Institut für Softwaretechnologie
Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Bachelorarbeit Nr. 103

Testüberdeckung als Maß in Quamoco

Stefanie Dressel

Studiengang:	Softwaretechnik
Prüfer:	Prof. Dr. rer. nat. Stefan Wagner
Betreuer:	Dipl.-Ing. Jan-Peter Ostberg
begonnen am:	01. Dezember 2013
beendet am:	28. Mai 2014
CR-Klassifikation:	D.2.5, D.2.8, D.2.9

Kurzfassung

Um wettbewerbsfähige Softwareprodukte entwickeln und sich von anderen Unternehmen abgrenzen zu können, werden die Anforderungen an die Softwarequalität immer höher. Eine Möglichkeit diese greifbar und bewertbar zu machen, bieten die Qualitätsmodelle und die zur Verfügung gestellten Werkzeuge von Quamoco. Ein Qualitätsmaß von Software ist deren Testüberdeckung, welche bis zum Zeitpunkt der Erstellung dieser Arbeit noch nicht von Quamoco berücksichtigt wird. Ziel dieser Arbeit ist es daher, die Testüberdeckung als Maß in Quamoco zu integrieren.

Abstract

In order to develop competitive software products and to make yourself unique amongst other companies, the need for high quality software rises. One approach to make the quality of software more tangible and assessable is the usage of the Quamoco quality model and its tool chain. One metric of software quality is test coverage. This is not yet covered by Quamoco. Hence, this work aims to integrate test coverage into Quamoco.

Inhaltsverzeichnis

Abbildungsverzeichnis

Kapitel 1

Einleitung

1.1 Motivation

In den letzten Jahrzehnten hat sich Software zu einem *zentralen Werkstoff des Informationszeitalters* entwickelt [10]. Ein Leben ohne ist heutzutage nicht mehr denkbar, denn Software ist ein fester Bestandteil des modernen Alltags geworden. Es gibt so gut wie keinen Bereich, in dem sie nicht Einzug gehalten hat. Selbst einfache Dinge wie das Kaffeekochen werden mittlerweile per Software gesteuert, ganz zu schweigen von einem komplexen Objekt wie dem Automobil.
Dabei verlassen wir uns darauf, dass die Produkte fehlerfrei funktionieren. Dass dies nicht immer der Fall ist und fehlerhafte Software in der Vergangenheit schwere Unfälle verursacht hat, bezeugen nicht nur Zeitungsberichte. Ein bekanntes Beispiel ist der Raketenabsturz der Ariane 5 am 4. Juni 1996 bei ihrem Jungfernflug [17]. Ein Programmierfehler im Lenksystem führte schon wenige Sekunden nach dem Start zur Selbstzerstörung der Rakete.
Daher ist es notwendig eine gute Softwarequalität anzustreben, um wettbewerbsfähige Softwareprodukte entwickeln zu können. Hierfür wurden Software-Qualitätsmodelle erstellt, welche genau definieren was unter guter Softwarequalität verstanden wird. Quamoco (Kapitel 3) stellt ein solches Qualitätsmodell, sowie eine Werkzeugunterstützung zur Verfügung. Diese ermöglicht es, selbst Modelle erstellen und bearbeiten, sowie eine automatische Qualitätserhebung durchführen zu können.

1.2 Ziel

Ein Maß für die Qualität von Software ist die Testüberdeckung, welche zum Zeitpunkt der Erstellung dieser Arbeit noch nicht in Quamoco integriert ist. Um das Modell und die Werkzeugunterstützung von Quamoco jedoch weiter zu optimieren, stellt sich nun die Frage, wie Quamoco um das Maß der Testüberdeckung erweitert werden kann? Hierzu gehören eine detaillierte Qualitätsmodellierung und die Integration der Testüberdeckung in die Werkzeugkette. Ziel dieser Bachelorarbeit ist es daher, dieser Frage nachzugehen, wobei der Schwerpunkt auf der Integration in die Werkzeugkette liegt.

1.3 Aufbau der Arbeit

Diese Arbeit gliedert sich in folgende Teilbereiche:

Kapitel 2 – Grundlagen stellt den Begriff *Softwarequalität* und das Konzept *Qualitätsmodell* vor, und gibt einen Überblick über *Softwaremetriken*.

Kapitel 3 – Quamoco erklärt das allgemeine Konzept von Quamoco und beschreibt das Basismodell und die Werkzeugkette.

Kapitel 4 – Testüberdeckung benennt gebräuchliche Maße der Testüberdeckung und befasst sich mit den Werkzeugen, um diese zu messen.

Kapitel 5 – Integration stellt die durchgeführten Anpassungen am Basismodell und an der Werkzeugkette vor.

Kapitel 6 – Evaluation geht auf die Beurteilung des erweiterten Basismodells und der Werkzeugkette bei der Erprobung an Beispielsystemen eines Unternehmens ein.

Kapitel 7 – Fazit fasst die Ergebnisse der Arbeit zusammen und stellt mögliche Anknüpfungspunkte vor.

Kapitel 2

Grundlagen

Das folgende Kapitel beschreibt die nötigen Grundlagen zum Verständnis dieser Arbeit. Zu Beginn werden die Begriffe Softwarequalität und Qualitätsmodell erklärt und ein Modell für die Produktqualität vorgestellt. Anschließend wird ein Überblick über Softwaremetriken gegeben.

2.1 Softwarequalität

Unter dem abstrakten Begriff Qualität wird laut Duden [9] die "Gesamtheit der charakteristischen Eigenschaften einer Sache" verstanden. Übertragen gesagt, ist die Softwarequalität die Gesamtheit aller spezifischen Merkmale, die ein Softwareprodukt ausmachen. So definiert die Norm ISO/IEC 25000 Softwarequalität als "die Gesamtheit von Funktionen und Merkmalen eines Softwareprodukts, das die Fähigkeit besitzt, angegebene oder implizierte Bedürfnisse zu befriedigen". Je mehr die Anforderungen erfüllt werden, desto höher ist die Qualität der Software. Daher wird in der ISO 9000:2008 zitiert nach [15], die Softwarequalität auch als "Grad, in dem ein Satz von inhärenter Merkmalen Anforderungen erfüllt" definiert. Um welche Eigenschaften es sich dabei handelt wird in Qualitätsmodellen (Kapitel 2.2) festgelegt. Zur Bewertung der Qualität ist es daher erforderlich, dass die geforderten Eigenschaften messbar sind.

Die Softwarequalität lässt sich in zwei Bereiche unterteilen [16]: Zum einen in die **Prozessqualität**, unter der die Qualität des Projekts, in dem das Softwareprodukt hergestellt wird verstanden wird. Zum anderen in die **Produktqualität** bei der nur die Qualität des Produkts selbst betrachtet wird. Mit Hilfe einer hohen Prozessqualität soll auch die Produktqualität positiv beeinflusst werden, sie ist aber weder eine Garantie noch eine Voraussetzung für eine gute Produktqualität. Im weiteren Verlauf dieser Arbeit wird der Begriff Softwarequalität als Synonym für die Produktqualität verwendet.

2.2 Qualitätsmodelle

Mit Qualitätsmodellen wird versucht, die abstrakte Bezeichnung Softwarequalität verständlich und greifbar zu machen. Hierfür wird der Begriff konkretisiert und durch eine weitere Detaillierung operationalisiert [14].

2.2.1 Konzept

Die Hauptelemente von Qualitätsmodellen sind die Qualitätsmerkmale (Faktoren), welche hierarchisch angeordnet sind. Qualitätsmerkmale sind oft noch unpräzise definiert und nicht messbar [14]. Weitere Elemente von Qualitätsmodellen sind die Qualitätsindikatoren (Maße). Im Gegensatz zu Merkmalen sind sie manuell oder automatisch messbare und voneinander unabhängige Kriterien und bilden daher die unterste Ebene der Hierarchie. Ein Indikator ist oft für mehrere Qualitätsmerkmale relevant, ebenso können für ein Merkmal verschiedene Indikatoren wichtig sein. Daher ist die Hierarchie nicht überschneidungsfrei und bildet somit keinen Baum, sondern eine Halbordnung (siehe Abb. 2.1), wobei die Indikatoren die Blätter sind.

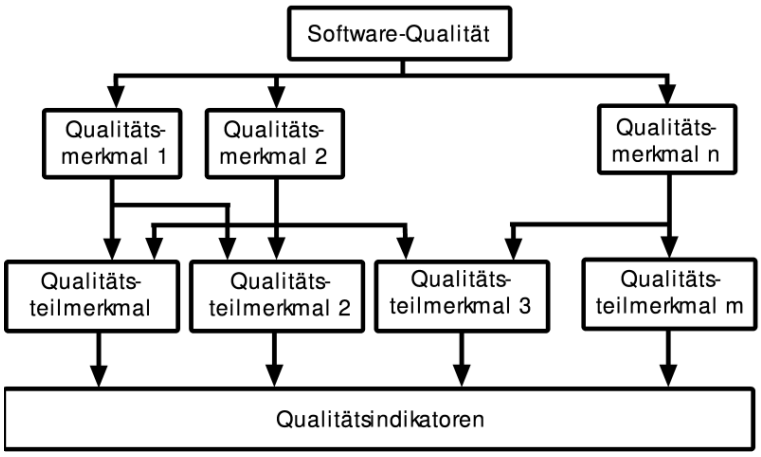

Abbildung 2.1: Schematischer Aufbau eines hierarchischen Qualitätsmodells [24]

2.2.2 Produktqualitätsmodell der ISO/IEC 25010

Die Abbildung 2.2 zeigt das Produktqualitätsmodell der ISO/IEC 25010 [6]. Die Produktqualität wird in diesem Modell in acht Merkmale unterteilt. Jedes dieser Merkmale stellt wiederum eine Gruppierung aus zusammengehörenden Untermerkmalen dar.

