Diplomarbeit

Bewertung von softwaretechnischen Prinzipien auf der Basis historischer Daten am Beispiel von Eclipse

Nadejda Nikitina

24. Februar 2007
1. Gutachter: Prof. Dr. Christiane Floyd
2. Gutachter: Prof. Dr. Michalichek
Betreuerin: Dipl.-Inf. Carola Lilienthal

Erklärung
Die vorliegende Diplomarbeit wurde von mir selbstständig angefertigt. Die verwendeten Hilfsmittel und Quellen sind im Literaturverzeichnis vollständig aufgeführt.

Hamburg, 24.02.2007

"When you can measure what you are speaking about, and express it into numbers, you know something about it, when you cannot measure it, when you cannot express it in numbers, your knowledge is of meager and unsatisfactory kind; it may be the beginning of knowledge, but you have scarcely in your thoughts advanced to the stage of science."

(Lord Kelvin)

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Kapitel 1

Einleitung

Wie die Praxis der Softwareentwicklung zeigt, enthalten große Softwaresysteme[1] immer Fehler. Diese können in unterschiedlichen Phasen bei der Erstellung von Systemen entstehen. So können Spezifikationsfehler, Entwurfsfehler oder Implementierungsfehler die Qualität von Systemen beeinträchtigen.

Um eine möglichst hohe Qualität von Systemen zu erzielen, wird stets versucht, diese so fehlerfrei wie möglich zu gestalten. Ein Mittel dazu ist die Anwendung von softwaretechnischen Prinzipien. Das sind Empfehlungen, deren Befolgung die Qualität der entwickelten Software beziehungsweise die Produktivität bei der Entwicklung fördern sollen. Bei einem Prinzip können sowohl die Eigenschaften des Entwicklungsprozesses als auch die des Systems im Mittelpunkt der Betrachtung stehen. Auf Prinzipien bauen softwaretechnische Methoden und sogar ganze Vorgehensmodelle auf.

Um einen möglichst hohen positiven Effekt bei der Befolgung der softwaretechnischen Prinzipien zu erzielen, müssen aus dem breiten Repertoire an Prinzipien entsprechend den Anforderungen die geeigneten ausgewählt werden. Zusätzlich zu den Anforderungen spielen die während der Entwicklung vorliegenden Bedingungen bei der Auswahl der einzelnen Prinzipien eine Rolle, beispielsweise die Zusammensetzung des Teams oder die Größe des zu entwickelnden Systems. Da die Bedingungen in einem konkreten Entwicklungsprojekt häufig in zu vielen Aspekten von denen abweichen, unter welchen ein Prinzip als hilfreich bewertet wurde, ist es oft nicht klar, ob die Befolgung eines bestimmten Prinzips zu einer Verbesserung des Systems beziehungsweise des Entwicklungsprozesses beitragen wird oder eher einen wirtschaftlichen Schaden nach sich ziehen wird.

Um die Entscheidung für oder gegen einzelne Prinzipien zu erleichtern, müssen diese auf eine objektive Weise unter unterschiedlichen Bedingungen bewertet werden. Im Rahmen dieser Arbeit wurde eine Fallstudie durchgeführt mit dem Ziel, zwei bekannte Prinzipien anhand der Daten des Systems *Eclipse* zu bewerten. Eclipse ist eine viel genutzte Programmierumgebung, deren Quelltext und Fehlerdatenbank frei verfügbar sind.

[1] In dieser Arbeit wird vereinfachend das Wort *System* anstelle von *Softwaresystem* benutzt.

Eins der untersuchten Prinzipien, die Zyklenfreiheit von Systemen, wurde bereits vor langer Zeit formuliert, ist jedoch ein wichtiges Thema in der Forschung geblieben. Es betrifft die Gestaltung des Systems. Dabei geht es darum, dass die Bestandteile des entwickelten Systems bezüglich ihrer Benutzungsbeziehungen hierarchisch angeordnet sein sollen. Zur Bewertung dieses Prinzips wurden bereits einige Studien durchgeführt. Diese Arbeit geht über diese Studien hinaus, da der Schwerpunkt dabei auf einer sorgfältigen Definition der Maße liegt.

Das zweite untersuchte Prinzip, Collective Code Ownership, betrifft die Handhabung von Änderungsrechten der Entwickler. Es liegt vor, wenn alle Entwickler alle Bestandteile des entwickelten Systems ändern dürfen. Collective Code Ownership bezieht sich somit auf die Gestaltung des Entwicklungsprozesses. Dieses Prinzip ist in den neunziger Jahren formuliert worden und wurde bisher noch nicht systematisch untersucht. Es ist Gegenstand kontroverser Diskussionen und wird von einigen Softwaretechnikern nicht akzeptiert. Aus diesem Grund wird es in dieser Arbeit untersucht.

Eine Möglichkeit, die Wirtschaftlichkeit derartiger Prinzipien unter bestimmten Bedingungen zu bewerten, besteht darin, Hypothesen über diese Prinzipien zu formulieren und zu überprüfen. Für eine Überprüfung der Hypothesen müssen bestimmte Merkmale des System beziehungsweise des Entwicklungsprozesses gemessen werden.

Diese Vorgehensweise wurde auch in meiner Arbeit verfolgt. Zunächst wurden Theorien darüber entwickelt, wie sich diese Prinzipien positiv oder negativ auf den Grad des Qualitätsmangels auswirken, der sich in Form von Fehlern äußert. Dabei wurde nach Merkmalen gesucht, welche diese positive beziehungsweise negative Wirkung aufzeigen könnten. Auf der Basis dieser Theorien wurden Hypothesen aufgestellt, die überprüft werden sollten. Anschließend wurde überlegt, durch welche Daten die zu untersuchenden Merkmale charakterisiert werden können. Dabei waren einige vereinfachende Annahmen erforderlich. Um zu entscheiden, welche Annahmen angemessen sind, wurden die vorliegenden Daten analysiert. Ausgehend von diesen Überlegungen wurden Maße definiert. Nach der Definition der Maße wurden die für die Untersuchung der Hypothesen erforderlichen Daten bereitgestellt. Die Daten mussten mit Hilfe unterschiedlicher Werkzeuge aufbereitet werden. Dafür wurden im Rahmen dieser Arbeit mehrere Skripte in der Programmiersprache *Perl* implementiert.

Anhand der vorliegenden Daten wurden die Hypothesen mit Hilfe der statistischen Verfahren Parametervergleich und Korrelationsanalyse untersucht. Diese Verfahren zielen darauf ab, den Zusammenhang zwischen statistischen Variablen zu untersuchen.

Die Analyseergebnisse wurden im Hinblick auf deren Gültigkeit kritisch geprüft. Dabei wurde auf die Annahmen im Rahmen dieser Studie und technische Probleme eingegangen.

Aufbau der Arbeit

In **Kapitel 2** werden zunächst die Begriffe *Fehler* und *Fehleranfälligkeit* geklärt. Anschließend wird der Gegenstand der Arbeit beschrieben. Nach einer Darstellung der zu bewertenden Prinzipien wird in diesem Kapitel eine Theorie über den Einfluss dieser Prinzipien auf die Qualität von Systemen entwickelt. Auf der Basis dieser Theorien werden Hypothesen aufgestellt, die auf eine Bewertung der beschriebenen Prinzipien abzielen.

In **Kapitel 3** wird das dieser Studie zu Grunde liegende System *Eclipse* beschrieben. Es werden dabei zahlreiche Gründe genannt dafür, warum dieses als Basis für diese Studie besonders gut geeignet ist. Darüber hnaus werden die Datenquellen dieser Studie, Fehlerdatenbanken und Softwarearchive, beschrieben.

In **Kapitel 4** wird der in dieser Arbeit zentrale Begriff *Maß* eingeführt. In diesem Kapitel wird auf den Prozess des Messens in der Softwaretechnik und die dabei zu erfüllenden Voraussetzungen eingegangen. Dabei sollte es deutlich werden, dass eine Definition von Maßen sich stark an den bei deren Einsatz verfolgten Zielen orientieren muss.

In **Kapitel 5** werden die statistischen Verfahren Korrelation und Parametervergleich beschrieben.

In **Kapitel 6** werden ausgehend von der Fragestellung der Arbeit Maße hergeleitet.

In **Kapitel 7** wird der Prozess der Beschaffung der Daten und deren Vorbereitung für die Analyse beschrieben.

In **Kapitel 8** werden die Hypothesen untersucht.

In **Kapitel 9** werden die Ergebnisse der im Rahmen dieser Arbeit durchgeführten Studie kritisch geprüft.

In **Kapitel 10** wird die Arbeit mit einer Zusammenfassung der gewonnenen Erkenntnisse und einer Ausführung über die offen gebliebenen Fragen in diesem Bereich abgeschlossen.

