden sollten. Alle sonstigen Dateien mit Quellcode sind auf der beigelegten CD zu finden.

Nachfolgend wird auf die wichtigsten Inhalte der Implementierung des RepFram eingegangen. In Abbildung 11 ist die Systemübersicht dargestellt. Die Elemente der Test-Management-Tool-Komponente sind die bereits bekannten Elemente aus ALM. In der Komponente des Test-Tools sind alle Parameter zu sehen, die beim Aufruf des RepFrams durch das Test-Tool mitgegeben werden. Die dargestellten Elemente der Komponente des RepFrams sind die erstellten Klassen. Ein Klassendiagramm mit Erläuterungen zu den Inhalten der Klassen ist im Anhang unter der Abbildung B.1 zu finden.

[8] von Wiztools:
`http://www.jarvana.com/jarvana/view/org/wiztools/commons/wiztools-commons-lib/0.2.0/wiztools-commons-lib-0.2.0-javadoc.jar!/org/wiztools/commons/MultiValueMapArrayList.html`

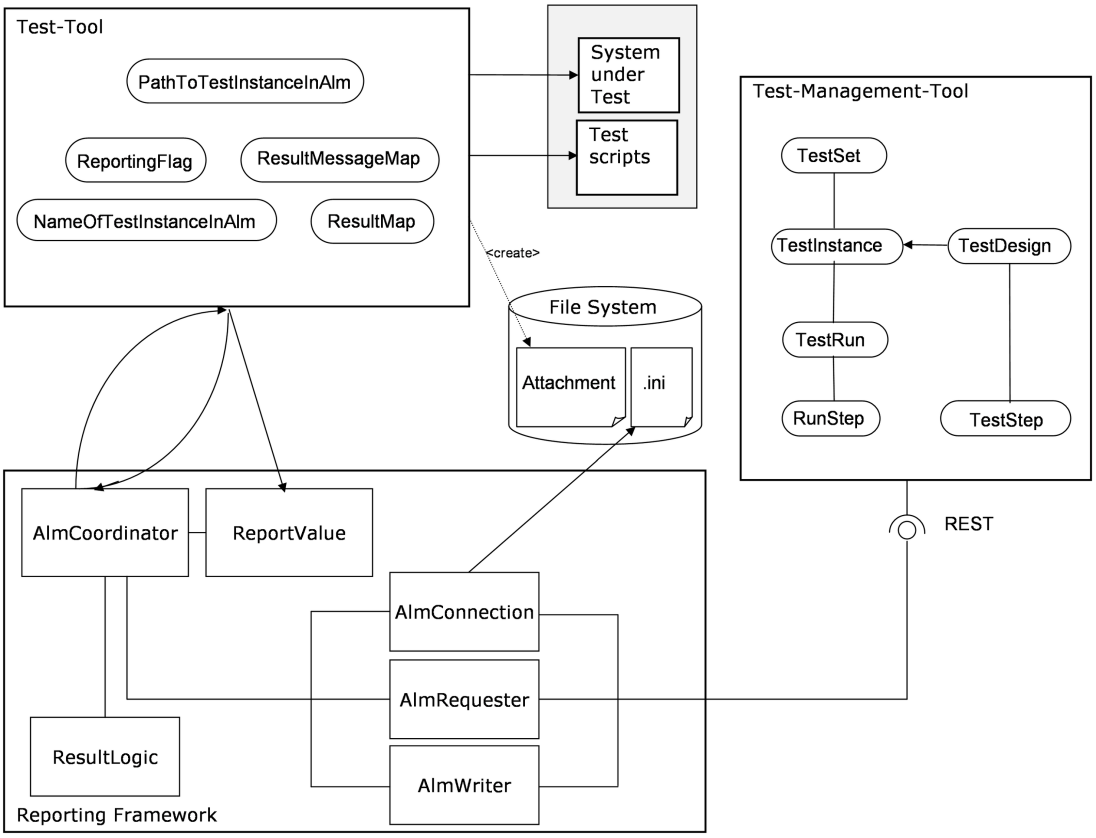

Abbildung 11: Systemübersicht

Der Ablauf innerhalb des RepFram unterteilt sich inhaltlich grob in vier Phasen, die den Klassen zugeordnet werden können:

- Verbindung mit ALM (`AlmConnection`),
- Anfragen der zur Identifizierung benötigten Elemente (`AlmRequester`),
- Ergebnisverarbeitung (`ResultLogic`) und
- Übertragen der Ergebnisse ins ALM (`AlmWriter`).

In der ersten Phase wird zur Vorbereitung des Verbindungsaufbaus die INI-Datei ausgelesen. Damit hat das RepFram alle benötigten Informationen, um die Verbindung zu ALM aufzubauen. Anschließend erfolgt die Authentifizierung mit dem `encUserPw` aus der INI-Datei. Nach erfolgreicher Anmeldung beim Server wird ein Cookie zurückgesendet. Dieses wird bei jeder weiteren Anfrage mitgegeben, um den Nutzer zu identifizieren.

In der zweiten Phase werden alle Elemente von ALM angefragt, die zur Ergebnisübertragung notwendig sind. Zum Beispiel werden die einzelnen TestSetFolder sowie das TestSet ermittelt, sodass die richtige TestInstance identifiziert werden kann. Weiterhin fragt das RepFram von ALM eine Liste aller TestStep-Namen ab, über welche

später die Zuordnung zu den Scenarios erfolgt.

In Phase 3 werden die Ergebnismengen, die von JavaTT übergeben wurden, so aufbereitet, dass es für jeden TestStep ein Ergebnis gibt.

Zur Phase 4 gehört neben der Übertragung der Ergebnisse im Vorfeld die Erstellung eines TestRuns in der TestInstance. Die Übertragung der Ergebnisse erfolgt anschließend schrittweise pro TestStep. Falls im angegebenen Ordner der INI-Datei Anhänge durch das Test-Tool abgelegt wurden, werden diese ebenfalls übertragen und sind über den TestRun in ALM zu finden.

Das RepFram liefert einen Rückgabewert über den Zustand des Übertragungsprozesses bei Beendigung des RepFrams an JavaTT. Dieser Rückgabewert wird in der Klasse `ResultValue` definiert und kann je nach aufgetretenem Problem während der Übertragung oder bei erfolgreichem Ablauf folgende Werte annehmen:

- NO_INI, wenn keine INI-Datei gefunden wurde oder in der INI-Datei Fehler enthalten waren,
- NO_SESSION, wenn keine Verbindung mit dem Server von ALM aufgebaut werden konnte,
- NO_RUN, wenn kein TestRun erstellt werden konnte,
- NO_RESULTS, wenn keine Ergebnisse übertragen werden konnten,
- NO_ATTACHMENT, wenn der angegebene Ordner für Anhänge nicht gefunden werden konnte und
- FINE, wenn die Übertragung erfolgreich war.

Durch diesen Rückgabewert kann JavaTT bei Bedarf weitere Aktionen ausführen. Zudem hat der Tester die Möglichkeit, bei Problemen die Ursache im Ablauf zu finden. Alle Fehlermeldungen, die zum jeweiligen Rückgabewert führen, werden immer in eine LOG-Datei geschrieben.

Für Einblicke in den detaillierten Ablauf des RepFram ist im Anhang unter Abbildung B.2 ein Sequenzdiagramm dargestellt.

4.2.3.1 REST-Anfragen

Da die Kommunikation mit dem Server von ALM über die vorhandene REST-Schnittstelle erfolgt, wird an dieser Stelle ein kurzer Überblick über die Nutzung der Schnittstelle gegeben. Die Schnittstelle von ALM ermöglicht das Hinzufügen, Auslesen, Ändern und Löschen von Elementen. Folgende Elemente und deren

Attribute/Felder werden während der Ausführung des RepFrams angefragt, hinzugefügt oder verändert.

Tabelle 7: Übersicht der ALM-Elemente

Elemente	Schnittstellenname	Attribute und deren Bedeutung	
TestDesign	test	id	ID des TestDesigns
		name	Name des TestDesigns
TestSetFolder	test-set-folder	id	ID des TestSetFolders
		parent-id	ID des Elternelements (falls vorhanden)
TestSet	test-set	id	ID des TestSets
		parent-id	ID des TestSetFolders
TestInstance	test-instances	id	ID der TestInstance
		cycle-id	ID des TestSets
		test-id	ID des TestDesigns
		owner	Nutzername des verantwortlichen Testers
TestRun	run	id	ID des TestRuns
		cycle-id	ID des TestSets
		test-id	ID des TestDesigns
		testcycl-id	ID der TestInstance

Das RepFram nimmt an, dass das TestDesign und die Ordnerstruktur bis hin zur TestInstance bereits angelegt sind. Für diese Elemente werden lediglich die IDs abgefragt. Zusammen mit der vollständigen Pfadangabe, die im TestSetup angegeben wird, kann über die ID des Elternelements die TestInstance identifiziert werden. In der TestInstance wird vom RepFram der TestRun angelegt. Anschließend werden die Ergebnisse in die einzelnen RunSteps eingetragen.

An dieser Stelle wird beispielhaft eine GET-Anfrage dargestellt, wie sie im Browser oder per Java ausgeführt werden kann. Die Nutzung der REST-Schnittstelle im Browser ist ebenfalls erst nach erfolgreicher Authentifizierung und nur in der selben Instanz des Browsers möglich.