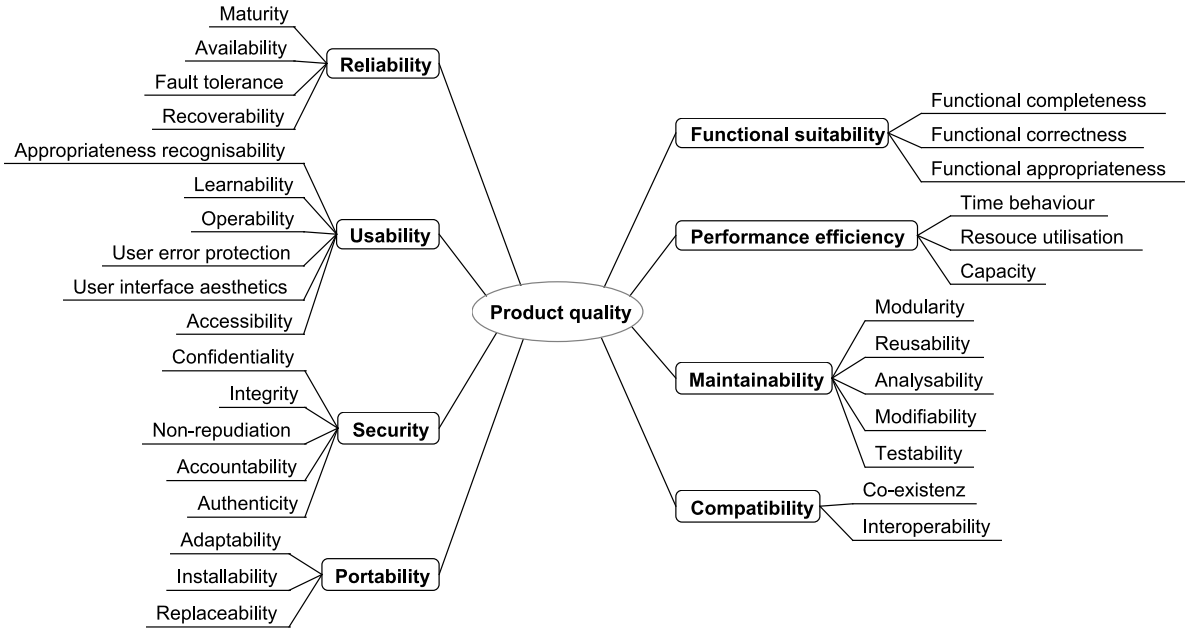

Abbildung 2.2: Produktqualitätsmodell der ISO/IEC 25010 [6]

2.3 Softwaremetriken

Im Software Engineering wird das Wort Metrik nicht als Distanzfunktion wie in der Mathematik, sondern in der umgangssprachlichen Bedeutung von Maß verwendet [16]. Es handelt sich hierbei um eine Abbildung einer Eigenschaft auf eine skalare oder vektorielle Größe. Zur besseren Übersicht werden Softwaremetriken nach ihrem Messobjekt in die drei Klassen Produkt-, Projekt- und Prozessmetriken unterteilt [19].

Produktmetriken

Diese Metriken befassen sich mit den Eigenschaften der Software selbst. Sie lassen sich wiederum in dynamische und statische Metriken einteilen. Dynamische Metriken sind Maße, die bei der Ausführung eines Programms gemessen werden, wohingegen statische Metriken durch Messung aus der Systemdarstellung ermittelt werden. Für die Beurteilung von Effizienz und Zuverlässigkeit sind dynamische Metriken hilfreich, statische Metriken eignen sich eher für die Beurteilung von Komplexität, Verständlichkeit und Wartbarkeit [20]. Bekannte Metriken sind zum Beispiel die Anzahl der Codezeilen (LoC - Lines of Code) oder die zyklomatische Komplexität nach McCabe [8]. Für viele Produktmetriken gibt es Analysewerkzeuge wie z.B. FindBugs [4] und Gendarme [5]. Sie dienen der effizienten und automatisierten Ermittlung der Produktmaße. Einige statische Analysen wie z.B. die Prüfung auf uninitialisierte Variablen, sind häufig schon in Compilern integriert.

Kapitel 3

Quamoco

Das Forschungsprojekt *Software-**Qu**alität: Flexible **Mo**dellierung und Integriertes **Co**ntrolling* (Quamoco) [12] befasste sich mit der Frage, wie die Leistungsfähigkeit und Wirtschaftlichkeit von Softwareprodukten bewertbar und nachweisbar gemacht werden kann. Projektteilnehmer waren die Unternehmen Capgemini, itestra, SAP und Siemens, sowie die Forschungseinrichtungen Fraunhofer IESE und Technische Universität München. Sie wurden dabei vom Bundesministerium für Bildung und Forschung, von 2009 bis 2012, im Rahmen des Förderprogrammes ITK 2020 mit 3,7 Millionen Euro unterstützt. Weitere rund 2,2 Millionen Euro brachten die Industriepartner selbst in das Forschungsvorhaben mit ein [21].

Wie bereits in Kapitel 2.2 beschrieben wird mit Qualitätsmodellen versucht, den abstrakten Begriff *Softwarequalität* durch eine hierarchische Verfeinerung zu konkretisieren. Allerdings sind bereits bestehende Qualitätsmodelle, wie z.B. das Produktqualitätsmodell (Kapitel 2.2.2) noch sehr abstrakt gehalten. Es werden dort zwar Qualitätsmerkmale, wie z.B. die „Zuverlässigkeit“ definiert, jedoch wird nicht angegeben wie sich diese messen lassen. Demgegenüber gibt es eine Reihe von Analysewerkzeugen, wie z.B. FindBugs [4] oder Gendarme [5], mit denen Softwarequalitätsmaße erhoben werden können. Diesen fehlt allerdings der Bezug zu den Qualitätszielen [21]. Einige Unternehmen definieren daher ihre eigenen Qualitätsrichtlinien, wobei diese meist nur für spezielle Bereiche der Softwarequalität nutzbar sind. Es ist daher wünschenswert einen überprüfbaren Qualitätsstandard zu haben.

Das Forschungsprojekt Quamoco hatte sich daher zum Ziel gesetzt, einen solchen Standard zu erarbeiten. Dieser Standard sollte ebenso die Vielfalt unterschiedlicher Softwareprodukte berücksichtigen [12]. Als Resultat des Projekts entstand eine Werkzeugkette zur Erstellung und Anwendung von Qualitätsmodellen, sowie ein Basis-Qualitätsmodell. Beides steht unter www.quamoco.de zur freien Verfügung [21].

3.1 Meta-Modell

Um eine konsistente Struktur von Qualitätsmodellen zu gewährleisten, hat das Forschungsprojekt ein Meta-Modell entwickelt [22]. Dieses definiert wie ein Quamoco-Qualitätsmodell

aufgebaut sein muss. Zudem wurde anhand des Meta-Modells ein Editor (Kapitel 3.3) implementiert, um neue Qualitätsmodelle erstellen und bestehende anpassen und erweitern zu können.
Quamoco-Qualitätsmodelle sind hierarchisch aufgebaut. Ebenso wie im Produktqualitätsmodell wird der abstrakte Begriff *Softwarequalität* durch Aufsplittung in Qualitätsmerkmale weiter verfeinert. Wie in Abbildung 3.1 zu sehen ist, beinhalten diese Modelle zwei unterschiedliche Hierarchien, mit Faktoren als Elemente. Diese sind entweder *Qualitätsaspekte* oder *Produktfaktoren*, welche beide jeweils weiter in Subaspekte und Subfaktoren verfeinert werden können. Alle Faktoren beschreiben die Eigenschaft einer *Entität*. Unter einer Entität wird ein eindeutig bestimmbares Objekt eines Softwareprodukts verstanden. Entitäten sind z.B. Klassen und Methoden, aber auch das gesamte Produkt wird als Entität bezeichnet. Der Unterschied zwischen den beiden Faktorarten ist, dass es sich bei einem Qualitätsaspekt um ein abstraktes Qualitätsziel handelt, welches immer das ganze Produkt als Entität besitzt. Demgegenüber ist ein Produktfaktor eine messbare Eigenschaft eines Produktteils. Sowohl die Qualitätsaspekte als auch die Produktfaktoren bilden eine eigene Hierarchie. Um die Lücke zwischen den beiden Hierarchieebenen zu schließen und damit eine Verbindung von den abstrakten Qualitätsaspekten zu den konkreten Produktfaktoren herzustellen, müssen diese beiden Faktorarten in Beziehung zueinander gesetzt werden. Hierbei können allerdings nur die Produktfaktoren die Qualitätsaspekte, entweder positiv oder negativ, beeinflussen. Um ausdrücken zu können, welcher Produktfaktor, welchen Qualitätsaspekt wie beeinflusst, werden im Modell *Aggregationen* definiert.

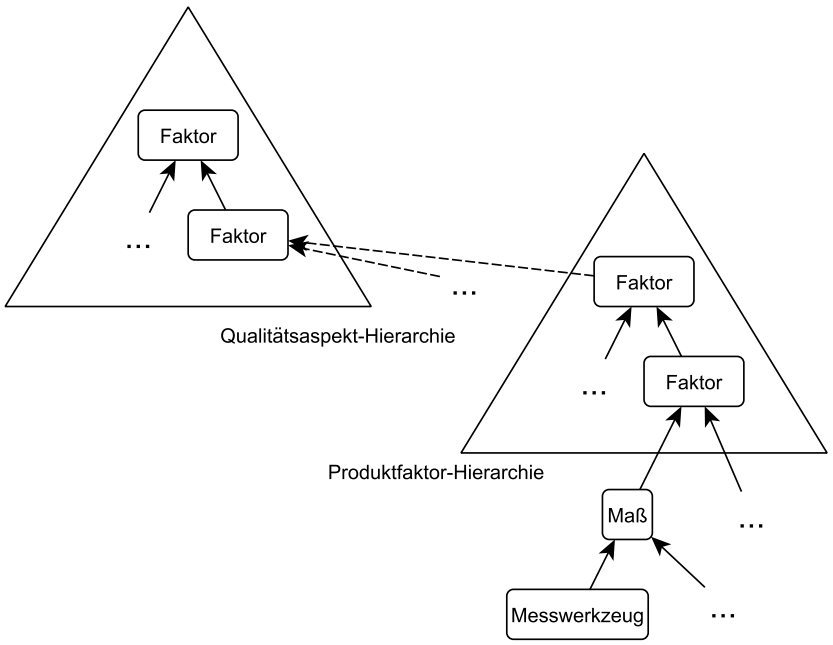

Abbildung 3.1: Schematische Darstellung
des Qualitätsmodellkonzept von Quamoco nach [22]

Weitere Elemente des Qualitätsmodells sind die *Maße*. Ein Maß ist eine konkrete Beschreibung, die für einen speziellen Kontext angibt, wie Produktfaktoren gemessen werden. Die Trennung von Produktfaktoren und Maßen ermöglicht es, die Faktoren in unterschiedlichen Kontexten zu messen. Um die beiden Elemente miteinander in Beziehung zu bringen, wird im Modell eine *Evaluation* definiert. Diese ermöglicht auch eine Zuordnung mehrerer Maße zu einem Produktfaktor. Des Weiteren besteht eine Trennung zwischen den Maßen und den Analysewerkzeugen. Hierdurch entsteht eine Flexibilität, um z.B. Daten manuell erheben oder unterschiedliche Werkzeuge in verschiedenen Kontexten verwenden zu können. Eine Verbindung zwischen diesen Elementen wird durch die Definition von *MeasurementMethods* (Messmethoden) im Modell hergestellt. Mit diesem hier vorgestellten Gesamtkonzept wird die Lücke zwischen dem abstrakten Begriff Softwarequalität und den konkreten Softwaremaßen geschlossen.