Kapitel 2

Gegenstand der Arbeit

Der Erfolg von Projekten hängt von vielen Faktoren ab. Unter anderem gehören die Qualifikation im Team, die Werkzeugunterstützung sowie die eingesetzten Methoden und die befolgten Prinzipien zu diesen Faktoren. Deshalb müssen die Methoden und Prinzipien für den Einsatz in der Entwicklung sorgfältig bestimmt werden. Um darüber entscheiden zu können, welche Methoden für ein Projekt am besten geeignet sind, müssen diese objektiv im Hinblick darauf bewertet werden, welche Auswirkung sie auf den Erfolg von Projekten haben. Der Erfolg eines Projekts wird unter anderem dadurch bestimmt, wie effizient der Entwicklungsprozess im Bezug auf den Ressourcenverbrauch ist und wie hoch die erreichte Qualität des entwickelten Systems ist. In dieser Arbeit werden die Prinzipien *Collective Code Ownership* und *Zyklenfreiheit* im Bezug darauf bewertet, wie fehlerbehaftet das unter deren Befolgung entwickelte System ist.

2.1 Eingrenzung des Fehlerbegriffs

Es muss stets geklärt werden, was in einer Studie unter einem Fehler verstanden wird, weil dieser Begriff unterschiedlich weit gefasst sein kann. So exstieren im englischen Sprachraum die Begriffe *fault*, *failure* und *defect*. Sie tauchen in der Literatur mit unterschiedlicher Bedeutung auf. Es ist jedoch meistens mit fault und failure der während der Benutzung festgestellte Fehler, und mit defect der durch Lesen im Quelltext gefundene Fehler gemeint. In dieser Arbeit ist mit einem Fehler stets der während der Benutzung festgestellte Fehler gemeint, wenn nicht explizit anders gesagt.

Die Art der Fehler ist dadurch bestimmt, welche Einträge in der Datenbank durch das Eclipse-Team als Fehler erkannt werden. Bei den Einträgen in die Fehlerdatenbank handelt es sich um im Prozess der Benutzung festgestellte Fehler. Die Einträge werden von den Entwicklern des Teams im Hinblick darauf überprüft, ob die beschriebene Fehlersituation tatsächlich der Vorstellung des Teams über Fehler entspricht. Diese Arbeit orientiert sich an dieser Vorstellung, ohne diese explizit zu kennen. Es wird also angenommen, dass diese Vorstellung

gut fundiert ist.

Darüber hinaus werden in dieser Untersuchung nur Fehler betrachtet, die auch tatsächlich behoben wurden, da andernfalls der Fehler nicht einer im Quelltext vorzunehmenden Änderung, und somit keiner konkreten Entwurfseinheit zugeordnet werden kann.

Je nach dem, ob ein Fehler im Verlauf der Qualitätssicherung vor der Auslieferung an den Kunden oder erst nach der Auslieferung festgestellt wurde, wird dieser in die Kategorie der Pre-Release-Fehler oder die der Post-Release-Fehler eingeordnet. Es muss entschieden werden, welche der beiden Kategorien betrachtet werden sollen. Es gibt darüber hinaus noch die Möglichkeit, beide Kategorien zu betrachten.

Die Anzahl der Pre-Release-Fehler ist ein verbreitetes Maß für die Qualität der Entwurfseinheiten. Es wird beispielsweise in [Basili et al., 1996] und [Ostrand et al., 2005] verwendet. Das war einer der Aspekte, die Fenton in seinen Arbeiten [Fenton and Neil, 1999], [Fenton and Neil, 2000] und [Fenton and Pfleeger, 1998] kritisiert hat. Da die Anzahl der Pre-Release-Fehler in einem hohen Maße von der Qualität der Tests abhängt, ist eine Interpretation der Ergebnisse schwierig, wenn Studien sich an der Anzahl der Pre-Release-Fehler orientieren.

Wenn man nur die Anzahl der Post-Release-Fehler betrachtet, muss ebenfalls bedacht werden, dass einige Fehler während der Testphase gefunden werden. Dadurch liegt wieder eine Abhängigkeit von der Qualität der Tests vor.

In dieser Studie wurde vereinfachend angenommen, dass die während der Testphase festgestellten Fehler auch die äußere Softwarequalität beeinträchtigt hätten, da eine Identifizierung der Fehler als unrelevant für die Benutzung an Hand der vorliegenden Daten nicht möglich war. Deshalb wurden sowohl die Post-Release- als auch die Pre-Release-Fehler betrachtet.

2.2 Definition der Fehleranfälligkeit

Die Prinzipien Collective Code Ownership und Zyklenfreiheit sollen im Hinblick darauf bewertet werden, wie deren Befolgung sich auf das Ausmaß des sich in Form von Fehlern äußernden Qualitätsmangels eines Systems auswirkt. Für diesen Zweck wird eine Größe benötigt, die einen Vergleich von zwei unterschiedlichen Systemen in Bezug auf diese Art von Qualitätsmängeln ermöglicht.

Um den Begriff der Fehleranfälligkeit zu definieren, wird im Folgenden der Fall betrachtet, dass eine Version X des Systems Y entwickelt werden soll. $K(X)$ sei dabei der minimale erforderliche Aufwand für die Erstellung von X unter Einsatz von bestimmten Technologien. Die erreichte Qualität von X hängt davon ab, wie sorgfältig bei der Entwicklung vorgegangen wird. Ein Qualitätsmangel einer Systemversion kommt unter anderem auch durch das Auftreten von Fehlern zum Vorschein. Die auftretenden Fehler sind unterschiedlich schwer zu beheben. Jeder Fehler wird also durch seinen Schweregrad charakterisiert im Hinblick darauf, wie aufwendig dessen Behebung ist. Der sich in Form von Fehlern äußernde Qualitätsmangel Q von X wird dann durch die Gesamtschwere

aller in X enthaltenen Fehler beschrieben werden, wobei mit der Gesamtschwere die über alle Fehler gebildete Summe der Schweregrade gemeint ist.

Wenn im Prozess der Entwicklung von X ein sich in Form von Fehlern äußernder Qualitätsmangel $Q(X)$ entstanden ist und $K(X)$ der minimale erforderliche Aufwand ist, dann ist die Fehleranfälligkeit $F(X)$ dieser Version definiert durch:

$$F(X) = \frac{Q(X)}{K(X)}$$

Verbal formuliert, ist die Fehleranfälligkeit einer Systemversion die Gesamtschwere seiner Fehler pro Einheit des minimalen erforderlichen Aufwandes. Diese Definition ist zunächst intuitiv und wird bei der Definition der Maße als Basis dienen.

2.3 Architektur

Viele softwaretechnische Prinzipien zielen darauf ab, die Architektur von Systemen zu verbessern. In [und H. Züllighoven, 2002], Seite 784, wird Architektur folgendermaßen definiert:

> "Im engeren Sinne wird unter einer Architektur die Aufteilung eines Softwaresystems in seine Komponenten (meist Module), deren Schnittstellen, die Prozesse und Abhängigkeiten zwischen ihnen, sowie die benötigten Ressourcen verstanden."
>
> C. Floyd und H. Züllighoven

Systeme mit einer sinnvollen Architektur sind einfach verständlich, testbar und wartbar. Dadurch werden die im Prozess der Entwicklung und Wartung eines Systems anfallenden Kosten gesenkt. Darüber hinaus wirkt sich eine sinnvolle Architektur des Systems positiv auf seine äußere Qualität aus. So entstehen im Entwicklungsprozess weniger Fehler. Es wird deshalb stets eine sinnvolle Architektur von Systemen angestrebt.

Bei einem Architekturentwurf werden also Systeme in Komponenten unterteilt. Viele Systeme sind so groß, dass eine Zerlegung auf einer einzigen Ebene nicht ausreicht. In diesem Fall erfolgt die Zerlegung auf unterschiedlichen Granularitätsebenen. So werden Systeme in Subsysteme, Subsysteme in Module und Module in Dateien untergliedert. Dateien können aus mehreren Klassen bestehen. Klassen können wiederum mehrere Methoden enthalten. Wenn Komponenten auf einer beliebigen Granularitätsebene gemeint sind, wird in dieser Arbeit von *Entwurfseinheiten* gesprochen.

Zwischen den Entwurfseinheiten eines Systems bestehen Abhängigkeiten verschiedener Art. Welche Abhängigkeiten möglich sind, hängt unter anderem von der verwendeten Programmiersprache und von dem der Entwicklung zu Grunde liegenden Programmierparadigma ab. Im Sinne dieser Arbeit liegt eine Abhängigkeit dann vor, wenn eine Entwurfseinheit Methoden einer weiteren Entwurfseinheit benutzt.

Es gibt eine Reihe von seit langem bekannten Prinzipien, die beim Architekturentwurf helfen. Eins davon ist die Zyklenfreiheit. Sie wird im nächsten Abschnitt beschrieben.