Der folgende REST-Befehl stellt eine GET-Anfrage des TestRuns mit der ID 77 im Projekt „ALM_HMSCTest“ und der Domäne „QMV“ im Browser dar.

```
http://hpqc1-2:8080/qcbin/rest/domains/QMV/projects/ALM_HMSCTest/runs/77
```

Nach der erfolgreichen Anfrage des obigen Befehls wird der folgende XML-Ausschnitt im Browser angezeigt:

```xml
<xml version="1.0" encoding="UTF-8" standalone="yes" >
<Entity Type="run">
  <Fields>
    <Field Name="test-instance"><Value>1</Value></Field>
    <Field Name="id"><Value>77</Value></Field>
    <Field Name="name"><Value>run</Value></Field>
    <Field Name="testcycl-id"><Value>185</Value></Field>
    <Field Name="cycle-id"><Value>102</Value></Field>
    <Field Name="status"><Value>Failed</Value></Field>
    <Field Name="test-id"><Value>402</Value></Field>
    <Field Name="owner"><Value>schroeter_t1</Value></Field>
  </Fields>
</Entity>
```

Listing 4: Ergebnisanzeige im Browser

Die Ausführung einer REST-Operation im Browser ermöglicht einen ersten Einblick in die Strukturen der Schnittstelle. Der gleiche Befehl in Java sieht folgendermaßen aus:

```java
pProjectResource.path("runs/77")
  .cookie(pCookie)
  .header("Cookie", cookieString)
  .type(MediaType.APPLICATION_XML_TYPE)
  .accept(MediaType.APPLICATION_XML)
  .get(ClientResponse.class);
```

Listing 5: GET-Anfrage in Java

Der erste Teil dieser Anfrage entspricht genau der URL, wie sie zuvor dargestellt wurde und ist in der Variablen `pProjectResource` enthalten. Durch den Aufruf von `.path("runs/77")` wird die URL vervollständigt. Für die Anfrage über Java sind, wie zu erkennen ist, zusätzliche Informationen bei einer Anfrage mitzugeben. Zum einen ist die Übergabe zweier Cookies notwendig, die den Nutzer und die aktuelle Sitzung identifizieren. Zum anderen ist die Übertragung des Datentyps, den der Server erwarten und zurückgeben soll, erforderlich. Die Abfrage der `ClientRespone.class` gibt neben dem Erhalt der Antwort-XML auch die HTTP-Antwort des Servers zurück.

Die Kommunikation mit dem Server erfolgt in Java über Jersey[9]. Alle Operationen, die das RepFram zur Übertragung der Ergebnisse ausführt, sind im Quellcode in den Klassen `AlmRequester` und `AlmWriter` einzusehen (siehe beigelegte CD).

4.2.3.2 Ergebnisverarbeitung

Bevor das RepFram die Ergebnisse ins ALM übertragen kann, müssen die Ergebnismengen von JavaTT verarbeitet werden. Durch die gewählte Datenstruktur für die Ergebnismengen gibt es für jedes Scenario eine Liste von Ergebnissen. Die Listen und der Scenarioname sind in einer `Map` abgelegt. Dabei ist der Scenarioname der

[9] `https://jersey.java.net/`

Schlüssel der Schlüssel-Werte-Paare. Zunächst wird für jedes Scenario in der `Map` die Liste mit den Ergebniswerten durchsucht. Ist ein Ergebnis nicht positiv – WARNING oder ERROR – wird das Gesamtergebnis für das Scenario auf den Wert „Failed“ gesetzt. Diese neue Ergebnismenge wird wiederum in einer `Map` abgespeichert.

Für die Übertragung ins ALM bedeutet dies: Sobald ein Scenario bei der Auswertung der Ergebnisse den Status „Failed“ erhält, wird der Status des zugehörigen TestSteps ebenfalls „Failed“. Ist ein TestStep „Failed“, wird auch der Status des gesamten TestRuns „Failed“.

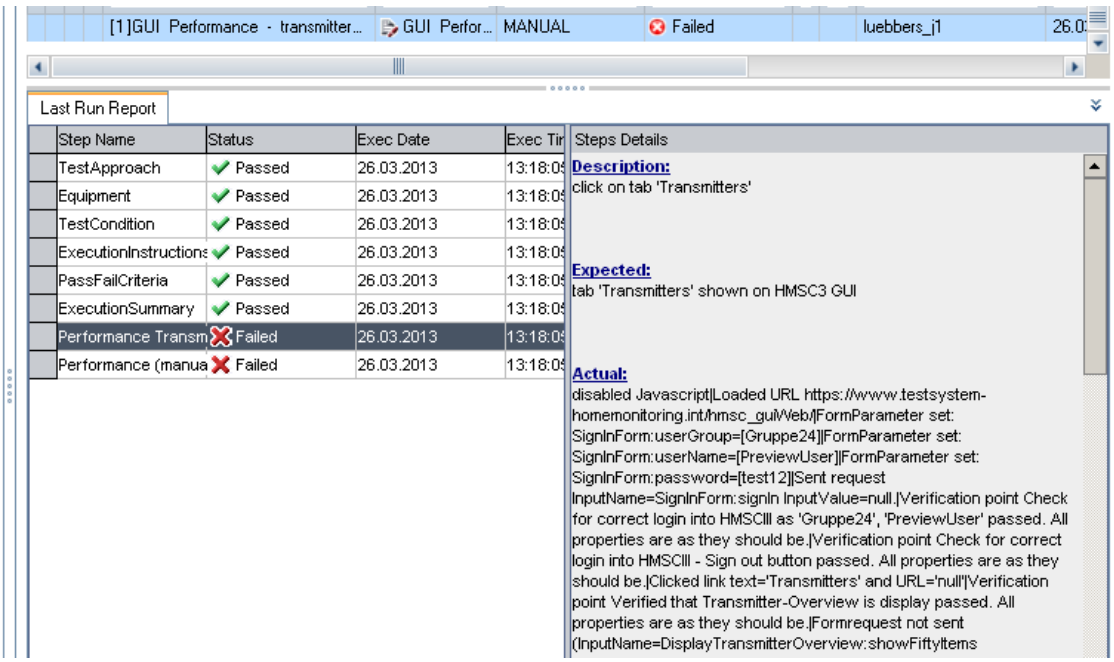

Abbildung 12: TestItem im Scenario liefert Warning – TestStep „Failed“

Neben den Ergebniswerten übergibt JavaTT beim Aufruf des RepFrams auch die Ergebnisnachrichten. Diese werden, wie in der Abbildung 12 zu sehen ist, in das „Actual“-Feld geschrieben, wenn der Status des TestSteps „Failed“ ist. So hat der Tester die Möglichkeit, in ALM festzustellen, an welcher Stelle im Skript der Fehler aufgetreten ist. Für die detaillierte Fehleranalyse bietet JavaTT aber weiterhin die CSV-Datei, die derzeit noch zur manuellen Ergebnisbewertung verwendet wird.

Sind alle Ergebniswerte in den Scenarios positiv, erhält der zugehörige TestStep den Status „Passed“. Das „Actual“-Feld erhält den Wert aus dem Feld „Expected“, also dem Wert, der bei der Erstellung des TestDesigns als erwartetes Ergebnis festgelegt wird. Haben alle TestSteps den Status „Passed“, dann erhält auch der TestRun den Status „Passed“.

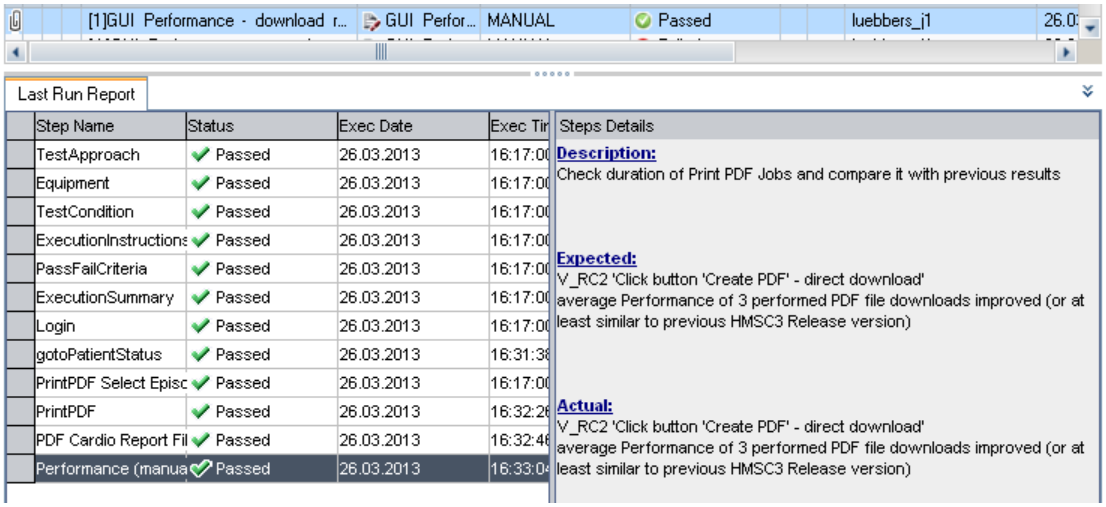

Abbildung 13: Default TestSteps – Status „Passed“

In der Abbildung 13 sind neben den TestSteps, die zu einem Scenario gehören, Default-Steps zu sehen. Zu diesen gehören die TestSteps mit dem Namen: „Equipment“, „TestApproach“, „TestCondition“, „ExecutionInstructions“, „PassFailCriteria“ und „ExecutionSummary“. Alle Default-Steps erhalten automatisch den Status „Passed“. Gleichfalls werden alle TestSteps, die eine Anforderung beschreiben, vom RepFram gesondert betrachtet und erhalten den Status „Passed“.