Modularisierung

Ein wichtiges Konzept von Quamoco ist die Modularisierung, welche die Unterteilung eines Modells in einzelne Module erlaubt [22].
Ein allumfassendes Qualitätsmodell, welches den Ansprüchen unterschiedlichster Technologiebereiche genügt, wäre sehr unpraktisch und unübersichtlich. Daher entwickelte das Forschungsteam ein Basismodell, welches in seinem Hauptmodul *Root* Qualitätsaspekte, Produktfaktoren und Maße, definiert, die weder von der Programmiersprache noch von der Produktsparte abhängen. Dieses Basismodell lässt sich durch neue Module erweitern. Hierdurch können Qualitätsmodelle erstellt werden, die auf der einen Seite auf spezielle Eigenschaften unterschiedlicher Technologien eingehen. Auf der anderen Seite die selben, allgemeinen Eigenschaften und Entitäten als Grundlage besitzen. Somit ist ein flexibler Umgang mit großen Modellen möglich. Im Folgenden wird das Basismodell von Quamoco genauer beschrieben.

3.2 Basismodell

Wie bereits in Kapitel 3.1 beschrieben, ist das Basismodell ein Qualitätsmodell, welches die allgemeingültigen Qualitätsaspekte und Produktfaktoren definiert, die nicht von einem bestimmten Technologiebereich oder einer Programmiersprache abhängen. Diese allgemeinen Eigenschaften sind in dem Modul *Root* definiert. Zum Zeitpunkt der Erstellung dieser Arbeit existieren zusätzlich noch drei weitere Module. Hierbei handelt es sich zum einen um ein Modul, welches spezifische Eigenschaften von objektorientierten Sprachen beinhaltet. Zum anderen gibt es zwei Module, welche die Eigenheiten der Sprachen Java und C# abbilden. Diese definieren zudem, wie die konkrete Messung von Eigenschaften übergeordneter Module stattfindet. In Abbildung 3.2 ist die Abhängigkeit der Module untereinander zu sehen. Das oberste Modul ist das Modul *Root*, welches für alle weiteren Module die Basis bildet. Darunter steht das Modul für die objektorientierten Spracheigenschaften und auf dritter

Ebene befinden sich die Module für die objektorientierten Sprachen. Die, mit gestrichelter Linie umrandeten Module stellen mögliche Erweiterungen des Basismodells dar.

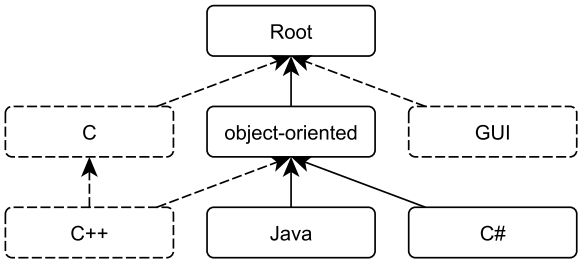

Abbildung 3.2: Modulhierarchie des Basismodells nach [23]

Das Modell besteht aus einer Auswahl von grundlegenden Qualitätsmerkmalen, d.h. von Faktoren und Maßen, die für die Qualitätsbeurteilung wichtig sind. Bei der Definition der Qualitätsaspekte wurden Produktqualitätsmerkmale des Produktqualitätsmodells ISO/IEC 25010 (Kapitel 2.2.2) verwendet.
Insgesamt umfasst das Modell 92 Entitäten und 284 Faktoren [23]. Einige dieser Faktoren dienen nicht direkt zur Qualitätsbeurteilung, sondern nur der Strukturierung. Und auch nicht jeder Faktor beeinflusst andere Faktoren. Insgesamt hat das Modell 490 definierte Einflüsse. Die 194 messbaren Faktoren sind im Vergleich zu den 526 Maßen, welche das Modell beinhaltet, sehr wenig. Die vergleichbar große Anzahl an Maßen lässt sich auf die Operationalisierung von verschiedenen Programmiersprachen zurückführen. Zur Messung der Maße werden acht manuelle Messmethoden und zwölf Werkzeuge zur Verfügung gestellt. Darunter FindBugs [4], welches für Java 361 Regeln modelliert und Gendarm [5] mit 146 Regeln für C#.

3.3 Werkzeuge

Das Forschungsteam hat neben den entstandenen Modellen auch Werkzeuge entwickelt, um Qualitätsmodelle erstellen und bearbeiten, sowie eine automatische Qualitätsbewertung durchführen zu können.

Qualitätsmodell-Editor

Der Editor basiert auf dem Eclipse Modeling Framework. Er ermöglicht die Erstellung und Bearbeitung von Qualitätsmodellen auf Grundlage des Meta-Modells.
Abbildung 3.3 zeigt den Qualitätsmodell-Editor. Auf der linken Seite in der Projektübersicht, sind das Basismodell und seine Module zu sehen. Jedes Modul wird in einer separaten Datei gespeichert, hierdurch wird das Konzept der Modularisierung unterstützt. Der Inhalt des Modells lässt sich über verschiedene Baumstrukturen einsehen. Im mittleren Bereich ist die

Faktorhierarchie dargestellt und auf der rechten Seite befindet sich eine Eingabemaske zur Definition von Eigenschaften zu einem ausgewählten Faktor.

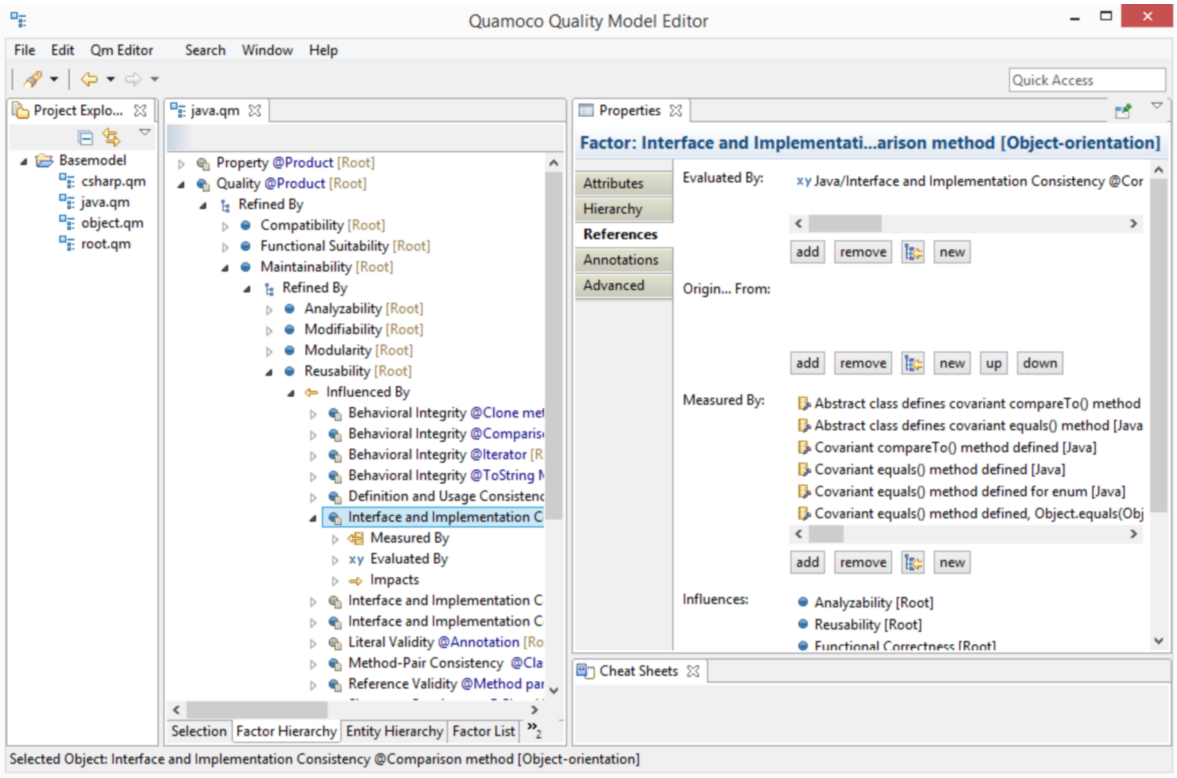

Abbildung 3.3: Qualitätsmodell-Editor

Anbindung an das Analysewerkzeug ConQAT

Das *Continuous Quality Assessment Toolkit* (ConQAT) [2] ist ein konfigurierbares Werkzeug zur Softwareanalyse. Mit einer graphischen Benutzeroberfläche lassen sich eigene Analysekonfigurationen erstellen. Hierfür werden sogenannte Prozessoren miteinander verknüpft und in Blöcken verwaltet. ConQAT selbst stellt viele Prozessoren zur Verfügung, z.B. zur HTML-Ausgabe von Daten.

Von dem Qualitätsmodell-Editor aus kann die Analysekonfiguration von Quamoco gestartet werden. Damit ConQAT eine Evaluation durchführen kann, wurde das Werkzeug durch das Forschungsteam um neue Prozessoren und Blöcke erweitert. Diese sind speziell auf die im Basismodell definierten Messwerkzeuge abgestimmt und können deren Messergebnisse verarbeiten. Zudem gibt es Prozessoren, die das Modell selbst einlesen und mit den gemessenen Maßen in Beziehung setzen. Die Ergebnisse werden als HTML-Ausgabe gespeichert und an den Editor gesendet, der diese visuell darstellt.

Kapitel 4

Testüberdeckung

Dieses Kapitel beschäftigt sich mit der Definition der Testüberdeckung und deren Eingliederung in die Klassifikation von Softwaretests. Es werden die Anweisungs- und die Zweigüberdeckung beschrieben. Im Anschluss erfolgt eine Vorstellung von Werkzeugen, zur Ermittlung der Testüberdeckung.

4.1 Definition

Testfälle können noch so gut sein, dennoch ist es mit ihnen nicht möglich nachzuweisen, dass keine Fehler mehr im Programmcode vorhanden sind [11]. Daher wird versucht die Testfälle so abgeschlossen wie möglich zu erstellen [18]. Dies bedeutet, mit den vorhandenen Testfällen, den gesamten Programmcode mindestens einmal auszuführen, denn Fehler können von UnitTests nur in einem ausgeführten Programmteil gefunden werden [19]. Indem nun Codestellen identifiziert werden, die noch nicht oder unzureichend von Testfällen abgedeckt sind, steigt die Wahrscheinlichkeit Fehler zu entdecken. Der Grad der Abdeckung von Code durch Testfälle wird auch als **Testüberdeckung** bezeichnet und in Prozent angegeben. Mit Hilfe der gemessenen Überdeckung können die vorhandenen Testfälle bewertet und verbessert werden, um so eine größtmögliche Testüberdeckung zu erzielen. Wichtigster Anwendungsbereich zur Messung der Überdeckung sind Modultests. Bei Systemtests hingegen werden sie nicht eingesetzt [15].

4.2 Klassifikation von Softwaretests

Es gibt viele unterschiedliche Arten von Softwaretests, welche sich nach verschiedenen Kriterien klassifizieren [16] lassen:

- Grundlage des Tests
- Aufwand für Vorbereitung und Archivierung

- Komplexität des Prüflings
- getestete Eigenschaft
- beteiligte Rollen

Bei der Aufteilung der Softwaretests nach dem Kriterium *Grundlagen des Tests*, entstehen zwei Klassen von Tests:

Funktionstest (*Black-Box-Test*) bezeichnet einen Test, dessen Testfälle sich auf die in der Spezifikation geforderten Eigenschaften bezieht.