2.4 Zyklenfreiheit

Die Abhängigkeiten zwischen Entwurfseinheiten lassen sich in Form eines Graphs darstellen. Die einzelnen Entwurfseinheiten bilden dann die Knoten, während die Abhängigkeiten durch die Kanten dargestellt sind. Ein Teilgraph davon heißt *stark zusammenhängend*, wenn es für jedes Paar von Entwurfseinheiten a und b einen Pfad von a nach b gibt [Neumann, 2005]. Eine Menge von Entwurfseinheiten, die einen stark zusammenhängenden Teilgraph im System bildet, wird als ein *Zyklus* bezeichnet. Innerhalb eines Zyklus sind jeweils zwei Entwurfseinheiten durch eine Folge von Abhängigkeiten miteinander verbunden.

Abbildung 2.1: Zyklen in einem Graph

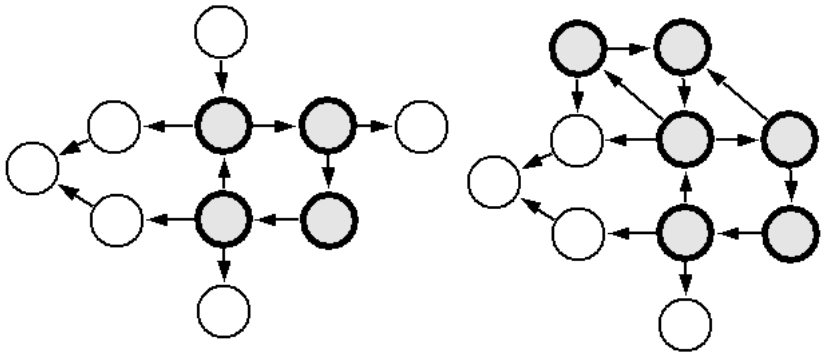


Abbildung 2.1 zeigt in der linken Hälfte einen Zyklus in einem Graph. In der rechten Hälfte der Abbildung 2.1 bildet die Menge der markierten Knoten ebenfalls einen Zyklus nach der oben gegebenen Definition. Erweitert man diese Definition dahingehend, dass man jeweils die kleinsten stark zusammenhängenden Teilgraphen als Zyklus bezeichnet, entspricht es besser dem intuitiven Verständnis. Im vorangegangenen Beispiel besitzt der Graph in diesem Fall drei Zyklen.

Ein Teilgraph heißt ein *echter Teilgraph*, wenn er aus weniger Knoten oder Kanten besteht als der ihn enthaltende Graph. Wenn ein stark zusammenhängender Teilgraph kein echter Teilgraph eines anderen stark zusammenhängenden Teilgraphs ist, heißt dieser ein maximaler stark zusammenhängender Teilgraph. Eine Gruppe von Entwurfseinheiten, die einen maximalen stark zusammenhängenden Teilgraph im System bildet, wird als eine *Zyklengruppe* bezeichnet. In Abbildung 2.1 rechts ist eine Zyklengruppe dargestellt.

Wenn zwischen den Entwurfseinheiten eines Systems ein Zyklus vorliegt, werden die Vorteile der Modularisierung abgeschwächt. So muss man, um die

Implementierung einer Entwurfseinheit zu verstehen, auch alle von ihr benutzten Entwurfseinheiten wenigstens teilweise verstehen. Im Falle einer Achitektur mit hierarchisch angeordneten Abhängigkeiten kann dies für die einzelnen Entwurfseinheiten nacheinander erfolgen.

Im Falle einer Änderung müssen möglicherweise weitere Entwurfseinheiten angepasst werden, die in der betroffenen Entwurfseinheit benutzt werden. Auch die Qualitätssicherung wird durch viele Abhängigkeiten zwischen den einzelnen Entwurfseinheiten erschwert, da sich Entwurfseinheiten mit vielen Abhängigkeiten schwieriger testen lassen.

Durch zyklische Abhängigkeiten im System wird also dessen Produktqualität verringert. Eine geringe Produktqualität hat jedoch zur Folge, dass die nachfolgenden Änderungen aufwendig sind und ein hohes Risiko für die Entstehung der Fehler haben. Deshalb wird beim Architekturentwurf angestrebt, das System frei von zyklischen Abhängigkeiten zu halten. Dieses Prinzip wird in dieser Arbeit als Zyklenfreiheit bezeichnet. Dieses Prinzip soll im Bezug darauf bewertet werden, inwieweit zyklische Abhängigkeiten zu einer Steigerung der Fehleranfälligkeit in Systemen führen.

Zyklische Abhängigkeiten können durch eine geeignete Umstrukturierung des Systems aufgehoben werden. Man spricht in diesem Fall auch vom Auflösen eines Zyklus oder einer ganzen Zyklengruppe. Es wird vermutet, dass schwer auflösbare Zyklengruppen eine stärkere Auswirkung auf die Fehleranfälligkeit haben als Zyklengruppen, die einfach aufzulösen sind. In dieser Arbeit wird der Begriff *Verflochtenheit einer Zyklengruppe* benutzt, wenn es darum geht, wie schwierig diese aufzulösen ist. Es wird also ebenfalls untersucht, ob die Verflochtenheit einer Zyklengruppe sich auch auf die Fehleranfälligkeit ihrer Entwurfseinheiten auswirkt.

Empirische Untersuchungen

Es wurden bereits einige Studien durchgeführt, die sich mit der Auswirkung der Abhängigkeiten auf die Fehleranfälligkeit auseinander gesetzt haben. So haben Zimmermann et al. [Zimmermann and Nagappan, 2006] eine ähnliche Studie durchgeführt, in der sie am System Microsoft Server 2003 die Eignung des Abhängigkeitsgraphen zur Prognose der Fehleranfälligkeit untersucht haben. In dieser haben sie folgende Beobachtungen gemacht:

- Untereinander zyklisch abhängige ausführbare Dateien[1] hatten im Durchschnitt doppelt so viele Fehler wie diese ohne zyklische Abhängigkeiten.
- Je größer die Zyklengruppe, desto mehr Fehler enthalten die in ihr enthaltenen ausführbaren Dateien

Diese Untersuchung hat im Gegensatz zu dieser Arbeit die Anzahl der Fehler und nicht die Fehleranfälligkeit als ein für die Qualität maßgebliches Kriterium angenommen. Deshalb kann aus den Ergebnissen dieser Studie nicht ohne weitere

[1] Der in der Studie verwendete Begriff *Binary* wird hier mit *ausführbare Datei* übersetzt.

Untersuchungen geschlussfolgert werden, dass eine Auflösung der Zyklengruppen zu einer Verringerung der Fehler im System insgesamt führen würde.

In einer weiteren Arbeit [Neumann, 2005], in der Abhängigkeiten untersucht wurden, wurde auch der Zusammenhang der definierten Maße mit der Größe von Entwurfseinheiten berücksichtigt. Robert Neumann hat im Rahmen seiner Diplomarbeit die nachfolgenden Hypothesen statistisch bestätigt:

- Dateien in Zyklengruppen ändern sich häufiger als Dateien außerhalb von Zyklengruppen.
- Dateien innerhalb von Zyklengruppen sind größer als diese außerhalb von Zyklengruppen.
- Dateien innerhalb von Zyklengruppen ändern sich auch relativ zu ihrer Größe häufiger als Dateien außerhalb von Zyklengruppen.
- Dateipaare ändern sich innerhalb von Zyklengruppen häufiger gemeinsam als außerhalb.
- Dateipaare mit und ohne Abhängigkeit ändern sich innerhalb von Zyklengruppen gleich häufig gemeinsam.
- Dateipaare ohne syntaktische Abhängigkeit ändern sich innerhalb von Zyklengruppen häufiger gemeinsam als außerhalb.

In einer Untersuchung hat Robert Neumann die Änderungshäufigkeit als einen Indikator für die Fehleranfälligkeit verwendet. Diese Arbeit geht über diese Diplomarbeit hinaus, indem sie zum einen ein Maß für die Fehleranfälligkeit verwendet, welches auf den Daten der Fehlerdatenbank aufbaut. Zum anderen werden auch die Größe und Verflochtenheit der Zyklengruppen untersucht.

2.5 Collective Code Ownership

Collective Code Ownership[2] ist eins der Konzepte im Rahmen des von Kent Beck Extreme-Programming-Ansatzes [Beck and Andres, 2004].

Extreme Programming ist eine der agilen Softwareentwicklungsmethoden. Bei dem Vorgehen nach Extreme Programming nähert sich das System in vielen wiederholten Schritten unter Verwendung von Rückkopplungen mit dem Kunden den gestellten Anforderungen an. Dieses Vorgehen soll einigen Problemen vorbeugen, die bei der Verwendung der klassischen Entwicklungsmethoden entstehen. Beispielsweise soll das Risiko von Terminverzögerungen und Abweichungen von den Anforderungen der Kunden verringert werden. Es soll die Fähigkeit erhöht werden, flexibel auf Änderungen der Anforderungen und anderer Bedingungen bei der Entwicklung zu reagieren. Die Kosten für eine Änderung in einer späteren Phase der Entwicklung sollen im Vergleich zu den klassischen

[2] Dieses Konzept wurde später in Shared Code umbenannt.