Bei der Ergebnisverarbeitung wird neben der Prüfung der Ergebnismengen die Zuordnung vom Scenario zum TestStep geprüft. Alle TestSteps, die über kein zugehöriges Scenario verfügen, erhalten den Status „Failed“. In das Feld „Actual“ wird eingetragen, dass der jeweilige TestStep nicht implementiert wurde (siehe Abbildung 14).

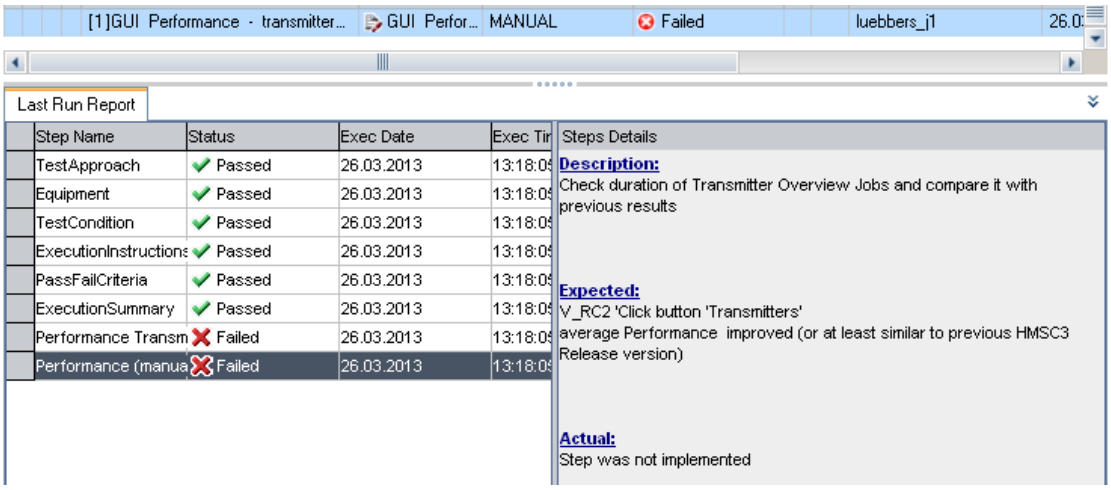

Abbildung 14: TestStep ohne zugehöriges Scenario – TestStep „Failed“

Für alle Scenarios, die keinen zugehörigen TestStep in ALM haben, wird ein neuer RunStep an das Ende des TestRuns angelegt. Dieser erhält ebenfalls den Status „Failed“. In das Feld „Actual“ des neuen RunSteps wird eingetragen, dass es keinen passenden

TestStep (bzw. RunStep) gab (siehe Abbildung 15).

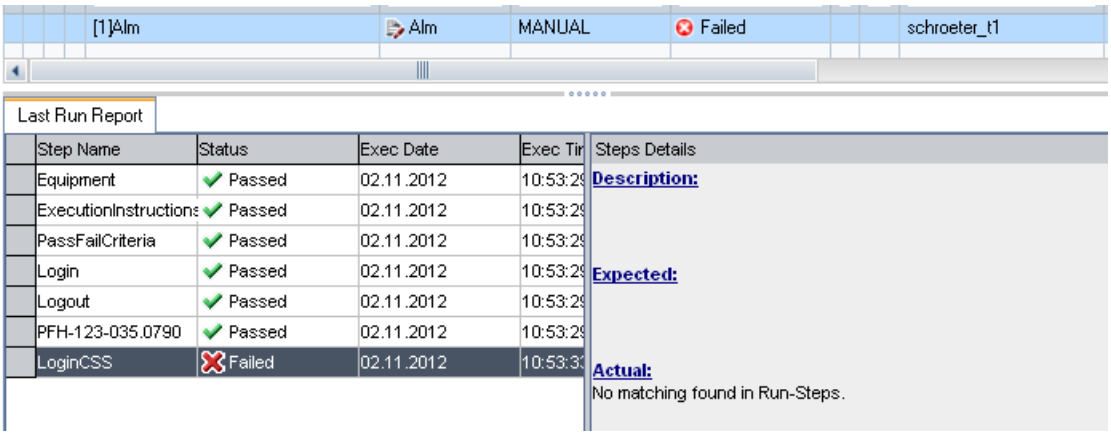

Abbildung 15: Scenario ohne zugehörigen TestStep – neuer TestStep „Failed“

5 Bewertung der erreichten Verbesserungen

In diesem Kapitel werden die Verbesserungen anhand des Tests „Patient Option Template“ bewertet. Es wird überprüft, inwiefern die Zielvorgaben für Verbesserungen erreicht wurden.

Nach der Implementierung sind folgende Regeln und Zielvorgaben umgesetzt und automatisiert prüfbar:

- Die Namenskonvention, dass das TestDesign und das TestSetup den gleichen Namen besitzen,
- die Zuordnung der TestSteps zu den zugehörigen Scenarios,
- die Aufdeckung nicht implementierter TestSteps und
- die Aufdeckung nicht spezifizierter Scenarios.

Die automatisierte Prüfung dieser Regeln ist durch die Umsetzung des RepFrams möglich. Durch die Zuordnung von TestDesign und TestSetup kann der Tester nach jedem Testdurchlauf feststellen, ob es TestSteps gibt, die zuvor nicht implementiert wurden, oder Scenarios, die zuvor nicht spezifiziert wurden. Die Metriken 1.2.1, 1.2.2, 2.2.1 und 2.2.2 sowie zum Teil die Metrik 3.2.1 können durch die erreichte Umsetzung ebenfalls automatisiert gemessen werden. Für die Metrik 3.2.1 ist nur der Teil 2 der Bewertung der Namensgebung mit einer positiven Wertung sichergestellt. Eine Übertragung der Ergebnisse funktioniert nur, wenn der Name des TestDesigns mit dem Namen des TestSetups übereinstimmt.

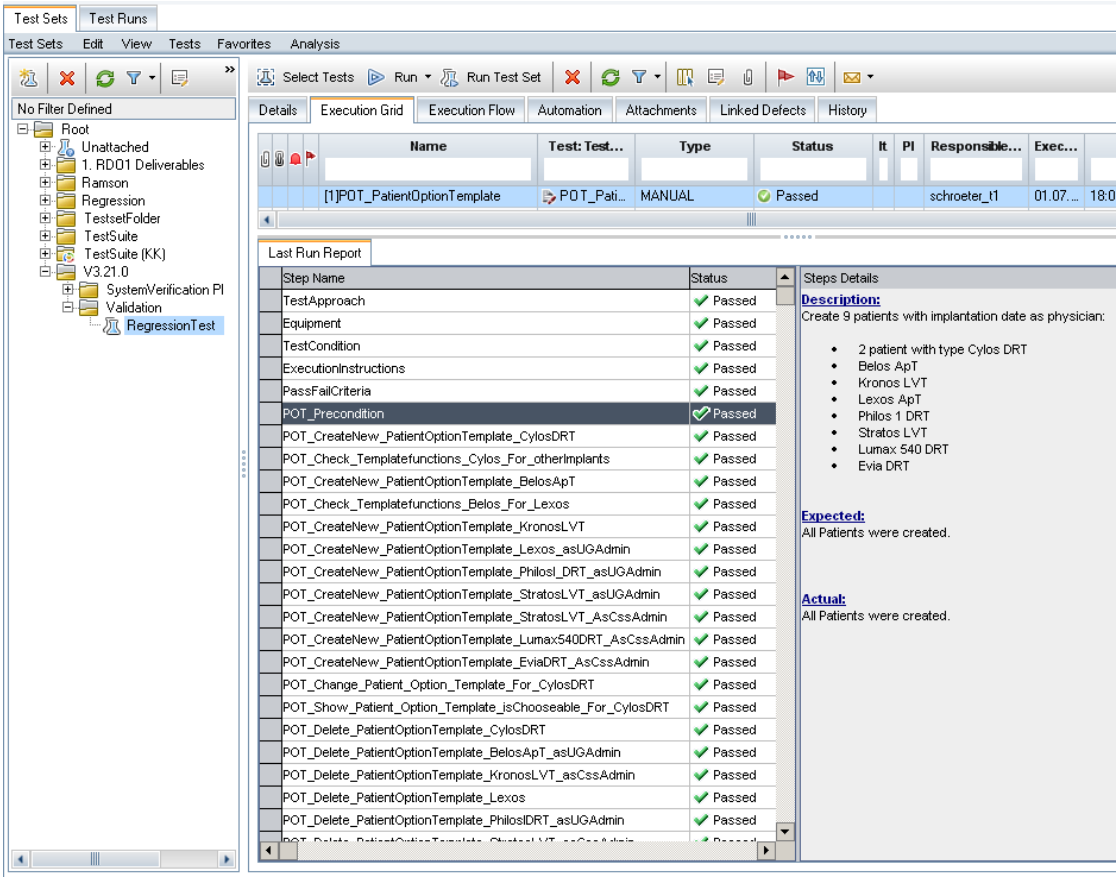

Abbildung 16: Erfolgreicher Testdurchlauf mit Ergebnisübertragung

In der Abbildung 16 ist zu sehen, dass die Ergebnisse des Testlaufs vom Test „Patient Option Template“ vollständig nach ALM übertragen werden konnten. Durch die Prüfung der Zuordnung zwischen TestSteps und Scenarios ist bei diesem Test – da der Test den Status „Passed“ hat – sichergestellt, dass eine vollständige Testabdeckung besteht. Des Weiteren ist die Nachverfolgbarkeit der Anforderungen sichergestellt. Alle TestSteps, die eine Anforderung testen, haben bei dem Testdurchlauf den Status „Passed“ erhalten.