Strukturtest (*Glass-Box-Test, White-Box-Test, Clear-Box-Test*) wird ein Test genannt, bei dem als Grundlage für die Testfälle, die innere Struktur des zu testenden Programms und die Aufzeichnungen früherer Programmabläufe herangezogen werden.

Die Ermittlung der Testüberdeckung gehört zu den kontrollflussorientierten Strukturtests und ist ein dynamisches Testverfahren. Im Gegensatz zu den statischen, wird bei dynamischen Verfahren die Software beim Testen ausgeführt. Die Testvollständigkeit wird anhand der Kontrollflussüberdeckung bewertet [15].

4.3 Kontrollflussorientierte Strukturtests

Zu den kontrollflussorientierten Strukturtests gehören nach [15]:

- Anweisungsüberdeckungstest
- Zweigüberdeckungstest
- Bedingungsüberdeckungstest
- Strukturierter Pfadtest und boundary interior-Pfadtest
- Linear Code Sequence and Jump - barsierter Test
- Pfadüberdeckungstest

4.3.1 Anweisungsüberdeckungstest

Mit einem Anweisungsüberdeckungstest wird ermittelt, wie viele Anweisungen des Programmcodes beim Testen ausgeführt werden. Um eine vollständige Überdeckung zu erzielen, muss jede Anweisung mindestens einmal ausgeführt werden. Das Testmaß ist der erreichte Anweisungsüberdeckungsgrad:

$$\text{Grad der Anweisungsüberdeckung} = \frac{\text{Anzahl der ausgeführten Anweisungen}}{\text{Gesamtanzahl der Anweisungen}}$$

Mit diesem Überdeckungstest wird sichergestellt, dass alle Anweisungen ausgeführt werden. Die Ausführung einer Anweisung ist ein notwendiges Kriterium, um in dieser ein Fehler lokalisieren zu können. Da das Auftreten eines Fehlers jedoch an die Ausführung mit bestimmten Testdaten gekoppelt sein kann, ist die Anweisungsüberdeckung kein hinreichendes Kriterium. Sie kann zur Quantifizierung der Testüberdeckung herangezogen werden, ist jedoch ein zu schwaches Kriterium für eine sinnvolle Testdurchführung und von der alleinigen Verwendung wird abgeraten [15].

Die Anweisungsüberdeckung ist das am einfachsten zu messende Überdeckungsmaß. Sofern der Programmcode keinen „toten Code“ beinhaltet, kann theoretisch eine hundertprozentige Anweisungsüberdeckung realisiert werden [15]. In der Praxis lässt sich jedoch nur sehr schwer eine Überdeckung von mehr als achtzig Prozent erzielen [16]. Es existiert oft defensiver Code, der nur bei Fehlern in anderen Programmteilen ausgeführt wird. Zur Erkennung von Wartungsfehlern ist solcher Code jedoch sinnvoll. Auch Testfälle, die abhängig von Uhrzeit und Datum sind, sind oft schwer zu realisieren.

4.3.2 Zweigüberdeckungstest

Bei einem Zweigüberdeckungstest müssen mindestens einmal alle Zweige des zu testenden Programms ausgeführt werden, um eine vollständige Überdeckung zu erzielen. Wird eine hundertprozentige Zweigüberdeckung erreicht, so liegt auch eine vollständige Anweisungsüberdeckung vor. Testmaß ist der erreichte Zweigüberdeckungsgrad:

$$\text{Grad der Zweigüberdeckung} = \frac{\text{Anzahl ausgeführter Zweige}}{\text{Gesamtanzahl der Zweige}}$$

Die Zweigüberdeckung gilt als Minimalkriterium von kontrollflussorientierten Tests [15]. Sie ist eine notwendige Testtechnik und wird von den meisten anderen Testmethoden subsumiert.
Alle Programmzweige mit Testfällen abzudecken, erweist sich in manchen Fällen als schwierig, da manchmal zwar der Zweig ausführbar wäre, jedoch die Erzeugung des Testfalls sehr kompliziert ist. Auf der anderen Seite kann es auch Zweige geben, die nie ausführbar sind. Solche überflüssigen Zweige entstehen oft durch Entwurfsfehler. Aber auch eine hundertprozentige Zweigüberdeckung ist keine Garantie für ein fehlerfreies Programm.

4.4 Werkzeuge zur Messung der Testüberdeckung

Heutzutage gibt es eine ganze Reihe von Werkzeugen zur Ermittlung der Testüberdeckung für verschiedenste Programmiersprachen. Allein für Java existieren mehr als zehn solcher Werkzeuge, wie z.B. Cobertura, CodeCover und Emma [7].

Werkzeuge zur Testüberdeckung liefern im Allgemeinen folgende Informationen [13]:

- Quellcodeabdeckung in Prozent
- Identifizieren und markieren ungetesteter Teile
- Ermittlung der Testüberdeckung nach Normen und Standards

Im Weiteren soll auf die zwei Werkzeuge CodeCover und EclEmma genauer eingegangen werden, welche beide als Eclipse-Plugin verfügbar sind:

4.4.1 CodeCover

CodeCover [1] ist ein an der Universität Stuttgart entwickeltes Glass-Box-Test-Werkzeug. Es entstand im Rahmen eines Studienprojekts und wird unter anderem in Form von Abschlussarbeiten aktiv weiterentwickelt. Mit CodeCover können neben der Anweisungs- und Zweigüberdeckung noch fünf weitere Überdeckungsmaße gemessen werden. Zur Zeit werden die Programmiersprachen Java, COBOL und C unterstützt. CodeCover steht unter EPL-Lizenz und ist unter www.codecover.org als Standalone Version, sowie als Eclipse-Plugin frei verfügbar. Mit CodeCover ist es möglich, die Art der Testüberdeckung individuell auszuwählen und nur für vorher definierte Klassen und Pakete messen zu lassen. Die Analyseergebnisse sind beim Eclipse-Plugin direkt in Eclipse einsehbar (siehe Abbildung 4.1) und können zusätzlich als XML- und HTML-File exportiert werden.

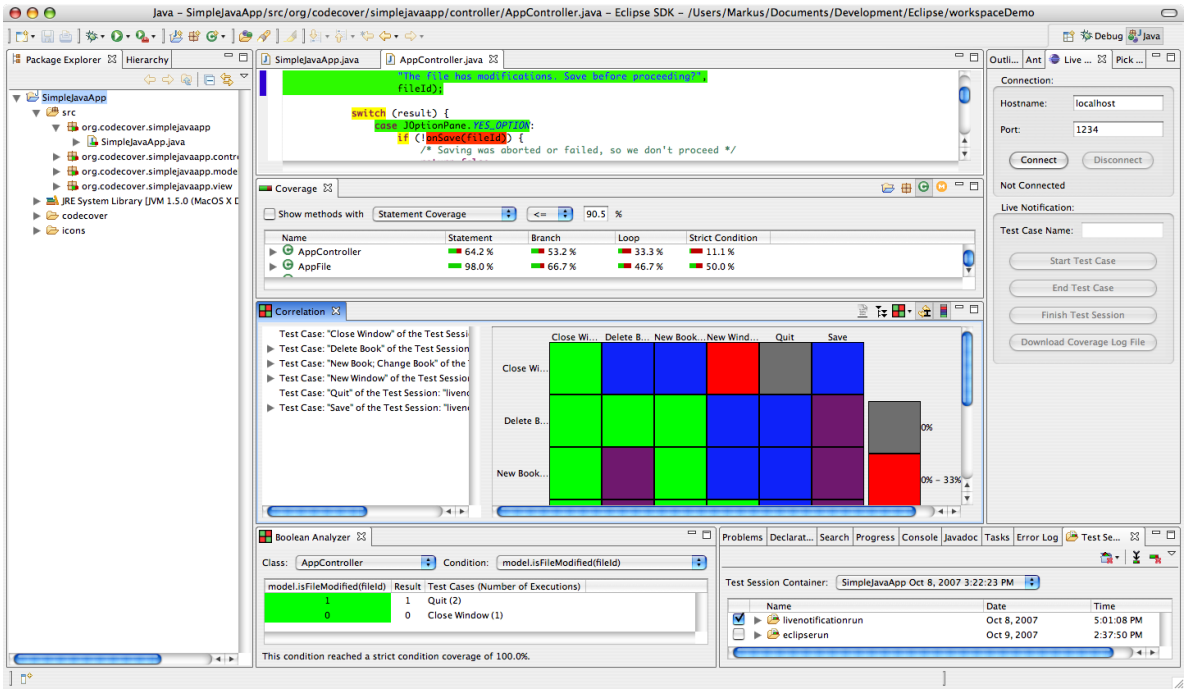

Abbildung 4.1: CodeCover als Eclipse-Plugin [1]

4.4.2 EclEmma

EclEmma [3] ist ein frei verfügbares Eclipse-Plugin, mit dem die Testüberdeckung in Java-Programmen gemessen werden kann. Die Analyse lässt sich über das Menü oder die Run-Workbench Symbolleiste starten. Neben der Zeilen-, Methoden- und Typenüberdeckung kann EclEmma auch die Anweisungs- und Zweigüberdeckung ermitteln. Die Ergebnisse sind nach der Messung direkt im Workbench verfügbar wie in Abbildung 4.2 zu sehen ist. Mit der Exportfunktion von EclEmma lassen sich diese unter anderem als HTML-, XML- oder CSV-Datei exportieren.