Entwicklungsmethoden geringer ausfallen. Um das zu erreichen, wird eine Reihe von Best Practices postuliert, unter anderem auch Collective Code Ownership.

Dieses Prinzip bezieht sich auf die Handhabung der Zuständigkeiten der Entwickler für unterschiedliche Teile des entwickelten Systems in Entwicklungsprojekten. Übersetzt aus dem Englischen heißt Collective Code Ownership kollektiver Quelltextbesitz. Im Sinne von XP besitzt ein Entwickler einen Quelltextabschnitt, wenn er das Recht hat, diesen zu ändern. Diese Bezeichnung kommt aus der traditionellen Vorgehensweise, bei der jeder Entwickler für bestimmte Teile des Systems Verantwortung trägt und andere Entwickler nicht das Recht haben, diese ohne Rücksprache mit dem Besitzer zu ändern. Kollektiver Quelltextbesitz soll zum Ausdruck bringen, dass, im Gegensatz zur traditionellen Vorgehensweise, das gesamte Team in gleichem Maße das Recht hat, die einzelnen Entwurfseinheiten zu ändern, und zwar jederzeit und ohne Rücksprache mit dem ursprünglichen Autor des Quelltextabschnitts.

Collective Code Ownership wird praktiziert, indem die Entwickler ihre Änderungsrechte tatsächlich wahrnehmen. Bei der Definition der Maße in Kapitel sechs wird der Begriff *praktiziertes Collective Code Ownership* verwendet. Er bringt zum Ausdruck, wie kollektiv der Quelltextbesitz für diese Entwurfseinheit wirklich ist. Praktiziertes Collective Code Ownership für eine bestimmte Version einer Entwurfseinheit gibt also an, wieviele verschiedene Entwickler ihre Änderungsrechte für diese wahrgenommen haben pro Einheit des für diese Version erforderlichen Aufwands.

Vorteile, die von der Anwendung des Prinzips versprochen werden, sind die folgenden:

Bessere Qualität des Quelltextes

Da das Expertentum nicht gern gesehen wird, müssen die Entwickler häufig Quelltext lesen und verstehen, den sie nicht geschrieben haben. Deshalb besteht ein hoher Anspruch an die Verständlichkeit des Quelltextes für ein System, was unter Collective Code Ownership entwickelt wird. Es ist eine weit verbreitete Annahme, dass diese Anforderung dazu führt, dass die Teammitglieder verstärkt auf die Qualität des Quelltextes achten.

Minimierung des Truck-Faktors

Der Truck-Faktor gibt an, mit welcher Wahrscheinlichkeit das Entwicklungsprojekt scheitert, wenn ein Teammitglied von einem Lastwagen überfahren wird. Dadurch, dass das Wissen über das System im Team weitgehend homogen ist, kann jeder Entwickler jede Aufgabe übernehmen. Auf diese Weise ist die Behinderung des Entwicklungsprozesses im Falle eines Personalausfalls minimal, und somit auch der Truck-Faktor.

Minimierung der Umständlichkeit

Bei der traditionellen Vorgehensweise muss stets mit dem Besitzer eines Quelltextabschnitts kommuniziert werden, wenn festgestellt wird, dass dieser geän-

dert werden soll. Martin Fowler kritisiert diese Vorgehensweise in [Fowler, 2006] wegen der Umständlichkeit und Verzögerungen. Er weist darauf hin, dass in den meistens Projekten Änderungen in anderen Teilen des Systems oft erforderlich sind und diese Vorgehensweise deshalb zu einer bedeutenden Behinderung im Entwicklungsprozess führt. Um diese Kritik nachvollziehen zu können, muss bedacht werden, dass Collective Code Ownership mit weiteren Best Practices von Extreme Programming zusammen praktiziert wird. So ist unter anderem Refactoring eine viel genutzte Best Practice. Typische Refactoring-Maßnahmen wie Umbenennen von öffentlichen Methoden können oft dazu führen, dass Änderungen über den Rahmen einer Entwurfseinheit hinaus gehen. So müssen im Falle der Umbenennung einer öffentlichen Methode in einer bestimmten Klasse alle Stellen außerhalb dieser Klasse geändert werden, an denen diese benutzt wird.

Die Umsetzung dieses Konzepts in der Praxis erscheint jedoch schwierig, was von den Autoren selbst in [Beck and Andres, 2004], [Beck and Fowler, 2000] und [Auer and Miller, 2002] thematisiert wird.

Collective Code Ownership bedeutet weniger Kontrolle und mehr Vertrauen in die Arbeit anderer. Deshalb müssen Teammitglieder sich bereits während der Änderung intensiv mit deren Folgen auseinandersetzen. Ein Entwickler kann sich nicht darauf verlassen, dass anderen Entwicklern, die mit dem geänderten Quelltextabschnitt mehr vertraut sind als er selbst, seine Denkfehler auffallen. Er muss die volle Verantwortung für seine Änderungen tragen.

Da man bei großen Systemen nicht alle Einzelheiten der Implementierung im Kopf behalten kann, muss man sich häufig zur Durchführung einer Änderung erst in den betroffenen Teil des Systems einarbeiten. So müssen die Entwickler viele Teile des Systems lesen und verstehen. Eine sorgfältige Einarbeitung ist mit einem zusätzlichen zeitlichen Aufwand verbunden und setzt stark ausgeprägtes Verantwortungsgefühl bei den Teammitgliedern voraus. Wenn dafür jedoch nicht mehr Zeit eingeplant ist als bei der traditionellen Vorgehensweise, ist der Zeitdruck höher. Zeitdruck führt häufig dazu, dass die Einarbeitung nicht im erforderlichen Maße erfolgt und deshalb nicht durchdachte Änderungen vorgenommen werden, die zur Entstehung von Fehlern führen.

Dieses Problem gewinnt in großen Projekten an Schärfe. Zur Entwicklung großer Systeme in akzeptabler Zeit sind große Entwicklerteams erforderlich. Je größer das System und das Team, desto schwieriger wird eine sinnvolle Umsetzung dieses Konzepts. Im Rahmen dieser Fallstudie soll geprüft werden, wie schwerwiegend dieses Problem in einem konkreten Projekt ist.

2.6 Fragestellung der Arbeit

In dieser Arbeit sollen die zwei oben beschriebenen Prinzipien in Bezug auf die Fehleranfälligkeit des gesamten Systems bewertet werden. Die zu untersuchenden Prinzipien werden in den einzelnen Entwurfseinheiten unterschiedlich stark befolgt. So wurde Collective Code Ownership in verschiedenen Teilen des

Systems unterschiedlich intensiv praktiziert. Das gleiche gilt auch für die Zyklenfreiheit. Deshalb werden in dieser Arbeit die Prinzipien bewertet, indem für einzelne Teile des Systems untersucht wird, ob ein Unterschied im Ausmaß der Befolgung von einem Unterschied in der Fehleranfälligkeit begleitet ist. In diesem Abschnitt werden die Hypothesen zunächst unter Verwendung intuitiver Begriffe formuliert. Die Beschreibung der tatsächlich erforderlichen Daten folgt erst nach der Herleitung der Maße.

Im Bezug auf das Collective Code Ownership soll geprüft werden, ob man die in Abschnitt 2.5 beschriebene negative Auswirkung des praktizierten Collective Code Ownership in einem konkreten Projekt nachweisen kann. Um das zu überprüfen, muss das Vorliegen der beschriebenen Situation anhand von messbaren Merkmalen zu erkennen sein. Zum einen ist die Situation dadurch charakterisiert, dass durch Änderungen Fehler entstehen. Deshalb muss eine Änderung der Fehleranfälligkeit des Systems vorliegen. Zum anderen muss die Entstehung der Fehler in Verbindung mit einem hohen Ausmaß an praktiziertem Collective Code Ownership stehen. Es wird deshalb überprüft, ob ein hohes Ausmaß an praktiziertem Collective Code Ownership zu einer höheren Fehleranfälligkeit in einem System führt. Daraus ergibt sich die folgende Hypothese:

Hypothese 1: Entwurfseinheiten mit hohem Ausmaß an praktiziertem Collective Code Ownership weisen eine Tendenz zu einer höheren Fehleranfälligkeit auf als die Entwurfseinheiten mit einem geringen Ausmaß an praktiziertem Collective Code Ownership

Bezüglich der zyklischen Abhängigkeiten soll die Vermutung überprüft werden, dass diese ebenfalls die Fehleranfälligkeit von Systemen beeinflussen. Darüber hinaus wäre eine Untersuchung der Frage interessant, ob bestimmte Eigenschaften der Zyklengruppen, konkret deren Größe und Verflochtenheit, auch dazu führen, dass die Fehleranfälligkeit eines Systems steigt. Ausgehend von diesen Überlegungen werden die nachfolgenden Hypothesen aufgestellt:

Hypothese 2: Entwurfseinheiten in Zyklen weisen eine Tendenz zu einer höheren Fehleranfälligkeit auf als die Entwurfseinheiten außerhalb von Zyklen

Hypothese 3: Entwurfseinheiten in größeren Zyklengruppen weisen eine Tendenz zu einer höheren Fehleranfälligkeit auf als die Entwurfseinheiten in kleineren Zyklengruppen

Hypothese 4: Entwurfseinheiten in stärker verflochtenen Zyklengruppen weisen eine Tendenz zu einer höheren Fehleranfälligkeit auf als die Entwurfseinheiten in weniger verflochtenen Zyklengruppen

Kapitel 3

Das untersuchte Projekt Eclipse

Für die Untersuchung der Prinzipien Collective Code Ownership und Zyklenfreiheit von Systemen wurden die Daten des OpenSource-Projekts *Eclipse* verwendet. Eclipse ist eine erweiterbare Entwicklungsumgebung, die in zahlreichen Projekten und unterschiedlichen Kontexten eingesetzt wird. Die bekannteste Verwendung von Eclipse ist deren Verwendung als IDE für Entwicklung in Java. Wegen seiner Erweiterbarkeit wird Eclipse jedoch zur Entwicklung in vielen unterschiedlichen Programmiersprachen eingesetzt. Es wird auch zunehmend zur Entwicklung von Rich-Client-Applikationen auf Basis der Eclipse Rich Client Platform (RCP) verwendet.