Die Metriken in Tabelle 8 sind weiterhin nicht automatisiert prüfbar. Obwohl ein paar der hier betrachteten Metriken nicht durch konkrete Zielvorgaben erfasst wurden, sind Verbesserungen erreicht worden. Die dargestellten Metriken wurden erneut mittels statischer Analyse ermittelt.

Tabelle 8: Bewertung der erreichten Verbesserungen
Metrik 1.1.1 – Verh. Sc./TestSteps, Metrik 1.1.2 – Verh. aller Sc.-Aufrufe/TestSteps,
Metrik 1.3.1 – Anz. aller Sc.-Aufrufe, Metrik 1.3.2 – Anz. Sc.,
Metrik 1.3.3 – Anzahl TestSteps,
Metrik 2.1.1 – Fehlerquotient,
Metrik 2.3.1–2.3.3 – min, max, mittlere Anz. Actions pro Scenario,
Metrik 2.3.4–2.3.6 – min, max, mittlere Anz. TestCases pro Scenario,
Metrik 2.3.7–2.3.9 – min, max, mittlere Anz. TestItems pro Scenario,
Metrik 3.2.1 – Bewertung der Namensgebung

Ziel 1	Patient Option Template (vorher)	Patient Option Template (nachher)
Metrik 1.1.1	0,1	1
Metrik 1.1.2	0,9	1,96
Metrik 1.3.1	110	47
Metrik 1.3.2	13	24
Metrik 1.3.3	119	24
Ziel 2		
Metrik 2.1.1	0,9	0
Metrik 2.3.1	0	4
Metrik 2.3.2	1	11
Metrik 2.3.3	0	6
Metrik 2.3.4	1	5
Metrik 2.3.5	9	16
Metrik 2.3.6	2	7
Metrik 2.3.7	0	0
Metrik 2.3.8	51	14
Metrik 2.3.9	12	5,5
Ziel 3		
Metrik 3.2.1. (Teil 1)	--	++
Metrik 3.2.1. (Teil 2)	--	++
Metrik 3.2.1. (Teil 3)	-	+

Die Metriken für Ziel 1 zeigen, dass sich das Verhältnis der Scenarios zu den TestSteps verbessert hat. Das Verhältnis der Scenario-Aufrufe zu den TestSteps (Metrik 1.1.2) hat sich durch den Umbau des Tests verschlechtert. Dies liegt an der schlechten Parametrisierung in JavaTT. Um alle konkreten Testfälle zu durchlaufen, müssen einige Scenarios aus dem TestSetup mehrfach aufgerufen werden. So muss das Scenario „Precondition“ neun mal aufgerufen werden, um alle benötigten Patienten anzulegen.

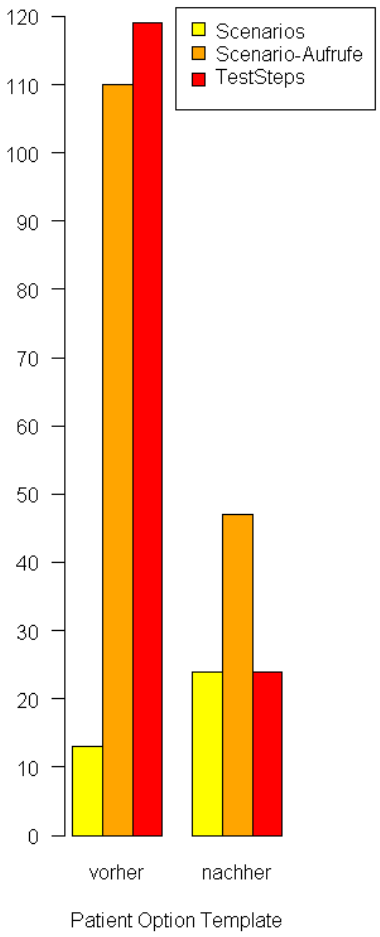

Abbildung 17: Anzahl der Scenarios, Scenario-Aufrufe und TestSteps – vorher/nachher

Gleichfalls konnte – wie in Abbildung 17 zu sehen ist – die Anzahl der Scenarios, der Scenario-Aufrufe und der TestSteps deutlich verringert werden. Die Komplexität der aufgerufenen Scenarios im Test „Patient Option Template“ konnte verringert werden. Die Anzahl der TestCases ist bei gleichzeitig geringerer Anzahl an TestItems gestiegen.

Abbildung 18: Anzahl der TestCases in den Scenarios (nachher)

Die Streuung der Datenmenge für die TestCases und TestItems zeigt, dass es auch keine Ausreißer gibt. Die Datenpunkte sind gleichmäßig verteilt, sodass auch der Umfang eines Scenarios gut abzuschätzen ist.

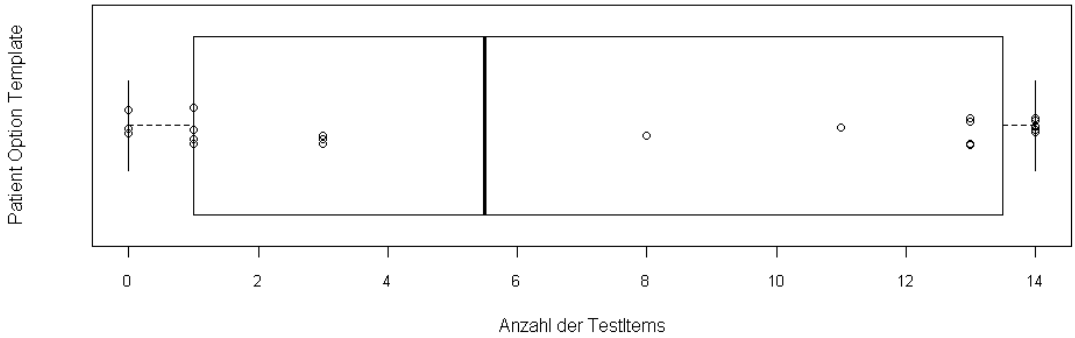

Abbildung 19: Anzahl der TestItems in den Scenarios (nachher)

Ein Grund für den Anstieg der Anzahl der TestCases ist die Anzahl der aufgerufenen Actions. Da jede aufgerufene Action auch einem TestCase entspricht, ist die Anzahl der TestCases pro Scenario gestiegen. Durch die Auslagerung von Abläufen aus den Scenarios in Actions konnte die Anzahl der TestItems pro Scenario verringert werden.

Tabelle 9: Bewertung der erreichten Verbesserungen der Metrik 3.1.1 für Ziel 3
Metrik 3.1.1 – Anzahl der Fan-Ins

Ziel 3	Check_Patient_Option_Template	CheckApplyButtonEnabled	Fill_Login	CheckConfirmButtonEnabled	Click_Menu_AllPatient	Select_Patient	Deactivate_Monitoring	Delete_Patient	Delete_Template	FillinPatientData	SavePDF	Sign_Out
Metrik 3.1.1	22	20	24	12	33	33	1	1	9	1	10	24

Die Anzahl der Aufrufe einer Action liegt zwischen vier und elf Actions pro Scenario für den verbesserten Test „Patient Option Template“ (siehe Tabelle 8 Metrik 2.3.1–2.3.3). Dabei ist der mehrfache Aufruf einer Action in dieser Wertung beinhaltet. Im Dateisystem der Regressionstests sind nach der Verbesserung zwölf Actions vorhanden. Beim Umbau weiterer Tests sind zudem weitere Actions zu erwarten. Für die in dieser Arbeit erstellten Actions sind in der Tabelle 9 die Gesamtanzahl der Aufrufe aus allen Scenarios dargestellt. Neben der hohen Anzahl an Aufrufen von Actions ist

zu betonen, dass alle 24 Scenarios des Tests „Patient Option Template“ mindestens eine Action aufrufen. Die Zielvorgabe einer Modularisierung der Testskripte wurde durch das entwickelte Konzept erreicht. Das verwendete Konzept bewirkt zudem eine Verbesserung der Werte der Metrik 3.4.1 – Codeduplizierung – da alle wiederkehrenden Abläufe in eine Action ausgelagert wurden.

Neben den Verbesserungen, die anhand der Metriken festgestellt werden können, wurden weitere Zielvorgaben umgesetzt. Eine automatische Überprüfung dieser Regeln ist mit den vorliegenden Testwerkzeugen jedoch nur begrenzt möglich. Es handelt sich um:

- Das Löschen aller nicht genutzten TestSetups, Scenarios und Actions,
- Wiederholungen im TestDesign (z.B. durch technische Angaben),
- das Vorhandensein einer eigenen Menge an Scenarios für jedes TestSetup sowie
- das Erfüllen weiterer Namenskonventionen.