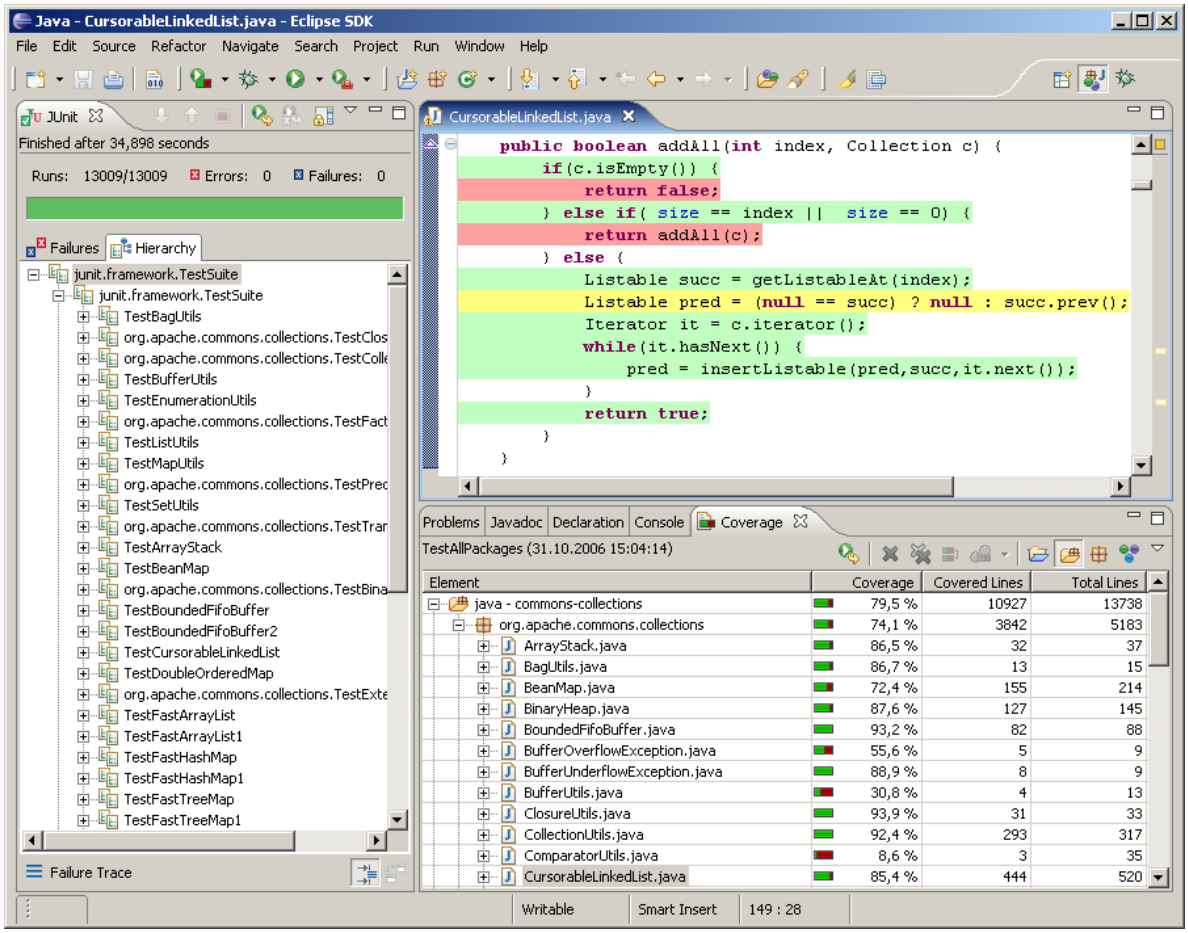

Abbildung 4.2: EclEmma als Eclipse-Plugin [3]

Kapitel 5

Integration

Das folgende Kapitel befasst sich mit der Umsetzung der Integration der Testüberdeckung in Quamoco. Zu Beginn wird das Konzept vorgestellt. Anschließend werden die Erweiterungen des Basismodells erläutert und die Auswirkungen der Testüberdeckung auf die Qualitätsaspekte diskutiert. Zum Schluss wird auf die Änderungen der Werkzeugkette eingegangen.

5.1 Konzept

Um die Testüberdeckung als Maß in Quamoco integrieren zu können, müssen folgende drei Schritte durchgeführt werden:

1. Bestimmung eines Analysewerkzeugs und der Maße
2. Erweiterung des Basismodells
3. Erweiterung der ConQAT-Anbindung

Diese leiten sich aus der Arbeitsweise von Quamoco ab, welche in der Abbildung 5.1 zu sehen ist. In den nächsten Abschnitten werden die einzelnen Schritte genauer erklärt.

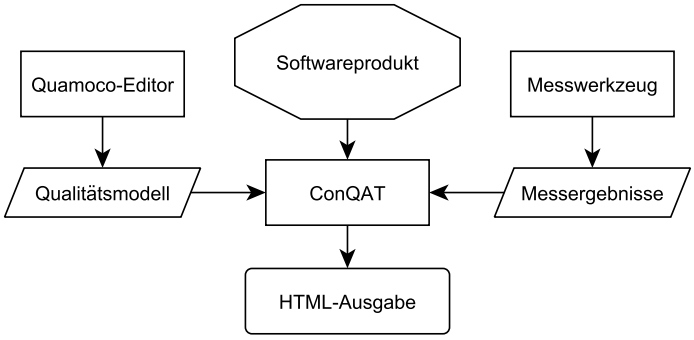

Abbildung 5.1: Das Funktionsprinzip einer Evaluierung mit Quamoco.

Die Evaluierung eines vorgegebenen Softwareprodukts wird mit dem Analysewerkzeug ConQAT durchgeführt. Als Eingabe wird neben dem Produkt selbst, auch das Quamoco-Qualitätsmodell, nach dem die Software bewertet werden soll, benötigt. Enthält das Modell Maße, deren Messwerkzeug nicht von ConQAT zur Analyse angestoßen wird, so ist es notwendig die Messungen vor der Evaluierung durchzuführen. Die Messergebnisse werden ebenfalls ConQAT, als Eingabe, zur Verfügung gestellt. Damit ConQAT diese Messdaten für die Evaluierung weiter verarbeiten kann, ist eine Anpassung der ConQAT-Anbindung notwendig (siehe Kapitel 5.4). Nach der Evaluation gibt ConQAT die Ergebnisse als HTML-Datei aus.

5.2 Bestimmung eines Analysewerkzeugs und der Maße

Analysewerkzeug

Wie bereits in Kapitel 4.4 beschrieben, gibt es eine Vielzahl von Analysewerkzeugen zur Ermittlung der Testüberdeckung. Da die Integration im Rahmen dieser Arbeit sich auf die Messung von Java-Code beschränkt, fiel die erste Wahl auf das Messwerkzeug CodeCover. Die Vorteile des Werkzeugs waren zum einen der Ausschluss ausgewählter Klassen von der Analyse, sowie zum anderen die Möglichkeit durch ein *Open Language Interface*, das Werkzeug auch für andere Programmiersprachen erweitern und verwenden zu können. Zu Beginn der Erprobung stellte sich allerdings heraus, dass CodeCover mit der neu in Java 7 eingeführten Typinferenz und der dadurch geschaffenen Möglichkeit '<>' zu verwenden, nicht zurecht kommt und eine Messung verweigert. Daher wurde für die Messung der Testüberdeckung auf das Analysewerkzeug EclEmma zurückgegriffen. Die Messergebnisse werden in eine CSV-Datei gespeichert. Diese beinhaltet unter anderem vier Spalten, wobei zwei die Anzahl der überdeckten und zwei die Anzahl der nichtüberdeckten Anweisungen bzw. Zweige angibt. Die Zeilen spiegeln die Klassen und innere Klassen des Projekts wider.

Maße

Mit EclEmma lassen sich verschiedene Testüberdeckungsmaße messen. Im Rahmen dieser Bachelorarbeit beschränken wir uns auf die Maße Anweisungs- und Zweigüberdeckung (Kapitel 4.3). Diese Maße gehören zu den am einfachsten zu ermittelnden Überdeckungsmaße und stellen eine ausreichende Grundlage dar.

5.3 Erweiterung des Basismodells

Die Modifizierung des Modells wurde mit dem Qualitätsmodell-Editor (Kapitel 3.3) durchgeführt. Zuerst wurde das *Tool* EclEmma im Modell definiert. Da es sich hierbei um ein Analysewerkzeug für die Programmiersprache Java handelt, stand die Zuordnung zum Modul *Java* außer Frage. Ebenso wie bereits für alle anderen Tools, ungeachtet welche

Programmiersprache sie unterstützen, wurde für EclEmma im Modul Root eine *Source* angelegt. Zudem wurden zwei neue *Measures*, eins für die Anweisungs- und eins für die Zweigüberdeckung im Java Modul definiert, da die Integration für diese Programmiersprache durchgeführt wurde. Des Weiteren wurde für jedes dieser beiden Elemente eine *MeasurementMethod* im Modul Java angelegt, welche das Tool EclEmma mit den Measures verknüpft.
Bevor neue Produktfaktoren für die Maße ins Modell eingefügt werden konnten, war die Frage zu klären, welche der folgenden Aufteilungen am sinnvollsten ist:

1. Ein Produktfaktor Testüberdeckung, der sich aus den Maßen Anweisungs- und Zweigüberdeckung zusammensetzt.
2. Zwei unabhängige Produktfaktoren, einen für die Anweisungs- und einen für die Zweigüberdeckung, die nebeneinander existieren.
3. Drei Produktfaktoren, wobei der Produktfaktor Testüberdeckung, den Subfaktor Anweisungsüberdeckung und den Subfaktor Zweigüberdeckung besitzt.

Um diese Frage beantworten zu können, ist es notwendig die Einflüsse der Testüberdeckung auf die Qualitätsaspekte zu kennen. Beeinflussen beide Überdeckungsmaße dieselben Aspekte, so ist es sinnvoll nur einen gemeinsamen Produktfaktor zu definieren. Dies würde die Anzahl der zu modellierenden Einflüsse halbieren. Beeinflussen hingegen beide Maße unterschiedliche Qualitätsaspekte, so wäre es notwendig zwei Produktfaktoren zu modellieren. Die dritte Variante ist sinnvoll, wenn sowohl beide Überdeckungsmaße gemeinsam Qualitätsaspekte und zusätzlich noch allein Aspekte beeinflussen.
Im Rahmen dieser Bachelorarbeit wurde eine erste Überlegung zu der Beeinflussung der Qualitätsaspekte durch die Testüberdeckung vorgenommen. Zuerst wurde angenommen, dass die Anweisungsüberdeckung andere Qualitätsaspekte beeinflusst, als die Zweigüberdeckung, deshalb wurden im Modell zwei Produktfaktoren definiert. Jedoch konnte nur für die drei Merkmale Reife, Portierbarkeit und Effizienz eine Begründung gefunden werden, dass diese nur von einem der beiden Überdeckungsmaße beeinflusst werden. Nach weiterer Diskussion darüber, stellten sich diese Begründungen allerdings als nicht haltbar heraus. Die Sinnhaftigkeit einer getrennten Betrachtung der Maße ist hiermit jedoch nicht ausgeschlossen. Denn um dies letzten Endes beurteilen zu können, sind umfangreiche Studien und Expertenwissen notwendig. Daher wurden die zwei Produktfaktoren beibehalten, um dadurch so flexibel wie möglich zu bleiben.

Die neu definierten Produktfaktoren wurden im Modul Root integriert, da sie weder von der Programmiersprache, noch von deren Art oder einem speziellen Technologiebereich abhängen. Um eine Verknüpfung zwischen diesen neuen Produktfaktoren und den zuvor modellierten Measures aus dem Java Modul herzustellen, wurde ebenfalls im Modul Java eine Evaluation modelliert.

Einfluss der Testüberdeckung auf die Qualitätsaspekte

Die Testüberdeckung identifiziert Codestellen, die durch UnitTests abgedeckt werden. Bei Fehlschlagen von UnitTests können die hierfür verantwortlichen Codezeilen geändert und so das fehlerhafte Verhalten beseitigt werden. Die Testüberdeckung hat somit einen Einfluss auf Qualitätsaspekte, die durch die Verwendung von UnitTests beeinflusst werden.
Im Weiteren werden nun die Qualitätsaspekte einzeln betrachtet und jeweils begründet, weshalb die Testüberdeckung auf diesen Aspekt einen Einfluss hat oder nicht. Wie schon in Kapitel 3.2 beschrieben, wurden bei der Definition der Qualitätsaspekte, Merkmale des Produktqualitätsmodells der IISO/IEC 25010 für Quamoco verwendet und angepasst. Abbildung 5.2 gibt einen Überblick über die Einflüsse. Grün bedeutet, dieser Aspekt wird beeinflusst, rot er wird nicht beeinflusst.