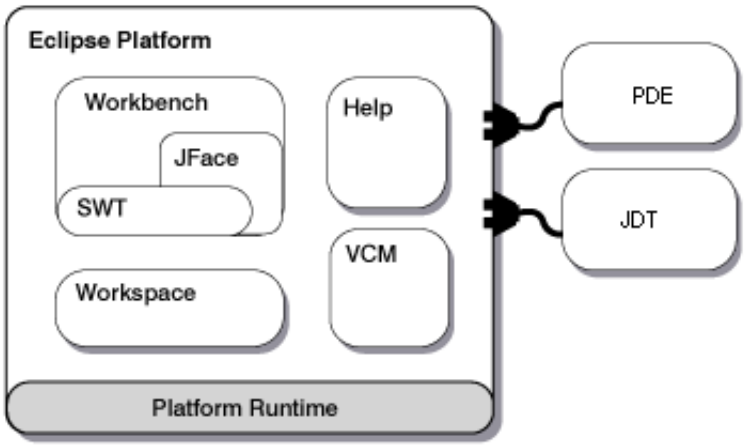
Eclipse stellt Schnittstellen und Funktionen für die Integration von Plugins bereit. Es gibt eine sehr große Anzahl von Plugins, die diese Umgebung für verschiedenste Zwecke einsetzbar macht. So kann Eclipse auch als allgemeiner Client Container für verschiedenste Applikationen dienen. Sowohl Eclipse als auch die Plugins sind vollständig in Java implementiert. Diese Studie ist ausschließlich auf das Eclipse SDK beschränkt. Die Struktur von Eclipse SDK ist in der Abbildung[1] *Struktur von Eclipse* dargestellt. Dieses enthält neben der eigentlichen Plattform zwei Plugins: JDT und PDE.

Die Wahl von Eclipse ist vor allem begründet durch die Notwendigkeit der freien Verfügbarkeit sowohl des Quelltextes als auch einer Fehlerdatenbank. Diese für eine Analyse notwendigen Voraussetzungen werden nur von wenigen OpenSource-Systemen erfüllt, da deren Entwickler zur Beseitigung der Fehler nicht verpflichtet sind und deshalb meistens nicht gründlich genug dieser Beschäftigung nachgehen. Im Fall von Eclipse sind diese Voraussetzungen gegeben, da es einen industriellen Ursprung hat und erst nachträglich als ein OpenSource-Projekt freigegeben wurde.

Zum anderen wird Eclipse in unterschiedlichen Organisationen eingesetzt

[1] Diese Abbildung wurde modifiziert übernommen aus [Object Technology International, 2003].

Abbildung 3.1: Struktur von Eclipse



und intensiv genutzt. Eine intensive Benutzung des Systems ist von Vorteil, da in diesem Fall das System von vielen Benutzern und auf verschiedene Weisen bedient wird. Das beschleunigt das Auffinden und Melden der Fehler und erhöht somit die Vollständigkeit der Fehlerdatenbank. Ein weiterer Grund für die Wahl von Eclipse ist die Implementierung in Java. Sie ermöglicht eine einfache Strukturanalyse, da es dafür bereits einige frei verfügbare Werkzeuge gibt. Darüber hinaus ist Eclipse von seiner Größe her typisch für Projekte, die in der Praxis ablaufen (siehe Abbildung *Übersicht über die Kenngrößen von Eclipse*).

Eclipse hat bereits 10 Release-Versionen, was zu einer höheren Genauigkeit der Ergebnisse beiträgt. Das ist insbesondere vorteilhaft, wenn ein Vorhersagemodell durch Regression aus den vorliegenden Daten gebildet werden soll. Dann können nämlich die Daten einiger Release-Versionen als Testdaten benutzt werden, um die Güte des unter Verwendung der Daten übriger Release-Versionen geschätzten Regressionsmodells zu messen. Eine Übersicht über alle Release-Versionen von Eclipse ist in der Tabelle 3.1[2] gegeben. In dieser Studie werden die Daten der letzteren der aufgelisteten Versionen, **3.2.1**, benutzt, um bestimmte Strukturmerkmale des Quelltextes, beispielsweise Anzahl der Zeichen, zu untersuchen.

3.1 Daten des Versionsverwaltungssystems

Bei der Entwicklung von Eclipse wird das Versionsverwaltungssystem *CVS* (Concurrent Version System) zur Verwaltung der Versionsdaten eingesetzt. CVS wird zur Verwaltung der meisten Softwareprojekte bei *Sourceforge* genutzt und wird wegen einer einfachen Benutzung mit der Entwicklungsumgebung Eclip-

[2]Quelle: [Open Source Competence Group (OSCG), 2006]

Abbildung 3.2: Übersicht über die Kenngrößen von Eclipse

Metrik	Metrikwert
Attribute	69211
Aufrufe	547947
Dateien	13586
Klassen	23442
Klassen in Zyklen	5764
Kommentarblöcke	220538
Kommentarzeilen	924812
LOC	2869566
Methoden	176137
Packages	978
Packages in Zyklen	667
Schnittstellen	2424
Vererbungen	20286

Tabelle 3.1: Release-Versionen von Eclipse und deren Zeitpunkte

Release-Version	Release-Datum
3.2.1	30. September 2006
3.2.0	1. Juli 2006
3.1.2	26. Januar 2006
3.1.1	29. September 2005
3.1.0	28. Juni 2005
3.0.2	31. März 2005
3.0.1	18. September 2004
3.0.0	27. Juni 2004
2.1.3	17. März 2003
2.1.2	November 2003

se in vielen Entwicklungsprojekten bevorzugt. Eine ausführliche Beschreibung der Funktionalität sowie der gespeicherten Daten findet man beispielsweise in [Vesperman, 2003]. Im Rahmen dieser Studien wurden die folgenden Daten über die einzelnen Änderungen als nützlich vermutet:

Autor

Der Autor ist die Person, welche die Änderung durchgeführt hat.

Zeit

In diesem Datenfeld wird das Datum und die Uhrzeit des Commit-Vorgangs abgelegt.

Kommentar

Der Commit-Kommentar ist eine Beschreibung der Transaktion durch den Autor, in der der Inhalt der durchgeführten Änderungen erklärt wird.

Änderungsumfang

Zu jeder Änderung speichert CVS die Anzahl gelöschter und hinzugefügter Textzeilen. Dabei wird eine Zeile, in der Text geändert wird, durch ein Löschen der alten und Hinzufügen der neuen Zeilen repräsentiert.

Total Revisions

Total Revisions ist die gesamte Anzahl der an dieser Datei vorgenommenen Änderungen.

3.2 Daten der Fehlerdatenbank

Zur Verwaltung der Fehlerdaten wurde in Eclipse das Werkzeug *Bugzilla* eingesetzt. Die in diesem System üblicherweise gesammelten Daten sind beispielsweise in [Barnes, 2004] ausführlich beschrieben. In diesem Fall waren die folgenden Daten verfügbar:

Identifikationsnummer (ID)

Diese ID ist eine den Fehlereintrag eindeutig identifizierende Nummer und hat in dieser Studie eine hohe Bedeutung, da sie bei der Zuordnung einer Änderung zu einem Fehlereintrag benutzt wird.

Priorität (priority)

Diese Angabe ist wichtig, um zu beurteilen, wie schwerwiegend der betroffene Fehler ist.

Eröffnungsdatum (open date)

Dieses gibt an, wann der Fehler in die Datenbank eingetragen wurde.

Antragsteller (reporter/author)

Der Antragsteller ist die Person, die den Fehler entdeckt hat und in die Datenbank eingetragen hat. Für Eclipse gibt es zwei Spalten mit dieser Bezeichnung. Eine davon enthält die E-Mail-Adresse des Antragstellers, die andere seinen Namen.