Die nicht genutzten Scenarios und Actions wurden im Rahmen der statischen Analyse ermittelt und gelöscht. Es gibt aber derzeit keine Möglichkeit, diese mit den vorhandenen Mitteln automatisch zu erkennen und zu löschen. Das Gleiche gilt für alle Wiederholungen im TestDesign (z.B. technische Angaben und Nutzerdaten). Diese wurden bei der statischen Analyse mittels Metrik 3.3.1 aufgedeckt und für den Test „Patient Option Template“ aus dem TestDesign entfernt.

Sowohl die dritte als auch die vierte Regel können zumindest teilweise automatisiert sichergestellt werden, wenn sich bei der Spezifikation der Testfälle im TestDesign an die vorgegeben Regeln gehalten wird. So muss bereits bei der Spezifikation der Name des TestSetups den Namen der TestSteps als Kürzel vorangestellt werden. Mit diesem Schritt ist die Regel, dass jedes TestSetup eine eigene Menge an Scenarios erhält, und die Namenskonvention, dass alle Scenarios den Kürzel des TestSetups vorangestellt bekommen, erfüllt. Die Überprüfung erfolgt durch die Ergebnisübertragung mit dem RepFram, welches für die TestSteps und Scenarios den gleichen Namen erwartet.

Die folgende Zielvorgabe kann derzeit nicht umgesetzt werden:

- Begrenzung der TestItems pro Scenario.

Eine automatisierte Prüfung dieser Zielvorgabe ist zwar technisch gesehen einfach umzusetzen. Eine Prüfung der Ergebnismengen im RepFram könnte die Anzahl der Ergebnisse pro Scenario gegen einen vorgegebenen Wert begrenzen. Diese Zielvorgabe konnte jedoch bisher nicht realisiert werden, da kein geeigneter Wert für eine Begrenzung zu ermitteln war. Dies liegt vor allem in den großen Unterschieden in der Komplexität der einzelnen Tests begründet, die insbesondere durch unterschiedlich umfangreiche Testziele verursacht sind.

6 Zusammenfassung

In dieser Arbeit wurde die Testqualität in einer Gruppe von automatisierten System-Tests untersucht. Durch eine Kombination aus statischer Analyse und der Goal Question Metric wurden in den Bereichen Testeffektivität und Wartbarkeit Defizite in der Testqualität aufgedeckt. Der Einsatz der Goal Question Metric ermöglichte dabei die gezielte Eingrenzung der zu messenden Eigenschaften. Durch die Anwendung quantitativer Metriken konnten leicht verständliche und vergleichbare Datenmengen erhoben werden. Nach Betrachtung und Wertung der Ergebnisse der statischen Analyse der TestDesigns und Testskripte konnten verschiedene Defizite in den Schlüsselbereichen aufgedeckt werden. Das Aufstellen konkreter Zielvorgaben ermöglichte ein Konzept zur Verbesserung und zum Erhalt der Testqualität. Auf der Grundlage der Zielvorgaben wurde die Entwicklung einer Musterstruktur für die TestDesigns und Testskripte sowie die Umsetzung des Reporting-Frameworks realisiert. Anhand der beispielhaften Implementierung wurde festgestellt, dass eine Verbesserung für die Qualitätsmerkmale Testeffektivität und Wartbarkeit in den betrachteten Metriken erreicht wurde. Zudem kann ein Teil der zuvor aufgestellten Regeln, Zielvorgaben und Metriken durch die realisierte Umsetzung zukünftig automatisiert sichergestellt werden. Dies ermöglicht neben dem Erhalt der Testqualität die Nachverfolgbarkeit der Anforderungen von der Spezifikation der logischen Testfälle, über die derzeit noch manuelle Erstellung konkreter Testfälle bis hin zur automatisierten Übertragung der Ergebnisse ins Test-Management-Tool.

6.1 Fazit

Der umgebaute Test ist im Rahmen der durch die Testwerkzeuge gegebenen Möglichkeiten verbessert worden. Auch wenn alle Werte der betrachteten Metriken optimiert wurden, gibt es weiterhin Verbesserungspotential. Die Testfälle sind nach dem Umbau des Tests teilweise noch zu konkret. Durch die Verwendung abstrakterer Testfälle könnte die Anzahl der TestSteps und Scenarios weiter gesenkt werden. Die Anzahl der Scenario-Aufrufe würde dementsprechend ansteigen. Durch eine bessere Parametrisierung in JavaTT wäre ein Modularisierungskonzept mit kleineren Modulen, wie es ursprünglich in Abschnitt 4.1.1 (Seite 37) vorgestellt wurde, umsetzbar. Zudem führt die momentane Parametrisierung zu Fällen, in denen ein Mehrfachaufruf von Scenarios notwendig ist, der durch einen Schleifenaufruf von Actions vermieden werden könnte.

Auch die bisher fehlende Begrenzung der TestItems ist kritisch zu sehen. Trotz der eingeführten Nutzung von Actions, kann dies weiterhin zu sehr komplexen Scenarios führen, welches die Auswertung eines Scenarios im Fehlerfall erschwert.

Für die erreichte Umsetzung ist positiv zu werten, dass durch die automatisierte Übertragung der Ergebnisse der manuelle Schritt der Ergebnisbewertung und Dokumentation entfällt. Die Erstellung der Testskripte bleibt hingegen vorerst manuell, ist aber durch die umgesetzte Zuordnung von TestSteps zu Scenarios nicht mehr losgelöst von der Erstellung der TestDesigns.

Über erreichte Verbesserungen im Kommentaranteil der Tests wurde keine Aussage getroffen. Einerseits ist die Berechnung des Kommentaranteils für XML-Skripte bisher in keiner anderen Arbeit vergleichbar durchgeführt worden, sodass Vergleichswerte für die Bewertung dieser Metrik fehlen. Andererseits wurden keine Angaben über den Sollwert des Kommentaranteils in wissenschaftlichen Arbeiten gefunden, sodass eine weitere Bewertung der Ergebnisse der Metrik nicht sinnvoll erscheint.

6.2 Ausblick

Die vorliegende Arbeit betrachtet die Testqualität lediglich für die Testgruppe der Regressionstests. Um eine Vergleichbarkeit mit weiteren Ergebnissen herzustellen, sollten die weiteren Testgruppen der Validierungsgruppe ebenfalls untersucht werden. Hierbei wäre auch der Vergleich mit anderen wissenschaftlichen Arbeiten, die sich gleichfalls mit der Betrachtung von XML-Skripten befassen, erforderlich.

Darüber hinaus wäre auch die Anwendung weiterer Metriken insbesondere in Bezug auf die Betrachtung der Komplexität der Testskripte und die Analysierbarkeit und Verständlichkeit der TestDesigns vorstellbar. So könnte die Anwendung der Halstead-Metriken eine Aussage über die Komplexität der Testskripte treffen. Dabei besteht die Schwierigkeit in der Bestimmung der Operatoren und Operanden in den vorliegenden TestSetups, Scenarios und Actions. Die Nutzung eines Lesbarkeitsindexes wie z.B. „Clarity-Index“ oder „Flesch Reading Ease“ ermöglicht eine Aussage zur Analysierbarkeit und Verständlichkeit der TestDesigns. Hierbei muss jedoch ein Ansatz gefunden werden, der die Anwendung des ermittelten Wertes auf einen technischen Text erlaubt. Dies beinhaltet die Berücksichtigung von häufigen Aufzählungen und Listen mit technischen Details in den TestDesigns.

Um den Erhalt der Testqualität auch zukünftig sicherzustellen, sollte die derzeitige manuelle Prüfung von Zielvorgaben und Regeln durch eine Erweiterung der Logik im RepFram ebenfalls automatisiert umgesetzt werden.

Ein weiterer interessanter Ansatz wäre die Erstellung von XML-Grundgerüsten aus dem TestDesign heraus. Damit könnte die manuelle Erstellung konkreter Testfälle zukünftig entfallen. Die Grundlage hierfür wurde bereits in dieser Arbeit mit der Zuordnung von TestSteps zu Scenarios gelegt. Dadurch könnte sich aus jedem TestStep das zugehörige Scenario erstellen lassen. So würde aus einem TestStep „Login“ ein äquivalentes XML-Gerüst des Scenarios „Login“ entstehen.