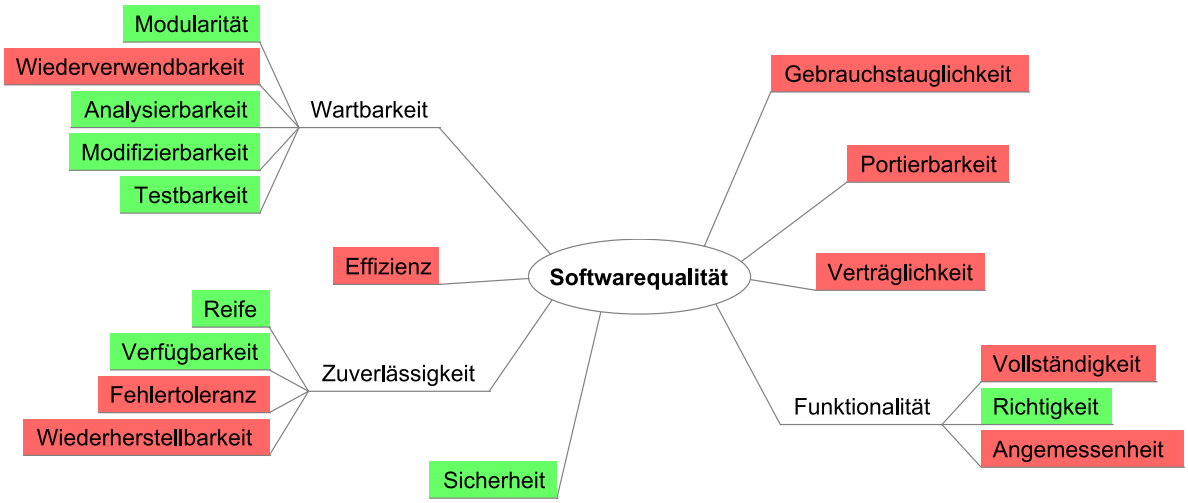

Abbildung 5.2: Übersicht der Einflüsse der Testüberdeckung auf die Qualitätsaspekte von Quamoco.

Gebrauchstauglichkeit

Kann das Produkt bei der Verwendung durch bestimmte Benutzer in einem bestimmten Kontext, die vorgegebenen Ziele effizient, effektiv und zufriedenstellend erfüllen?
Bei der Gebrauchstauglichkeit geht es darum, wie ein Benutzer mit der Software zurechtkommt, z.B. wie leicht sie zu erlernen ist. Aber auch die Ästhetik und die sinnvolle und benutzerfreundliche Gestaltung der Oberfläche beeinflussen die Gebrauchstauglichkeit. Da dies Eigenschaften sind, die nicht mit UnitTests getestet werden können, hat die Testüberdeckung auch keinen Einfluss auf die Gebrauchstauglichkeit.

Portierbarkeit

Wie einfach lässt sich ein System oder eine Komponente effizient und effektiv von einer Hardware, Software oder einer Betriebs- oder Nutzungsumgebung auf eine andere übertragen?
Der Aufwand, um eine Software in eine andere Umgebung übertragen zu können hängt unter anderem von der Kapselung der umgebungsspezifischen Anweisungen ab.

Solche Eigenschaften in der Struktur eines Programmcodes sind nicht durch UnitTests heraus findbar. Auch der Benutzeraufwand zur Installation des Programms beeinflusst dessen Portierbarkeit. Dies kann eben sowenig durch UnitTests getestet werden. Daher beeinflusst die Testüberdeckung auch nicht die Portierbarkeit einer Software.

Verträglichkeit

Können zwei oder mehr Systeme oder Softwarekomponenten in der selben Hardware oder Softwareumgebung, Informationen austauschen oder erforderliche Funktionen ausführen, ohne dass sich die Programme gegenseitig beeinträchtigen?
Um eine Aussage über die Verträglichkeit der Software mit anderen Softwareprodukten machen zu können, bedarf es Tests in der die beiden Programme parallel ausgeführt werden. Mit UnitTests hingegen werden nur die funktionalen Eigenschaften einer Software getestet. Daher hat die Testüberdeckung keinerlei Auswirkungen auf die Verträglichkeit.

Funktionalität

- **Vollständigkeit**
  Erfüllt die Menge an Funktionen alle spezifizierten Aufgaben und Benutzerziele?
  Es wird angenommen, dass alle Anforderungen des Benutzers spezifiziert und die UnitTests alle in der Spezifikation angegebenen Funktionalitäten abdecken. Die Vollständigkeit ist gegeben, sobald mit den gegebenen UnitTests keine Fehler mehr gefunden werden. Die Höhe der Testüberdeckung ist hierbei nicht von Bedeutung. Auch bei niedriger Überdeckung kann eine hundertprozentige Vollständigkeit gegeben sein. Die geringe Überdeckung kann durch defensiven Code hervorgerufen werden.

- **Richtigkeit**
  Liefert das Produkt die geforderten Ergebnisse mit der benötigten Genauigkeit?
  Die Richtigkeit wird von der Testüberdeckung beeinflusst. Wird eine Funktion überdeckt, so wurde sie mindestens einmal von einem UnitTest ausgeführt. Wurde mit einem UnitTest ein Fehlverhalten gefunden, so ist davon auszugehen, dass die fehlerhafte Funktion verbessert wurde und nun wie gewünscht funktioniert. Je höher nun die Testüberdeckung ist, desto mehr Funktionen wurden getestet, daher ist die Wahrscheinlichkeit größer, dass die implementierten Funktionen auch die geforderten Ergebnisse liefern.

- **Angemessenheit**
  Eignet sich die Menge der Funktionen für die spezifizierte Aufgabe und Benutzerziele, d.h. sind sie z.B. aus aufgabenorientierten Teilfunktionen zusammengesetzt?
  Die Angemessenheit von Funktionen ist von der Struktur dieser abhängig. Wird z.B. eine bestimmte Folge von Anweisungen von mehreren Methoden verwendet, so ist es sinnvoll diese in eine eigene Methode auszulagern. Da sich durch das Testen mit UnitTests die Programmstruktur einer Software nicht ändert, hat die Testüberdeckung auch keinen Einfluss auf die Angemessenheit.

Wartbarkeit

- **Modularität**
  Wie gut teilt sich ein System oder ein Computerprogramm in separate Komponenten auf, so dass eine Änderung an einer Komponente nur eine minimale Auswirkung auf andere Komponenten hat?
  Die Testüberdeckung hat einen Einfluss auf die Modularität, sofern man immer nur die UnitTests einer Softwarekomponente ausführt und die Überdeckung des gesamten Programmcodes betrachtet. Bei diesen UnitTests werden nur Funktionen der zu testenden Komponente aufgerufen, wird nun eine Überdeckung in anderen Komponenten festgestellt, so ist eine Abhängigkeit der Komponenten vorhanden. Je niedriger nun die Testüberdeckung in den anderen Komponenten ist, desto wahrscheinlicher ist es, dass die getestete Komponente gut gekapselt ist.

- **Wiederverwendbarkeit**
  Ist das Programm so aufgebaut, dass Komponenten davon in mehr als einem System oder beim Bau von anderen Programmteilen wiederverwendet werden können?
  Die Wiederverwendbarkeit hängt hauptsächlich von der Struktur und dem abstrakten Charakter einer Softwarekomponente ab. Da sich dies mit UnitTests nicht testen lässt, wird die Wiederverwendbarkeit auch nicht von der Testüberdeckung beeinflusst.

- **Analysierbarkeit**
  Wie einfach lassen sich Auswirkungen von Änderungen auf den Rest des Produkts feststellen, und wie leicht können fehlerhafte Programmstellen ausfindig gemacht werden?
  Die Analysierbarkeit wird von der Testüberdeckung beeinflusst. Je größer die Überdeckung ist, desto kleiner ist der potenzielle Bereich für Fehler, vorausgesetzt mit den vorhandenen UnitTests können keine weiteren Fehler diagnostiziert werden. Je kleiner der Bereich in dem ein Fehler erwartet wird, desto geringer ist auch der Aufwand ihn zu finden. Daher nimmt mit zunehmender Testüberdeckung die Analysierbarkeit zu.

- **Modifizierbarkeit**
  Wie einfach lässt sich das Produkt effizient und effektiv ändern, ohne dabei Fehler oder einen Leistungsabfall zu verursachen?
  Die Modifizierbarkeit wird durch die Testüberdeckung beeinflusst. Je höher die Überdeckung ist, desto wahrscheinlicher ist es, dass die zu ändernde Stelle durch einen UnitTest abgedeckt wird. Die Korrektheit einer Änderung kann somit direkt von diesem UnitTest überprüft werden. Bei neu umgesetzten Anforderungen gibt es unabhängig von der Höhe der Testüberdeckung, noch keinen UnitTest. Allerdings kann mit den vorhandenen UnitTests, eine negative Beeinflussung der anderen Funktionen durch die neue Programmstelle direkt identifiziert werden. Somit ist auch hier eine hohe Testüberdeckung von Vorteil.

- **Testbarkeit**
  Wie einfach lassen sich für ein System oder Komponenten Testkriterien einführen und Tests zur Überprüfung auf Einhaltung der Kriterien durchführen?

Auch bei einer Software, die schwer zu testen ist, kann eventuell mit viel Aufwand eine hohe Testüberdeckung erreicht werden. Die Wahrscheinlichkeit ist jedoch größer, dass wenn eine hohe Testüberdeckung erzielt wurde, die Testbarkeit ebenfalls hoch ist.

Effizienz

Wie verhält sich die Leistung in Bezug auf die spezifizierte Ressourcenmenge?
Die Effizienz ist abhängig von der Benutzung der Hardwareressourcen und dem Zeitverhalten bei der Ausführung von Funktionen. Um die Effizienz eines Systems testen zu können ist es erforderlich, das System im Ganzen zu betrachten. Dies ist bei den UnitTests nicht gegeben, daher hat auch die Testüberdeckung keinen Einfluss auf die Effizienz.

Zuverlässigkeit

- **Reife**
  Wie viele Anforderungen an die Zuverlässigkeit werden bei normalen Betriebsbedingungen erfüllt?
  Die Testüberdeckung beeinflusst die Reife einer Software. Denn je höher die Überdeckung ist, desto mehr Code wurde mit UnitTests getestet und desto geringer ist die Wahrscheinlichkeit, dass das Programm Fehler enthält, somit ist das Programm reifer.