Beschreibung (summary)

Dabei handelt es sich um eine kurze textuelle Beschreibung des Fehlers.

Status (status)

Status ist die Information über den Stand der Bearbeitung. Sobald der Fehler als beseitigt gilt, kann man dies an dem Wert des Status erkennen. Im Fall von Eclipse sind die Werte UNCONFIRMED, NEW, ASSIGNED,RESOLVED, REOPENED, VERIFIED, CLOSED zulässig.

Wenn ein Fehler zum ersten Mal in die Datenbank eingetragen wird, dann wird der Status auf UNCONFIRMED gesetzt. Sobald ein Entwickler mit den entsprechenden Rechten diesen Eintrag bestätigt hat, wird der Status auf NEW gesetzt. Der Fehler wird dann gleich einem der Entwickler zur Bearbeitung vorgeschlagen und auf seine Bearbeitungsliste gesetzt. Er geht seine Liste durch und kann alle neu eingetragenen Einträge akzeptieren oder ablehnen. Sobald einer der Entwickler den Fehler akzeptiert hat, wird der Status des Fehlers auf ASSIGNED gesetzt. Wenn der zuständige Entwickler den Fehler bearbeitet hat, wird der Status auf RESOLVED gesetzt. Danach muss von dem Autor des Fehlereintrags bestätigt werden, dass der Fehler tatsächlich behoben wurde. Wenn das der Fall ist, wird der Status auf VERIFIED gesetzt, andernfalls auf REOPENED. Wenn eine neue Version von Eclipse zur Verfügung gestellt wird, werden alle Fehler mit dem Status VERIFIED geschlossen und der Status auf CLOSED gesetzt. Wenn zu einem späteren Zeitpunkt festgestellt wird, dass der Fehler wieder auftritt, kann der Status wieder auf REOPENED gesetzt werden.

Resolution (resolution)

Die Resolution eines Fehler wird erst nach seiner Abarbeitung durch die Entwickler festgelegt. Sie dient dazu, die Fehler in Kategorien einzuteilen. Es sind folgende Kategorien definiert:

FIXED Der Fehler wurde behoben und die Problemlösung wurde getestet.

INVALID Das beschriebene Problem ist kein Fehler in dem definierten Sinne.

WONTFIX Der beschriebene Fehler wird nie behoben.

DUPLICATE Der beschriebene Fehler ist bereits in die Datenban eingetragen worden.

WORKSFORME Der beschriebene Fehler konnte nicht reproduziert werden.

MOVED Dieser Fehler wurde in eine falsche Datenbank eingetragen und wurde deshalb an die richtige Stelle verschoben.

Kapitel 4

Grundlagen der Messungen

Eine Aussage, ob ein System ein bestimmtes Merkmal besitzt, beispielsweise gut strukturiert ist, ist zu ungenau und subjektiv. Deshalb stellt diese keine geeignete Basis dar, um zwei Systeme miteinander zu vergleichen oder um zu überprüfen, ob die gesetzten Ziele bei deren Entwicklung erreicht wurden. Um Systeme und Entwicklungsprozesse systematisch vergleichen und verbessern zu können, muss man in der Lage sein, objektive Aussagen über den Grad der Merkmalausprägung für diese zu treffen.

Eine objektive Beschreibung der Merkmale wird durch Messungen gewonnen. Bei einer Messung wird nach klar definierten Regeln die spezifische Ausprägung eines Merkmals festgestellt. Den Untersuchungsgegenständen der realen Welt werden dabei Zahlen oder Symbole zugeordnet.

Ausgangspunkt der Messungen ist eine praktische Fragestellung, für die eine Quantifizierung bestimmter Größen erforderlich ist. Da eine unmittelbare Quantifizierung ohne eine Formalisierung wegen begrenzter Wahrnehmungskapazitäten nur bedingt möglich ist, muss zunächst ein Modell des Untersuchungsgegenstandes gebildet werden, das die für die Fragestellung wesentlichen Merkmale hervorhebt. Ausgehend von dem Modell werden Maße definiert. Im Folgenden wird der Begriff *Maß* und das Vorgehen bei der Definition von Maßen beschrieben.

4.1 Maße

Wir lernen etwas über die reale Welt, indem wir deren Gegenstände beobachten und miteinander vergleichen. Durch einen Vergleich bringen wir jeweils zwei Gegenstände der realen Welt in eine Relation zueinander. Diese Relation wir als *empirische Relation* bezeichnet. Empirische Relationen kann man sich als Abbildungen aus der realen Welt in die formale, mathematische Welt vorstellen. Diese Abbildung ist nur dann sinnvoll, wenn sie die folgende Bedingung erfüllt:

> Wenn zwei Gegenstände in der realen Welt in einer Relation stehen, müssen auch die zugewiesenen Messwerte in einer analogen Relation

stehen.

Diese Bedingung wird in [Fenton and Pfleeger, 1998] als Repräsentationsbedingung (representation condition of measurement) bezeichnet, da sie erfüllt sein muss, damit die Relationen zwischen den Zahlen oder Symbolen repräsentativ für die Relationen zwischen den Merkmalen von Gegenständen sind. Beispielsweise, wenn eine Person A größer ist als eine Person B, muss auch der A zugeordnete Wert größer sein als der Wert von B. Andernfalls ist diese Abbildung nicht sinnvoll. Bei einem Maß handelt es sich um eine Abbildung, die die Repräsentationsbedingung erfüllt. Formal wird der Begriff *Maß* in [Fenton and Pfleeger, 1998], Seite 28, wie folgt definiert:

> "Formally, we define **measurement** as a mapping from the empirical world to the formal, relational world. Consequently, a **measure** is the number or symbol assigned to an entity by this mapping in order to characterize an attribute."
>
> Fenton und Pfleeger

Durch Messungen an einem Untersuchungsgegenstand wird eine Merkmalsausprägung präzisiert. Dies erfolgt auf Grundlage einer bestimten Skala. Eine Skala bestimmt die Stufe der Messbarkeit eines Merkmals. Sie bestimmt, welche Operationen sich auf die Menge der Messwerte anwenden lassen. Wenn Rechenoperationen für die Messwerte einer Skala zulässig sind, wird diese als quantitativ bezeichnet. Andernfalls ist die qualitativ. Skalen können nach bestimmten Eigenschaften zu Skalentypen zusammengefasst werden. Die zu einem Skalentyp zusammengefassten Skalen heißen akzeptable Skalen für diesen Skalentyp.

4.2 Unterscheidung von Maßen

Da bereits eine große Anzahl von Maßen definiert wurde, ist es schwer, sich einen Überblick zu verschaffen. Eine Klassifizierung soll dabei helfen. Man kann Maße nach ihrer Automatisierbarkeit, Messbarkeit der Merkmale, der Art des Messgegenstands und der Art der Skala unterscheiden.

Maße sind automatisierbar, wenn die erforderlichen Daten automatisch gesammelt und ausgewertet werden können. Andernfalls sind sie nicht oder nur teil- automatisierbar. So müssen die Fehler zunächst von Personen in eine Datenbank eingetragen werden und anschließend manuell überprüft werden. Erst dann steht die Anzahl der Fehler als ein mögliches Maß für das Fehlverhalten des Systems zur Verfügung.

Wenn ein Maß durch einen einzelnen Messvorgang ermittelt werden kann, wird es als direkt bezeichnet. So kann beispielsweise das Maß *Anzahl der Änderungen* direkt einem Versionsverwaltungssystem entnommen werden. Wenn das nicht der Fall ist, muss ein Maß aus anderen Maßen berechnet werden. Dafür sind meistens mehrere Messungen erforderlich. So wird die Fehlerdichte aus den direkten Maßen *Anzahl der Fehler* und *Anzahl der Codezeilen* berechnet.

Nach der Art des Gegenstands wird in Prozessmaße, Ressourcenmaße, interne sowie externe Produktmaße unterschieden. Während Prozessmaße die Merkmale von Prozessen wie die Dauer eines bestimmten Schritts charakterisieren, beschreiben die Ressourcenmaße die Eigenschaften der Ressourcen, beispielsweise die Qualifikation der Entwickler. Externe und interne Produktmaße sind Ausdruck externer beziehungsweise interner Merkmale des Softwareprodukts. So sind Effizienz und Zuverlässigkeit eines Systems externe Produktmaße, während die Komplexität ein internes Produktmaß ist.

Ein Maß heisst qualitativ, wenn es nicht sinnvoll ist, mit seinen Messwerten zu rechnen. Es sind nur Vergleichsoperationen zulässig. Wenn für die Messwerte eines Maßes auch Rechenoperationen zulässig sind, wird das Maß als quantitativ bezeichnet.