Tabellenverzeichnis

1 Ausschnitt aus der Ergebnisdatei des TestSetups „AckPost“ 11

2 Bewertung des Fehlerquotienten 19
3 Auswertung der Metriken für das Ziel 1 24
4 Auswertung der Metriken für das Ziel 2 27
5 Auswertung der Metriken für das Ziel 3 31
6 Auswertung der Metrik 3.1.1 für Ziel 3 32

7 Übersicht der ALM-Elemente 47

8 Bewertung der erreichten Verbesserungen 54
9 Bewertung der erreichten Verbesserungen der Metrik 3.1.1 für Ziel 3 . . . 56

A.1 Namen der TestSetups und TestDesigns XX

Abbildungsverzeichnis

1 Home Monitoring [2] 1

2 Übersicht der Objekte aus QC 12
3 Ausschnitt aus ALM 13
4 Gegenüberstellung der Begriffe in ISTQB, JavaTT und QC 14

5 Anzahl der Scenarios, Scenario-Aufrufe und TestSteps 26
6 Anzahl der TestCases pro Scenario 28
7 Anzahl der TestItems pro Scenario 29

8 Momentaner Testablauf der HMSC3 Val-Tests 40
9 Ablauf mit integriertem RepFram 41
10 ALM – Pfad zum TestSet und Name des TestDesigns hervorgehoben . . . 42
11 Systemübersicht 45
12 TestItem im Scenario liefert Warning – TestStep „Failed“ 49
13 Default TestSteps – Status „Passed“ 50
14 TestStep ohne zugehöriges Scenario – TestStep „Failed“ 50
15 Scenario ohne zugehörigen TestStep – neuer TestStep „Failed“ 51

16 Erfolgreicher Testdurchlauf mit Ergebnisübertragung 53
17 Anzahl der Scenarios, Scenario-Aufrufe und TestSteps – vorher/nachher . 55
18 Anzahl der TestCases in den Scenarios (nachher) 55
19 Anzahl der TestItems in den Scenarios (nachher) 56

B.1 Klassendiagramm XXII
B.2 Standardablauf im RepFram – Sequenzdiagramm XXIV

Listingsverzeichnis

1 Ausschnitt eines Testskripts 9
2 Ausschnitt eines Scenarios 10

3 TestSetup mit Änderungen 42
4 Ergebnisanzeige im Browser 48
5 GET-Anfrage in Java 48

A.1 Ausschnitt eines TestSetups XIX
A.2 Ausschnitt eines Scenarios XIX

Literaturverzeichnis

[1] Bildungsserver Rheinland-Pfalz. http://englisch.bildung-rp.de/sek2/abitur.html#c5956.

[2] BIOTRONIK. http://www.biotronik.de.

[3] Unterlagen der Vorlesung Softwaretechnik. http://www.inf.fu-berlin.de/inst/ag-se/teaching/V-SWT-2012/32_Modularisierung.pdf.

[4] *IEEE 610 IEEE Standard Computer Dictionary: Compilation of IEEE Standard Computer Glossaries.* Institute of Electrical and Electronics Engineers (IEEE), 1991.

[5] *IEEE 829-1998 - IEEE Standard for Software Test Documentation.* Institute of Electrical and Electronics Engineers (IEEE), 1998.

[6] JavaTT. http://javatt.sourceforge.net/, seit 2004.

[7] V.R. Basili and D.M. Weiss. A methodology for collecting valid software engineering data. *Software Engineering, IEEE Transactions on*, pages 728–738, Nov. 1984.

[8] Roy Fielding. *Architectural Styles and the Design of Network-based Software Architectures.* PhD thesis, University of California, Irvine, 2000.

[9] Dr. Matthias Hamburg and Dr. Uwe Hehn. ISTQB/GTB Standardglossar der Testbegriffe. http://www.software-tester.ch/PDF-Files/CT_Glossar_DE_EN_V21.pdf, 2010. German Testing Board e.V.

[10] John Kent M.Sc. An Entity Model of Software Testing: A Foundation for the Future. http://www.simplytesting.com/Downloads/An%20Entity%20Model%20For%20Software%20Testing%2005.pdf, 2008. EuroSTAR 2008, The Hague.

[11] I.S. Organization. *ISO/IEC 9126: Information Technology - Software Product Evaluation - Quality Characteristics and Guidelines for Their Use.* 1991.

[12] Andreas Spillner and Tilo Linz. *Basiswissen Softwaretest - Aus- und Weiterbildung zum Certified Tester, Foundation Level nach ISTQB-Standard (4. Aufl.).* dpunkt.verlag, 2010.

[13] Benjamin Immanuel Eberhardt Elmar Zeiß. *Quality Assurance of Test Specifications for Reactive Systems.* PhD thesis, Georg-August-Universität zu Göttingen, 2010.

[14] Benjamin Zeiss, Diana Vega, Ina Schieferdecker, Helmut Neukirchen, and Jens Grabowski. Applying the iso 9126 quality model to test specifications – exemplified for ttcn-3 test specifications. In *Software Engineering*, pages 231–242, 2007.

Anhang

A Erläuterung einzelner Kenngrößen zur Bestimmung der Metriken

A.1 Metriken 1.3.1 bis 1.3.3 – Anzahl Scenarios, Scenario-Aufrufe und TestSteps

Um die Anzahl der Scenarios, Scenario-Aufrufe und TestSteps darzustellen, wird als Diagramm ein Säulendiagramm verwendet. Die Erstellung des Säulendiagramms erfolgt mit R[10]. R ist sowohl eine Programmiersprache als auch eine Umgebung für statistische Berechnungen und Grafiken. Folgende Befehle wurden angewandt, um die Anzahl der Scenarios, Scenario-Aufrufe und TestSteps darzustellen. Nachdem die Datenmenge in einer CSV-Datei abgespeichert ist, kann man sie mit folgenden Befehl in R einlesen:

```
datSc <- read.csv("Pfad/bsp.csv", sep=";", header=T)
```

Die eingelesenen Datensätze entsprechen in R einer Liste, welche als Einträge Vektoren enthält. Jeder Vektor enthält die Anzahl der Scenarios, Scenario-Aufrufe und TestSteps des jeweiligen Tests. Die einzelnen Vektoren der eingelesenen Daten werden mit dem folgenden Befehl spaltenweise zusammengefasst.

```
dd <- cbind(c(datSc[[1]]),c(datSc[[2]]),c(datSc[[3]]), c(datSc[[4]]),
c(datSc[[5]]), c(datSc[[6]]), c(datSc[[7]]), c(datSc[[8]]),
c(datSc[[9]]), c(datSc[[10]]),c(datSc[[10]]))
```

Anschließend kann aus dieser Matrix ein Säulendiagramm erstellt werden, bei dem die Säulen pro Test gruppiert sind.

```
barplot(dd,beside = TRUE, names.arg=c("A","B", "C", "D","E",
"F","G","H","I","J", "K"),col=c('yellow','orange','red'), ylim=c(0,150))
```

A.2 Metriken 2.3.4–2.3.6 und 2.3.7–2.3.9 – Metriken über die Komplexität der Scenarios

Die Streuung der Datenmenge der TestCases und TestItems wird mit Hilfe einer Boxplot anschaulich dargestellt. Über die Boxplot wird zusätzlich eine Stripchart gelegt, die es

[10] The R Project for Statistical Computing: `http://www.r-project.org/`

ermöglicht, die einzelnen Datenpunkte der TestCases oder TestItems darzustellen. Die Erstellung der beiden Diagramme erfolgt ebenfalls mit R.

Folgende Befehle wurden angewandt, um die TestCases und TestItems grafisch darzustellen. Nachdem die Datenmengen für TestCases und TestItems jeweils in einer CSV-Datei abgespeichert sind, kann man sie wie zuvor in R einlesen (Befehl siehe A.1). Die eingelesenen Datensätze entsprechen in R einer Liste deren Einträge Vektoren sind. Jeder Vektor enthält die Anzahl der TestCases oder TestItems des jeweiligen Tests. Um das Fenster für die Grafik anzupassen und einige globale Einstellungen am Diagramm vorzunehmen, werden die beiden folgenden Befehle verwendet:

```
windows(width=10, height=5)
par(mar=c(5,16.5,2,2), las=1, lab=c(11,11,7))
```

Der erste Befehl bestimmt die Größe des Grafikfensters. Durch den zweiten Befehl werden der Abstand des Rahmens, der Stil der Achsenlabel auf horizontal und die Anzahl der Datenpunkte auf den Achsen festgelegt. Zunächst wird die Stripchart mit folgendem Befehl ausgegeben:

```
stripchart(datenTC, pch=1, ylim=c(0,11), cex.axis=0.8, method="jitter")
```

Durch den Parameter method="jitter" werden Datenpunkte, die den gleichen Wert haben, nicht übereinandergelegt, sondern leicht versetzt in das Diagramm eingezeichnet. Anschließend wird die Boxplot über die vorhandene Grafik gelegt:

```
boxplot(datenTC, horizontal= T, varwidth=T, ylim=c(0,11), cex.axis=0.8,
xlab="Anzahl der TestCases", add=T)
```

Durch den Parameter „add=T" werden beide Diagrammtypen in ein Fenster übereinander gezeichnet.

A.3 Metrik 2.4.1 – Kommentaranteil

Zur Berechnung des Kommentaranteils müssen die Anzahl der Kommentare und die Größe des TestSetups bestimmt werden. Auf die Bestimmung dieser einzelnen Größen wird detailliert eingegangen.

Metrik 2.4.1:

$$\textit{Kommentaranteil} := \frac{\textit{Anzahl Kommentare}}{\textit{Größe des TestSetups}} \tag{2.4.1}$$

A.3.1 Anzahl der Kommentare

Zur Berechnung der Anzahl der Kommentare werden alle Kommentare, die den Inhalt des Skriptes beschreiben, gezählt. Es werden alle Kommentare in den TestSetups, Scenarios und Actions gezählt. Es entfallen Kommentare, die lediglich Teile des Quellcodes auskommentieren und Kommentare, die Angaben zur letzten Änderung oder zum Autor enthalten.