- **Verfügbarkeit**
  Sind ein System oder Komponenten betriebsbereit und zugänglich, wenn sie benötigt werden?
  Die Verfügbarkeit hängt zum einen von äußeren Faktoren, wie z.B. der Verfügbarkeit von Netzwerkverbindungen, der Hardware oder dem Vorhandensein von Strom ab. Diese Faktoren lassen sich nicht durch UnitTests testen. Allerdings können zum anderen auch fehlerhafte Funktionen dazu führen, dass die Software nicht stabil läuft und somit zeitweise nicht verfügbar ist. Bei einer hohen Testüberdeckung ist allerdings die Wahrscheinlichkeit solcher Fehler geringer, da potenziell weniger Fehler im Code vorhanden sind. Daher hat die Testüberdeckung einen Einfluss auf die Verfügbarkeit.

- **Fehlertoleranz**
  Kann ein System oder Komponenten trotz der Anwesenheit von Hard- oder Softwarefehlern arbeiten?
  Um dies beurteilen zu können, ist es notwendig das gesamte System zu betrachtet und nicht wie bei den UnitTests nur einzelne Funktionen. Daher hat die Testüberdeckung keinen Einfluss auf die Fehlertoleranz.

- **Wiederherstellbarkeit**
  Wie einfach kann das Produkt bei einer Unterbrechung oder einem Ausfall, ein stabiles Leistungsnieveau und die direkt betroffenen Daten wiederherstellen?
  Ebenso wenig wie die Testüberdeckung auf die funktionale Vollständigkeit keinen Einfluss hat, verhält es sich auch bei der Wiederherstellbarkeit. Sind hierfür keine Funktionen implementiert die dafür sorgen, dass z.B. die Daten gesichert werden,

so kann die Testüberdeckung noch so hoch sein, die Daten lassen sich nicht wiederherstellen.

Sicherheit

Sind Informationen und Daten vor der Einsicht und Änderung von unberechtigten Personen und Systemen geschützt und wird der Zugang autorisierten Personen und Systemen nicht verweigert?
Die Testüberdeckung beeinflusst die Sicherheit eines Systems. Denn je höher die Überdeckung ist, desto wahrscheinlicher ist es, dass die Funktionen, welche Berechtigungen verwalten, mit UnitTests getestet wurden und gemäß der Spezifikation funktionieren.

5.4 Erweiterung der ConQAT-Anbindung

Zur Integration der Testüberdeckung war es notwendig die ConQAT-Anbindung zu erweitern. Eine Anpassung des Qualitätsmodell-Editors war nicht erforderlich.

Die Analysekonfiguration von Quamoco wurde um einen neuen Block für das Messwerkzeug EclEmma erweitert. Dieser hat im Block *NightlyQuamocoGeneratedJava* die Kette der Werkzeugblöcke um eine Glied erweitert. Diese Blöcke enthalten die Information, wo die Messdaten gespeichert sind, lesen diese ein und verarbeiten sie. Bei manchen Messwerkzeugen wird zuvor noch das Werkzeug zur Messung angestoßen.

Wie in Abbildung 5.3 zu sehen ist, besteht der Block für EclEmma aus zwei Eingaben, einer Ausgabe und zwei Prozessoren.

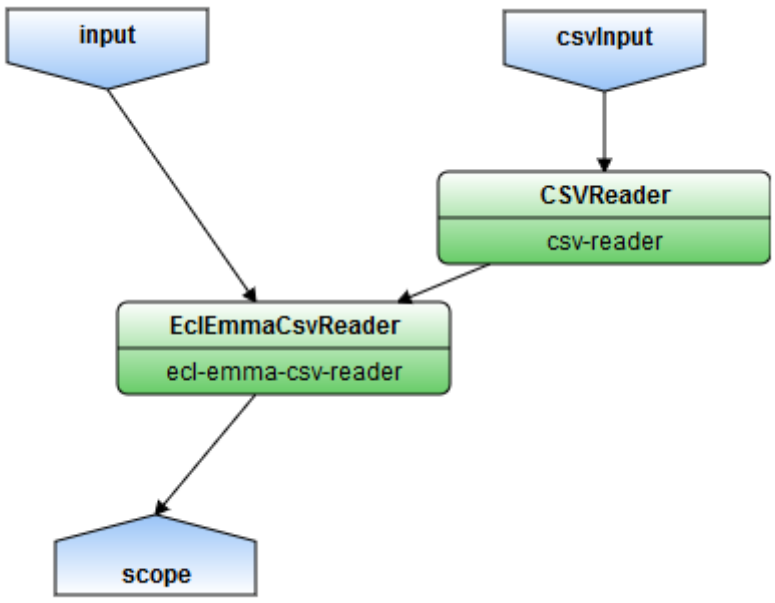

Abbildung 5.3: ConQAT-Block für EclEmma

Bei dem CSV-Reader handelt es sich um einen schon in ConQAT implementierten Prozessor, mit dem die in einer CSV-Datei vorliegenden Messergebnisse der Überdeckungsmessung eingelesen und in einer Liste abgespeichert werden. Im Block *NightlyQuamocoGeneratedJava* wird der Pfad zu der CSV-Datei mit den Messergebnissen definiert und mit der Eingabe *csvInput* an den CSVReader weitergegeben. Der Prozessor *EclEmmaCsvReader* hingegen wurde extra für die Verarbeitung der EclEmma-Messergebnisse geschrieben. Er ist ein Pipeline-Prozessor, d.h. er bekommt einen Datenstrom über die Eingabe *input* und gibt diesen nach Änderung, als Ausgabe *scope* weiter. Als zusätzliche Eingabe erhält der Prozessor eine Liste mit den Messdaten, die zuvor mit dem CSVReader eingelesen wurden. Der Prozessor berechnet für die Klassen und Pakete des Softwareprodukts die Anzahl und den Prozentwert der Testüberdeckung. Dazu überträgt er die Liste in eine Tabelle. Für jedes Element im Datenstrom wird überprüft, ob dieses auch in der Tabelle vorhanden ist. Falls ja, werden die Einträge für die Klassen und Pakete aus den Daten berechnet und zusätzlich für jedes Element der Prozentwert der Überdeckung ermittelt. Gibt es für ein Element kein Eintrag in der Tabelle, so wird der Prozentwert auf -1 und die restlichen Spalten auf null gesetzt. Dies ermöglicht es direkt in der Ausgabedatei zu erkennen, für welche Klassen es keine Messwerte gibt. Diese Elemente werden bei der Gesamtprozentberechnung selbstverständlich nicht mit berücksichtigt. Dem Datenstrom werden anschließend sechs neue Spalten (siehe Abb. 5.4) hinzugefügt und die berechneten Werte eingetragen. Durch entsprechende Bezeichnungen der Spalten wird eine Verknüpfung zu den Maßen des Basismodells hergestellt.

PERCENTAGE_INSTRUCTION_COVERAGE	INSTRUCTION_MISSED	INSTRUCTION_COVERED	PERCENTAGE_BRANCH_COVERAGE	BRANCH_MISSED	BRANCH_COVERED	Sear
0,415	598.806	425.161	0,275	57.694	21.938	
-1	0	0	-1	0	0	
-1	0	0	-1	0	0	
-1	0	0	-1	0	0	
-1	0	0	-1	0	0	
-1	0	0	-1	0	0	
-1	0	0	-1	0	0	
-1	0	0	-1	0	0	
-1	0	0	-1	0	0	
0	86	0	-1	0	0	
0	86	0	-1	0	0	
0	86	0	-1	0	0	
0	86	0	-1	0	0	
0	86	0	-1	0	0	
0	86	0	-1	0	0	
0	26	0	-1	0	0	
0	34	0	-1	0	0	
0	26	0	-1	0	0	
0,415	598.686	425.141	0,275	57.694	21.938	
0,417	588.819	420.549	0,274	56.682	21.430	
0,003	50.981	128	0	5.194	1	

Abbildung 5.4: HTML-Ausgabe der Evaluation mit den neuen Spalten für die Testüberdeckung.

Kapitel 6

Evaluation

Dieses Kapitel beschäftigt sich mit der Evaluierung des erweiterten Qualitätsmodells und diskutiert die Auswirkungen der Integration der Testüberdeckung.

6.1 Grundlagen

Um das erweiterte Qualitätsmodell zu beurteilen, wurde eine Evaluation eines Softwareprodukts mit diesem Modell durchgeführt. Das hierfür verwendete Softwareprodukt stellte ein deutsches mittelständisches Unternehmen zur Verfügung. Bei dem Produkt handelt es sich um ein Projekt in Java 7, welches aus einer Reihe von Unterprojekten besteht. Insgesamt umfasst es um die 762.000 Lines of Code (gemessen mit ConQAT-Prozessor LOCAnalyzer) und ca. 7000 Klassen. Die Testsuite mit den UnitTests, von denen ausgehend die Testüberdeckung gemessen wurde, enthält um die 800 Testklassen.

6.2 Messung der Testüberdeckung

Vor der Evaluierung des Softwareprodukts, wurde zuerst die Testüberdeckung gemessen. Hierzu wurde das Eclipse-Plugin von CodeCover verwendet. Allerdings stellte sich heraus, dass CodeCover mit der in Java 7 eingeführten Typinferenz, nicht zurecht kommt und daher die Messung nicht durchführen kann. Daher wurde die ConQAT-Anbindung auf EclEmma umgestellt, welches auch bei Programmcode in Java 7, die Testüberdeckung messen kann. Es wurden mit dem Eclipse-Plugin die Testüberdeckung ermittelt, anschließend die Messergebnisse als CSV-Datei exportiert und der Dateipfad ConQAT bekannt gegeben. Als Ergebnis wurde eine Anweisungsüberdeckung von 41,5 % und eine von Zweigüberdeckung 27,5% gemessen.

6.3 Qualitätsanalyse

In der ersten Analyserunde wurde zur Qualitätserhebung das anfängliche Basismodell und die ursprüngliche Werkzeugkette verwendet. Da zur Ausführung im Qualitätsmodell-Editor nur ein Pfad für das zu testende Programmcode angegeben werden kann und sich die Testklassen des Gesamtprojekts in diversen Unterprojekten befinden, wurden diese explizit von der Analyse ausgeschlossen. Des Weiteren verhinderte JavaDoc, ein weiteres Messwerkzeug, welches bei der Analyse von ConQAT zur Messung verwendet wird, eine reibungslose Durchführung der Evaluation. Aus diesem Grund wurde der Block für JavaDoc auskommentiert. Eine weitere Einstellung die notwendig für die Durchführung der Evaluation war, war die Erhöhung des Arbeitsspeichers.
Nachdem die anfänglichen Schwierigkeiten behoben wurden, konnte eine Qualitätsbeurteilung durchgeführt werden. Im Anschluss wurden zwei weitere Evaluationen mit dem geänderten Basismodell und der erweiterten ConQAT-Anbindung durchgeführt. Die zuvor durchgeführten Änderungen wurden auch hier vollzogen. In der ersten Runde wurden für folgende Einflüsse immer das niedrigste Ranking in der Aggregation und beim zweiten Durchlauf immer das höchste Ranking eingestellt. Je niedriger das Ranking eingestellt ist, desto einen geringeren Anteil an Einfluss hat der Produktfaktor im Vergleich zu den anderen Faktoren, welche den Qualitätsaspekt beeinflussen.