4.3 Definition von Maßen

Die Definition von Maßen beginnt mit der Festlegung des Untersuchungsziels. Nach der Festlegung der Ziele wird ein Modell des Untersuchungsgegenstandes gebildet, welches ausschließlich die wesentlichen Merkmale beinhaltet. Das Modell bestimmt unter anderem den Definitionsbereich der durch ein Maß festgelegten Abbildung. Die Qualität der Übertragung von Ergebnissen der formalen Welt in die Realität hängt von der Qualität des Modells ab. Deshalb spielt dieser Schritt eine große Rolle bei der Definition der Maße.

Auf dem vorliegenden Modell können hinsichtlich jedes zu untersuchenden Merkmals empirische Relationen identifiziert werden. Es soll dabei sichergestellt werden, dass die auf dem Modell ermittelten Relationen in der Realität interpretierbar sind. Die festgestellten Relationen sind für die Bestimmung eines Skalentyps für jedes Merkmal entscheidend. Anschließend können Maße formuliert werden. Dazu werden eine konkrete Skala sowie Vorschriften über die genaue Erhebung von Daten anhand des Modells festgelegt. Im folgenden werden die einzelnen Schritte bei der Definition von Maßen genauer beschrieben.

4.3.1 Festlegung des Untersuchungsziels

Hilfestellung bei der Beschreibung des Untersuchungsziels gibt die GQM-Methode [Basili and Weiss, 1984] mit ihrem GQM-Template. Das Template gibt vor, dass unter anderen die folgenden Fragen beantwortet werden müssen:

Gegenstand: Was ist der Untersuchungsgegenstand?

Zweck: Welchem Zweck dient die Untersuchung?

Fokus: Welches Merkmal des Gegenstands soll untersucht werden?

Die Beantwortung dieser Fragen ist erforderlich, um festzustellen, welche Merkmale betrachtet werden sollen. Sie wurden im Rahmen dieser Studie in Kapitel zwei beantwortet.

4.3.2 Ableitung eines Modells

Nach der Bestimmung der Hauptziele werden anhand des Untersuchungsgegenstandes konkret messbare Merkmale identifiziert, welche einen Einfluss auf den betrachteten Ausschnitt der Realität haben. Dies erfolgt in einem schrittweisen Verfeinerungsprozess, wobei die verfolgten Ziele immer weiter in Teilziele untergliedert werden.

Ein Modell abstrahiert von überflüssigen Details und stellt einen Gegenstand aus einer bestimmten Perspektive dar. So schränkt ein Kostenmodell den Blickwinkel auf Kennzahlen wie Produktumfang oder Produktivität und erlaubt es, Vorhersagen bezüglich der Kostenstruktur eines geplanten Projektes zu machen. Ein Modell dient dazu, bestimmte Art von Beziehungen zwischen den Gegenständen der realen Welt besser zu verstehen.

Die Modellbildung erfolgt in der Praxis häufig implizit und ohne die notwendige Sorgfalt. Während beim allgemein üblichen Messen die Modelle häufig klar sind, ist dies bei Software nur selten der Fall. Es gibt zu selten eine allgemeingültige Vorstellung über die Bedeutung eines Begriffes. So kann beispielsweise der Begriff Komplexität sehr unterschiedlich interpretiert werden. Die konkrete Beurteilung, ob ein Programm hinsichtlich einer solchen Interpretation komplexer als ein anderes sei, ist stark vom jeweiligen Betrachter abhängig. Es ist deshalb oft schwer, Maße zu definieren, die allgemein als eine geeignete Charakterisierung eines Merkmals angesehen werden.

Es wird angestrebt, dass jedes einer Messung zu Grunde liegende Modell folgende Kriterien erfüllt [Bennicke and Rust, 2005]:

Eindeutigkeit: Jedes betrachtete und nicht betrachtete Merkmal muss erläutert werden und es muss festgelegt werden, welche Rolle es in der Abstraktion einnimmt. Für das Beispiel der Beurteilung der Zyklenfreiheit muss beispielsweise geklärt werden, wann zwei Entwurfseinheiten als verbunden betrachtet werden und wann nicht.

Vollständigkeit: Alle möglichen Ausprägungen eines relevanten Merkmals müssen bedacht werden. Hierbei muss beispielsweise geklärt werden, ob alle relevanten Einflussfaktoren auf eine Größe erfasst wurden.

Konsistenz: Das Vorkommen eines Merkmals muss für verschiedene Entitäten konsistent behandelt werden. Eindeutigkeit und Vollständigkeit sollten Konsistenz zur Folge haben.

Verständlichkeit: Da die Modellbildung der erste Schritt zur Maßdefinition ist, muss das Modell leicht verständlich sein. Dazu ist ein Kompromiss zwischen exakten, aber mit zu vielen Details belasteten formalen Definitionen und nicht formalen Beschreibungen zu finden.

Die Einhaltung dieser Kriterien ist zwar erstrebenswert, es ist jedoch nicht möglich, zu überprüfen, ob ein Modell sie tatsächlich erfüllt. Der Grund dafür ist derselbe wie auch dafür, dass es keinen vollständigen Beweis der Korrektheit von Programmen geben kann. Die Kriterien sollen einen Hinweis darauf geben, was bei der Modellbildung bedacht werden soll.

4.3.3 Bestimmung des Skalentyps

Die Bestimmung des geeigneten Skalentyps hat für den Messprozess eine hohe Bedeutung, da der Skalentyp festlegt, welche Analysen anhand von Messwerten durchführbar sind. In der Messtheorie unterscheidet man eine Hierarchie von Skalentypen, die in der Tabelle 4.1 mit den entsprechenden Relationen und Operationen dargestellt ist[1].

Tabelle 4.1: Skalentypen

Niveau	**Relationen**	**Beispiele geeigneter statistischer Operationen**	**Geeignete statistische Tests**
nominal	Gleichheit	Modus Frequenz	nicht-parametrische Tests
ordinal	Gleichheit total geordnet	Median Perzentil	
Intervall	Gleichheit total geordnet Verhältnisse der Intervalle	arithmetisches Mittel Standardabweichung	
Verhältnis	Verhältnis von zwei Skalenwerten	geometrisches Mittel	parametrische Tests

Während es bei einer Nominalskala nur darum geht, die Gegenstände in unterscheidbare Mengen einzuordnen, wird bei einer Ordinalskala der auf eine geordnete Menge abgebildet. So kann beispielsweise jeder Entwurfseinheit eine von zehn Stufen der Fehleranfälligkeit zugeordnet werden. In dieser Skala ist nur die Reihenfolge von Bedeutung. Da die Abstände zwischen den einzelnen Stufen nicht definiert sind, kann weder Mittelwert noch die Standardabweichung berechnet werden. Es kann jedoch der Median bestimmt werden. Dieser wird von dem Wert an der mittleren Stelle in der geordneten Menge repräsentiert.

In Gegensatz zur Ordinalskala sind in der Intervallskala auch die Differenzen zwischen den Intervallen definiert. Deshalb kann für die Werte dieses Skalentyps ein Mittelwert gebildet werden. Ebenfalls sind Addition und Subtraktion für diesen Skalentyp zulässig.

Für die Verhältnisskala ist darüber hinaus ein natürlicher Nullpunkt definiert, was eine Voraussetzung für die Multiplikation ist. Deshalb hat das Verhältnis zwischen den Werten dieses Skalentyps eine Bedeutung. Es wäre damit eine Aussage "A ist doppelt so hoch wie B" zulässig.

In der Literatur finden sich oft der Begriff Absolutskala. Die Absolutskala ist eine Verhältnisskala, bei der die Werte auf natürliche Zahlen beschränkt sind. Es handelt sich dabei also um Werte, die durch Zählen bestimmt werden. So wird beispielsweise die Anzahl der Quelltextzeilen einer Entwurfseinheit durch Zählen ermittelt. Dividieren und Logarithmieren sind innerhalb dieser Skala nur beschränkt möglich. Deshalb kann für die Werte dieses Skalentyps, wie auch im

[1] Diese Tabelle wurde aus [Bennicke and Rust, 2005] entnommen.

Fall einer Ordinalskala, kein Mittelwert berechnet werden. Aus diesem Grund werden Absolutskalen meistens zu Verhältnisskalen erweitert. Der auf diese Weise berechnete Mittelwert ist zwar nicht immer eine ganze Zahl, beispielsweise 2,3 Fehler pro Datei, dieser Wert hat jedoch eine pragmatische Bedeutung.

Zur Bestimmung des Skalentyps müssen die Eigenschaften des untersuchten Merkmals mit den Eigenschaften der verschiedenen Skalentypen verglichen werden. Oft sind für ein Maß mehrere Skalentypen möglich. So könnte man die Anzahl der Fehler auch auf einer Ordinal- oder Intervallskala darstellen. Damit die Einschränkung bezüglich der zulässigen Operationen so gering wie möglich ist, muss immer die stärkste mögliche Skala zur Definition eines Maßes verwendet werden. Den meisten Maßen in der Softwaretechnik wird eine Absolutskala zu Grunde gelegt, und in einigen Fällen wird diese zu einer Verhältnisskala erweitert.