Für Mehrfachaufrufe der Scenarios oder Actions innerhalb eines TestSetups oder eines Scenarios gelten Zusatzregeln. Dabei wird zwischen

- Mehrfachaufrufen aus jeweils anderem Kontext und
- Mehrfachaufrufen als Schleifenaufruf

unterschieden. Im ersten Fall wird das Scenario oder die Action mehrfach ausgewertet, da jeder Aufruf aus einem anderem Kontext – z.B. später im Skript – gegebenenfalls ein mehrfaches Lesen und Verstehen der Kommentare erfordert. Die Anzahl der Kommentare multipliziert sich dabei mit der Anzahl der Aufrufe aus einem anderen Kontext. Für den zweiten Fall muss das Scenario oder die Action nur einmal im jeweiligen Kontext der Schleife verstanden werden. Die Anzahl der Kommentare wird dabei ebenfalls nur einfach gezählt.

A.3.2 Größe des TestSetups

Die Größe des TestSetups setzt sich zusammen aus:

- der Anzahl der Aufrufe der Scenarios,
- der Anzahl der Aufrufe der Actions,
- der Anzahl der TestItems in den Scenarios und den Actions sowie
- der Anzahl der Kommentare.

Für Mehrfachaufrufe gilt wiederum die Regel, die bereits für die Anzahl der Kommentare angewandt wurde. Um in der Größe nicht nur die Anzahl der TestItems über das gesamte Skript zu erfassen, sondern auch die Größe des TestSetups, werden für die Bestimmung der Größen auch die Anzahl der Aufrufe berücksichtigt. Die Anzahl der Kommentare ist für die Größe des TestSetups ebenfalls zu beachten, um die Kommentaranzahl anteilig bestimmen zu können.

A.3.3 Beispiel zur Bestimmung

```
<TestSetup Name="AckPost">
  <SetupEntry>
    <!-- Login -->
    <Scenario Name="SCN_HMSC3_Platform/Login">
      <!-- Parameters which are necessary for the login -->
      <Parameter Name="UserGroup"     Value="System"/>
      <Parameter Name="UserName"      Value="PreviewCSS"/>
      <Parameter Name="Password"      Value="test12"/>
      <Parameter Name="url" Value="www.testsystem-homemonitoring.int"/>
    </Scenario>
    <!-- Deactivate Monitoring -->
    <Scenario Name="SCN_PatientAdministration/PA_DeactivateMonitoring">
      <Parameter Name="Patient" Value="Val-60400025" />
    </Scenario>
  </SetupEntry>
</TestSetup
```

Listing A.1: Ausschnitt eines TestSetups

Für den vorliegenden Ausschnitt aus dem TestSetup ergeben sich zwei Aufrufe und drei Kommentare.

```
<Scenario Name="PA_DeactivateMonitoring">
  <TestCase Name="Click Menu All Patient">
    <FollowLink Name="Click menu 'All patient'" LinkText="All patient"/>
    <!-- Check if we can see all patient -->
    <HTMLTagVerificationPoint Name="Check if step is executed correctly">
      <HTMLTag Name="input">
        <HTMLTagAttribute Name="value" Value="Extended view"/>
      </HTMLTag>
      <Verifier Name="Exists" Type="java.lang.Boolean"  Value="true"/>
    </HTMLTagVerificationPoint>
  </TestCase>
  <TestCase Name="Actions/Act_Select_Patient"/>
   <TestCase Name="Deactivate Home Monitoring">
    <FollowLink Name="Click 'Patient profile'" LinkText="Patient profile"/>
    ...
  </TestCase>
</Scenario>
```

Listing A.2: Ausschnitt eines Scenarios

Für den Ausschnitt aus dem Scenario sind drei TestItems, ein Kommentar und ein Aufruf der Action bei der Bestimmung der Größe des TestSetups zu berücksichtigen. Damit ergeben sich für beide Testausschnitte insgesamt zehn Größeneinheiten, davon vier Kommentare. Der Kommentaranteil beträgt somit 40% .

A.4 Metrik 3.2.1 – Bewertung der Namensgebung der TestDesigns

Um die Bewertung der Namensgebung der Metrik 3.2.1 (siehe Auswertung Tabelle 5) nachzuvollziehen, erfolgt eine Gegenüberstellung der Namen von TestDesigns und TestSetups.

Tabelle A.1: Namen der TestSetups und TestDesigns

genutzter (Kurz-)Name[11]	TestSetups	TestDesigns
Performance Batchprint (AllPatients)	TS_Perf_BatchPrinting.tsu	GUI Performance - Batch Printing
Performance Batchprint (PDF with many Episodes)	Perf_PrintPDFwithEpi.tsu	
Patient Administration	PatientAdministration.tsu	Patient Administration Scenario
Patient Option Template	PA_patientOptionTemplate.tsu	
User Administration	UserAdmin.tsu	User Administration Scenario
Acknowledge and Postpone	AckPost.tsu	Legacy test Lumax - Finding acknowledge and postpone Scenario
Multiple User Access	MultipleUsersAccess.tsu	Multiple User Access Scenario
Implant and Feature Administration	IFAdmin.tsu	Implant and Feature Administration scenario
Performance Transmitter Overview	Perf_TransmitterOverview.tsu	GUI Performance - transmitter overview
Performance Print PDF	Perf_PrintPDF.tsu	GUI Performance - download report (saveprint PDF)
Performance Download Monitoring Data	Perf_DownloadMonitoringData.tsu	GUI Performance Download Monitoring Data

Die Bewertungen sind für die ersten beiden Teile des Bewertungsmaßstabs (siehe Metrik 3.2.1) einfach nachzuvollziehen. Für den dritten Teil werden die Auffälligkeiten, die mit einem „-“ negativ vermerkt werden, nachfolgend erläutert.

Die ersten beiden Tests erhalten jeweils ein „-“, weil die Tests in einem gemeinsamen TestDesign spezifiziert sind. Auch die folgenden beiden Tests werden aus diesem Grund entsprechend bewertet. Zusätzlich erhält der Test „Patient Administration“ ein weiteres „-“ in Teil 3 des Bewertungsmaßstabs, weil das TestDesign das Wort „Scenario“ im Namen enthält. Aufgrund des Scenario-Elements in JavaTT kann dies zu Missverständnissen führen und ist negativ zu werten. Auch das TestDesign „User Administration Scenario“ erhält aus diesem Grund im Teil 3 des Bewertungsmaßstabs ein „-“. Das TestDesign „Legacy test Lumax - Finding acknowledge and postpone Scenario“ enthält in der Bezeichnung zwei missverständliche Bestandteile („Scenario“ und „Legacy test Lumax“), die jeweils negativ bewertet werden. Die letztgenannte Bezeichnung ist missverständlich, da dieser Test kein Test der Gruppe der Legacytests ist (Testgruppen siehe Abschnitt 2.2.2.1). Die nächsten beiden Tests – „Multiple User Access“ und „Implant and Feature Administration“ – erhalten in der Bewertung ihrer TestDesigns für den Bestandteil „Scenario“ wiederum ein „-“. Die letzten drei Tests erhalten im dritten Teil keine negativen Bewertungen für den Namen des TestDesigns, da es keine Auffälligkeiten in der Namensgebung gibt.

A.5 Metrik 3.4.1 – Codeduplizierung

Metrik 3.4.1:

$$\textit{Code-Duplizierung} := \frac{\textit{Anzahl duplizierte TestCases}}{\textit{Gesamtanzahl TestCases}} \tag{3.4.1}$$

Für die Anzahl der duplizierten TestCases wurden alle Abläufe mit mehr als drei Zeilen wiederkehrendem Code gezählt, die innerhalb des betrachteten Scenarios oder in einem anderen Scenario auftauchen.

Die Gesamtanzahl der TestCases setzt sich aus zwei Werten zusammen:

- der Anzahl der TestCases in den Scenarios eines TestSetups und
- der Anzahl der wiederholten Abläufe, die keine TestCases sind.

Der erste Wert ist die Anzahl aller vorhandenen TestCases. Diese TestCases werden summiert. Es werden keine TestCases mehrfach gezählt, wenn das Scenario mehrfach aufgerufen wird, da sich der wiederholte Aufruf nicht auf die Anzahl der benötigten Aktionen bei einer Änderung auswirkt.

Der zweite Wert – die Anzahl der wiederholten Abläufe, die keine TestCases sind – rechnet alle Abläufe hinzu, die als dupliziert erkannt wurden, aber derzeit keine TestCases sind. Weil jeder wiederkehrende Ablauf in der Anzahl duplizierter TestCases als TestCase gezählt wird, muss er bei der Gesamtanzahl der TestCases ebenso berücksichtigt werden. Dadurch wird eine Verfälschung des Anteils der duplizierten TestCases vermieden.