Zur Analyse wurden folgende Einflüsse im Basismodell definiert:

- Funktionalität - Richtigkeit (A und Z)
- Portierbarkeit (A)
- Zuverlässigkeit (A und Z)
- Wartbarkeit - Analysierbarkeit (A und Z)
- Wartbarkeit - Modizierbarkeit (A und Z)
- Wartbarkeit - Testbarkeit (A und Z)

(A = Anweisungsüberdeckung, Z = Zweigüberdeckung)

6.4 Interpretation und Diskussion der Ergebnisse

Wie man der Tabelle 6.1 entnehmen kann, hat die Testüberdeckung einen größeren Einfluss auf die Gesamtqualität, wenn ein höheres Ranking eingestellt wird. Bei den in der Tabelle angegebenen Werten handelt es sich um Schulnoten, somit ist eine 1,0 die bestmögliche und eine 6,0 die schlechteste Bewertung. Allerdings sind die Messewerte, bei denen die Testüberdeckung berücksichtigt wurde, im Vergleich zu denen ohne die Testüberdeckung erheblich schlechter. Da die Testüberdeckung sowohl bei der Anweisungs- als auch bei der Zweigüberdeckung kleiner als fünfzig Prozent ist, war davon auszugehen, dass durch die Berücksichtigung der Testüberdeckung die Qualität schlechter bewertet wird. Solch eine

große Abweichung ist allerdings sehr überraschend. Sie kann ein Zeichen dafür sein, dass die Einflüsse noch nicht genau genug eingestellt sind. Um eine sinnvolle Einstellung vornehmen zu können, bedarf es allerdings einer genaueren Untersuchung der Einflüsse mit vielen Testdaten und dem Wissen und der Erfahrung von Experten.

	niedrigstes Ranking	**höchstes Ranking**	**ohne Test-überdeckung**
Qualität	4,183	5,784	1,082
Funktionalität	3,880	6,000	1,002
Funktionalität - Richtigkeit	3,880	6,000	1,002
Portierbarkeit	3,892	6,000	1,000
Zuverlässigkeit	3,610	5,526	1,000
Wartbarkeit	4,302	6,000	1,529
Wartbarkeit – Analysierbarkeit	3,647	5,192	1,774
Wartbarkeit – Modifizierbarkeit	6,000	6,000	1,947
Wartbarkeit – Testbarkeit	3,882	6,000	1,000

Abbildung 6.1: Übersicht der Messergebnisse der Qualitätsanalyse.

Kapitel 7

Fazit

Dieses Kapitel fasst die vorgestellte Arbeit zusammen und gibt einen Überblick auf mögliche Anknüpfungspunkte.

7.1 Zusammenfassung

Diese Bachelorarbeit beschäftigte sich mit der Integration der Testüberdeckung als Maß in Quamoco. Durch die Hinzunahme des neuen Qualitätsmaß, wurde das Basismodell weiter verfeinert und somit eine präzisere Qualitätsbestimmung möglich. Der Schwerpunkt dieser Arbeit lag in der Integration der Testüberdeckung in die Werkzeugkette. Als weiterer Punkt wurde die Einflussnahme der Testüberdeckung auf die Qualitätsaspekte diskutiert.

Zu Beginn dieser Arbeit wurden der Begriff *Softwarequalität* und die Konzepte *Qualitätsmodelle* und *Softwaremetriken* vorgestellt. Anschließend fand eine Beschreibung des Forschungsprojekts *Quamoco* und dessen Konzepte und Ergebnisse statt. Im Weiteren wurde näher auf Softwaretests und insbesondere auf die Testüberdeckung eingegangen. Um diese im Basismodell zu integrieren, wurde mit dem Qualitätsmodell-Editor dieses um neue Elemente, wie die Maße Anweisungs- und Zweigüberdeckung, erweitert. Zudem wurde die ConQAT-Anbindung um einen neuen Block für das Messwerkwerkzeug EclEmma und einen Prozessor, zur Verarbeitung der Messergebnisse ergänzt. Des Weiteren wurde über die Auswirkungen der Testüberdeckung auf die Qualitätsaspekte diskutiert. Abschließend wurde das erweiterte Modell an einem Softwaresystem eines Unternehmens getestet. Während der Erprobung stellte sich heraus, dass eine ausführliche Untersuchung der Auswirkungen der Testüberdeckung auf die Qualitätsaspekte notwendig ist, um die Testüberdeckung als Qualitätsmaß mit in der Evaluation berücksichtigen zu können.

7.2 Ausblick

Wie in Kapitel 6.4 beschrieben, ist die Integration noch nicht hundertprozentig abgeschlossen. Um die Testüberdeckung als Qualitätsmaß in die Qualitätsmessung aufnehmen zu können

ist es notwendig, die Auswirkungen der Testüberdeckung auf die Qualitätsaspekte genauer zu untersuchen und mit der Modellierung der Einflüsse im Basismodell zu experimentieren. Hierbei ist zu überlegen, ob die in Kapitel 5.3 beschriebene Aufteilung der Testüberdeckung in mehrere Produktfaktoren, durch eine andere ersetzt werden sollte. Dies Bedarf allerdings einer umfangreicheren Studie, bei der unterschiedliche Softwareprodukte evaluiert und die Ergebnisse durch Experten ausgewertet und beurteilt werden müssen.
Ein weiterer Anknüpfungspunkt an diese Arbeit ist die Anpassung des Qualitätsmodell-Editors. Dieser könnte um ein Eingabefeld erweitert werden, so dass der Dateipfad der Messwerte direkt über den Editor angegeben werden kann und nicht wie bisher in einem ConQAT-Block.
Bei der Erstellung dieser Arbeit zeigten sich weitere Aspekte, die für Quamoco sinnvoll erscheinen. Diese könnten in einer zukünftigen Weiterentwicklung von Quamoco integriert werden. Hierbei handelt es sich zum einen um die Berücksichtigung von weiteren Testüberdeckungsmaßen. Zum anderen um die Integration von Testüberdeckungswerkzeugen für die Programmiersprache C#.

Zusammenfassend lässt sich sagen, dass die Weiterentwicklung von Quamoco ein wichtiger Schritt für die Entwicklung der Softwarequalität ist.

Literaturverzeichnis

[1] *CodeCover*. http://codecover.org/ (Zitiert auf den Seiten 7 und 24)

[2] *ConQAT*. https://www.cqse.eu/en/products/conqat/overview/ (Zitiert auf Seite 19)

[3] *EclEmma - Java Code Coverage for Eclipse*. http://www.eclemma.org/ (Zitiert auf den Seiten 7 und 25)

[4] *Find Bugs in Java Programs*. http://findbugs.sourceforge.net/ (Zitiert auf den Seiten 13, 15 und 18)

[5] *Gendarme*. http://www.mono-project.com/Gendarme (Zitiert auf den Seiten 13, 15 und 18)

[6] *ISO/IEC 25010:2011 Systems and software engineering - Systems and software product Quality Requirements and Evaluation (SQuaRE) - System and software quality models* (Zitiert auf den Seiten 7, 12 und 13)

[7] *Open Source Code Coverage Tools in Java*. http://java-source.net/open-source/code-coverage (Zitiert auf Seite 23)

[8] *Software-Metrik*. http://www.itwissen.info/definition/lexikon/Software-Metrik-software-metric.html (Zitiert auf Seite 13)

[9] Bibliographisches Institut GmbH, Dudenverlag: *Duden*. http://www.duden.de/rechtschreibung/Qualitaet (Zitiert auf Seite 11)

[10] Broy, M. ; Jarke, M. ; Nagl, M. ; Rombach, H. D. u. a.: Dagstuhl-Manifest zur Strategischen Bedeutung des Software Engineering in Deutschland. In: *Perspectives Workshop*, Informatik Spektrum : 29, 2006 (3), S. 210 – 221 (Zitiert auf Seite 9)

[11] Dijkstra, E. W.: The Humble Programmer. In: *Commun. ACM* (1972), S. 859–866 (Zitiert auf Seite 21)

[12] Fraunhofer-Institut für Experimentelles Software Engineering: *Software-Qualitätsmodelle für die Praxis*. http://www.iese.fraunhofer.de/de/customers_industries/automotive/referenzprojekt_quamoco.html (Zitiert auf Seite 15)

[13] Gürkan, A. ; Hartmut, P. : Code Coverage - Tools. In: *Haking EXTRA* (06/2012) (Zitiert auf Seite 24)

[14] Kelter, U. : Software-Qualitätsmodelle. (2007) (Zitiert auf Seite 12)

[15] Liggesmeyer, P. : *Software-Qualität - Testen, Analysieren und Verifizieren von Software (2. Aufl.)*. Spektrum Akademischer Verlag, 2009 (Zitiert auf den Seiten 11, 21, 22 und 23)

[16] Ludewig, J. ; Lichter, H. : *Software Engineering - Grundlagen, Menschen, Prozesse, Techniken*. dpunkt.verlag, 2007 (Zitiert auf den Seiten 11, 13, 21 und 23)

[17] Mandau, M. : *Die größten Software-Desaster*. http://www.focus.de/digital/computer/chip-exklusiv/tid-14183/computer-fehler-die-groessten-software-desaster_aid_396628.html (Zitiert auf Seite 9)

[18] Myers, G. J. ; Sandler, C. : *The Art of Software Testing*. John Wiley & Sons, 2004 (Zitiert auf Seite 21)

[19] Schneider, K. : *Abenteuer Softwarequalität: Grundlagen und Verfahren für Qualitätssicherung und Qualitätsmanagement (2. Aufl.)*. dpunkt.verlag, 2012 (Zitiert auf den Seiten 13 und 21)

[20] Sommerville, I. : *Software Engineering*. Pearson, 2007 (Zitiert auf Seite 13)

[21] Wagner, S. : *Softwarequalitätt erfassen und verlässlich vergleichen*. http://newsletter.capgemini.de/mobile-fitness-energieversorgung-und-softwarequalitaet-22012/softwarequalitaet-erfassen-und-vergleichen (Zitiert auf Seite 15)

[22] Wagner, S. : *Software Product Quality Control*. Springer, 2013 (Zitiert auf den Seiten 7, 15, 16 und 17)

[23] Wagner, S. ; Lochmann, K. ; Heinemann, L. u. a.: *Practical Product Quality Modelling and Assessment: The Quamoco Approach*. 2013 (Zitiert auf den Seiten 7 und 18)

[24] Wikipedia, Die freie Enzyklopädie: *Softwarequalität*. http://de.wikipedia.org/wiki/Softwarequalität (Zitiert auf den Seiten 7 und 12)

Alle URLs wurden zuletzt am 25.05.2014 geprüft.

Erklärung

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

(Stefanie Dressel)