B Implementierungsdetails

B.1 Klassendiagramm des RepFram

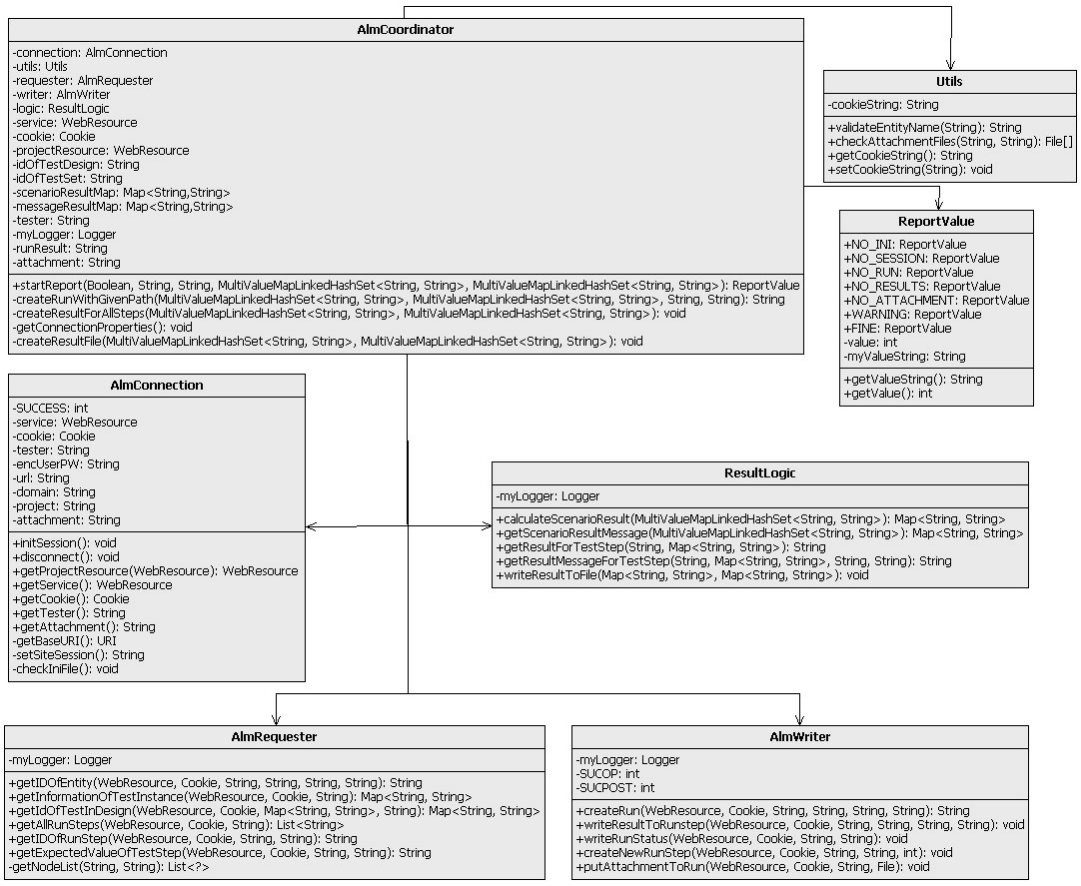

Abbildung B.1: Klassendiagramm

Die zentrale Klasse ist die `AlmCoordinator`-Klasse. Mit der enthaltenen Methode `startReport()` kann das Test-Tool den Übertragungsprozess des RepFram auslösen. Gleichfalls koordiniert diese Klasse alle Operationen innerhalb des Prozesses.

Die Klasse `Utils` enthält alle Hilfsmethoden und speichert den Wert für den session cookie. Diesen müssen die Klasse `AlmRequester` und `AlmWriter` vor jeder Anfrage abfragen und bei der Anfrage übergeben. Damit ist sichergestellt, dass alle Anfragen die selbe Session benutzen. Andernfalls würde ALM für jede Anfrage eine neue Session erstellen, so dass es eine große Menge an Leerlauf-Sitzungen nach Beendigung der Ergebnisübertragung nach ALM gäbe.

Die `AlmConnection`-Klasse erstellt die benötigte Client-Anwendung und initialisiert eine Session in ALM. In der Instanz dieser Klasse werden die Werte der INI-Datei ausgelesen und gespeichert. Dadurch ist es möglich, dass die Werte direkt für die Authentifizierung des Clients mit ALM genutzt werden. Nach der Anmeldung wird die Site-Session für die

weitere Übertragung gesetzt. Dabei wird mit einer POST-Operation eine Site-Session in ALM erstellt und der Client bekommt einen Cookie übergeben. Dieser Cookie wird in der Variable `cookieString` in der Klasse `Utils` abgespeichert. Der zweite vorhandene Cookie dient zur Identifizierung des Nutzers nach der Authentifizierung und muss zusätzlich bei jeder Anfrage an den bzw. Operation auf dem Server übergeben werden.

Die `AlmRequester`-Klasse ist verantwortlich für alle GET-Operationen über die REST-Schnittstelle von ALM.

Die Klasse `AlmWriter` führt alle POST- und PUT-Operationen über die REST-Schnittstelle von ALM aus.

Die Klasse `ResultLogic` enthält die Logik mit der die Ergebnisse von JavaTT ausgewertet werden. In der Instanz der Klasse werden die Ergebnismengen von JavaTT mit den TestSteps von ALM verglichen, um zu prüfen, ob eine Zuordnung zwischen TestStep und Scenario besteht. Des Weiteren werden Default-Steps und Requirement-Steps erkannt und gesondert behandelt. Um die Ausführungsreihenfolge der Scenarios beizubehalten, wird die Datenstruktur `MultiValueMapArrayList` benötigt. Diese Datenstruktur erlaubt zudem die Verwendung mehrerer Werte für jeden Schlüssel. So ist es möglich, dass bei Mehrfachausführung eines Scenarios, die Ergebnisse aller Ausführungsschritte innerhalb des Scenarios zusammengefasst werden.

Der Ergebniswert des RepFram ist eine Datenstruktur, die in der Klasse `ReportValue` definiert wird. Das RepFram kann damit den endgültigen Status des Übertragungsprozesses wiedergeben, wie z.B. „NO_SESSION, 0, ‘No session to ALM possible‘“.

B.2 Ablaufdiagramm/Sequenzdiagramm

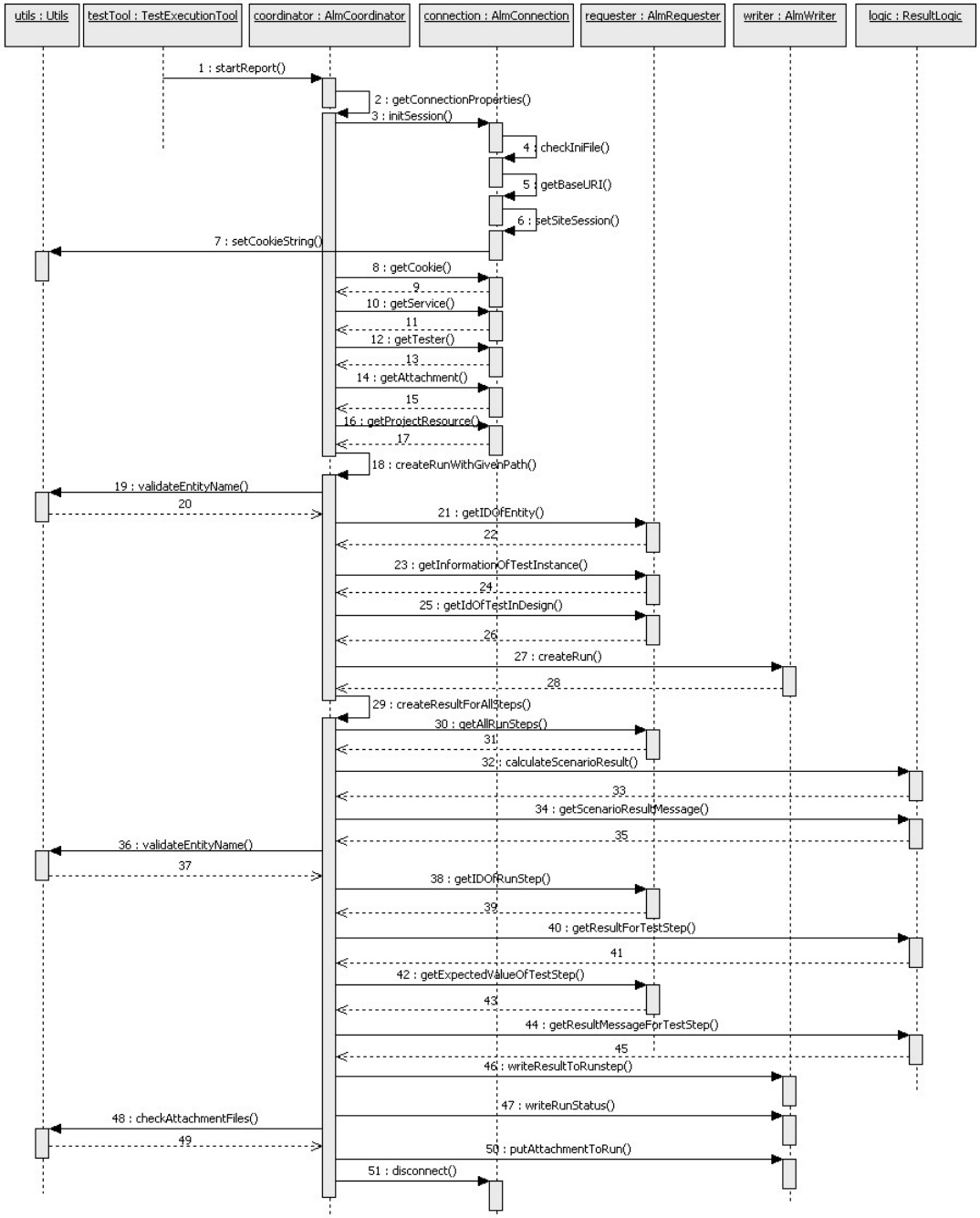

Abbildung B.2: Standardablauf im RepFram – Sequenzdiagramm