Da die empirischen Relationen von den Ansichten der Menschen abhängig sind, ist die Bestimmung des geeigneten Skalentyps oft schwierig. In
[Fenton and Pfleeger, 1998] wird dazu das folgende Beispiel gegeben:

Ein plausibles Verständnis von "Programmlänge" ist die Länge einer Beschreibung des Programms. Bereits bei dieser Fokussierung sind mehrere Auffassungen möglich: Anzahl von Quelltextzeilen oder Anzahl von Klassen in einem objektorientierten System? Es lässt sich weiter argumentieren, dass die Länge der Beschreibung kein unmittelbares Merkmal des Softwareprodukts ist, sondern lediglich eine Folge der im Produkt realisierten Funktionen ist. Ein besseres Maß für die Länge seien deshalb Function Points.

Wie das oben gegebene Beispiel zeigt, sind die empirischen Relationen häufig nicht klar und deshalb ist die Festlegung auf einen bestimmten Skalentyp oft anfechtbar. Für die Bestimmung des Skalentyps kann folgender Katalog von Forderungen an das Untersuchungsmerkmal herangezogen werden:

Nominalskala Das Merkmal besitzt voneinander unterscheidbare Ausprägungen, diese lassen sich jedoch nicht hinsichtlich des Untersuchungsziels anordnen.

Ordinalskala Die Merkmalsausprägungen können hinsichtlich des Untersuchungsziels in eine Reihenfolge gebracht werden, es können jedoch keine sinnvollen Aussagen über den Abstand zwischen den Ausprägungen gemacht werden.

Intervallskala Der Abstand zwischen je zwei Merkmalsausprägungen kann quantifiziert werden. Die Abstände sind miteinander vergleichbar. Es existiert jedoch kein Nullpunkt und deshalb können keine Aussagen über das Verhältnis von Ausprägungen getroffen werden.

Verhältnisskala Es macht Sinn, über Vielfache von Merkmalsausprägungen zu sprechen. Das Nichtvorhandensein einer Ausprägung kann interpretiert werden. Der genaue Wert der Merkmalausprägung bei einem Untersuchungsgegenstand ist irrelevant, solange das Verhältnis zu Merkmalsausprägungen bei anderen Gegenständen gewahrt bleibt.

Absolutskala Der genaue Wert einer Merkmalausprägung ist von Interesse. Eine verhältniserhaltende Skalierung der Merkmalsausprägungen bei allen Untersuchungsgegenständen würde die mit dem Messwert verbundene Aussage verändern.

Aufgrund zunehmend besserer Analysemöglichkeiten, werden in der Praxis Maße mit einem möglichst hohen Skalenniveau angestrebt. Da die empirischen Relationen nicht klar sind oder keine genaue Quantifizierung zulassen, kann nur schwer nachgewiesen werden, dass ein bestimmtes Skalenniveau zulässig ist. Würde man den exakten Nachweis fordern, so müsste man sich auf Ordinal- oder Nominalniveau begeben und könnte einige nützliche Analysemethoden nicht anwenden. So liegt häufig der Fall vor, dass das Niveau eines Maßes zwischen Ordinal- und Intervallniveau vermutet wird. Wenn man sich in diesem Fall auf die Ordinalskala beschränkt, verliert man wertvolle Informationen. Deshalb wird die Anwendung entsprechender statistischen Techniken in der Praxis als gerechtfertigt angesehen, wenn genügend Indizien für die Zugrundelegung eines möglichst hohen Niveaus vorliegen. Bei der Interpretation der Ergebnisse muss diese Tatsache jedoch berücksichtigt werden.

4.4 Gewünschte Eigenschaften von Maßen

In der Literatur werden verschiedene Eigenschaften genannt, die Maße im Idealfall besitzen sollen. Bei der Beschreibung einiger von ihnen wird in verschiedenen Literaturquellen der Schwerpunkt auf unterschiedliche Aspekte gelegt. Also varihiert nicht nur die Auswahl an beschriebenen Eigenschaften, sondern auch deren Bedeutung. Deshalb ist eine zusammenfassende Beschreibung schwierig. Hier wird eine Auswahl an gewünschten Eigenschaften präsentiert, die für diese Arbeit relevant zu sein scheinen.

Gültigkeit [Kan, 2002], [Fenton and Pfleeger, 1998]
Die Gültigkeit eines Maßes ist erste Voraussetzung für seinen praktischen Einsatz. Ein Maß ist gültig, wenn sie auf einem Modell basieren, welches das Verhalten der Gegenstände in der realen Welt bezogen auf die verfolgten Ziele korrekt nachbildet. Oft ist es nicht klar, ob ein Maß tatsächlich das Merkmal charakterisiert, was es beschreiben soll. Wenn ein Maß gültig ist, dann ist die Repräsentationsbedingung erfüllt. Diese Bedingung kann geprüft werden, indem ausgehend vom Modell des Untersuchungsgegenstandes Operationen identifiziert werden, die auf den Untersuchungsgegenstand angewendet werden können. Diese werden dann angewendet, um deren Wirkung auf das Maß zu beobachten. Sie muss den intuitiven Vorstellungen über die Veränderung der Messwerte entsprechen. So muss beispielsweise die Kopplung eines System kleiner werden, wenn ein Subsystem hinzugefügt wird, ohne mit den anderen Subsystemen vernetzt zu werden.

Zuverlässigkeit [Henderson-Sellers, 1996]
Die Zuverlässigkeit liegt vor, wenn ein Maß konsistente Resultate liefert.

Ein Maß soll bei Anwendung auf den gleichen Untersuchungsgegenstand unter gleichen Bedingungen das gleiche Ergebnis liefern.

Objektivität [Bennicke and Rust, 2005]
Fast alle Messung enthalten einen subjektiven Anteil. Der Grund dafür besteht darin, dass das Ergebnis zum Teil auch von der Umgebung abhängt, in der eine Messung durchgeführt wird. Der subjektive Anteil einer Messung muss so gering wie möglich gehalten werden, um Messungen miteinander vergleichen zu können, die von verschiedenen Personen zu verschiedenen Zeitpunkten an verschiedenen Ausprägungen eines Untersuchungsgegenstandes vorgenommen werden.

Sensitivität [Bennicke and Rust, 2005]
Ein Maß muss eine signifikante Unterscheidung zwischen Entitäten ermöglichen. Beispielsweise müssen Softwareprodukte mit einer merklich höheren Qualität auch deutlich höhere Messwerte erzielen als Produkte mit einer geringen Qualität.

Rentabilität [Ludewig and Lichter, 2007]
Damit eine Messung sich aus der wirtschaftlichen Sicht lohnt, muss der Aufwand bei der Erhebung von Daten vertretbar sein.

Verfügbarkeit [Ludewig and Lichter, 2007]
Es muss möglich sein, die erforderlichen Daten zu erheben, damit die Maßdefinition ihren Zweck erfüllt.

Plausibilität [Ludewig and Lichter, 2007]
Das Maß muss verständlich sein, anderenfalls erbringt es keinen Nutzen.

Kapitel 5

Statistische Analyseverfahren

Zur Untersuchung von Hypothesen können verschiedene statistische Verfahren angewendet werden. Eine Hypothese ist eine unbewiesene Annahme oder Behauptung, die zur Überbrückung wissenschaftlicher Unkenntnis geeignet ist. Eine Arbeitshypothese ist eine Hypothese, die dabei helfen soll, bestimmte Experimente bzw. Lösungsansätze zu planen, die zur Verifizierung einer Theorie dienen.

Die Wahl eines Verfahrens hängt unter anderem von den aufgestellten Hypothesen ab. So können einige Hypothesen mit vergleichsweise einfachen Verfahren wie Parametervergleich oder Korrelationsanalyse [Bosch, 1990], [Rönz, 2007], [Hartung et al., 1993], [Leonhart, 2004] untersucht werden, während andere Hypothesen weitere etwas aufwendigere Untersuchungsschritte erfordern.

In dieser Arbeit wurden die aufgestellten Hypothesen mit Hilfe eines Parametertests untersucht, um festzustellen, ob eine Tendenz für einen Zusammenhang zwischen zwei Größen vorliegt. Wenn eine Tendenz festgestellt wird, wird eine Korrelationsanalyse durchgeführt.

Bei der Korrelationsanalyse wird überprüft, wie gut der Zusammenhang zwischen den Größen druch eine Fuktion der Form $Y = A \cdot X + B$ approximiert werden kann. Es wird also untersucht, wie entscheidend den Indikator im Vergleich zu den nicht untersuchten Indikatoren für die abhängige Größe ist.

Es wurde darüber hinaus Regressionsanalyse benutzt, um festzustellen, wie hoch Wahrscheinlichkeit einer Scheinkorrelation zwischen bestimmten Größen ist.

5.1 Grundprobleme eines Hypothesentests

In der Statistik werden statistische Mengen untersucht, um Hypothesen zu überprüfen. Statistische Mengen sind Gesamtheiten von Ereignissen. Die Menge aller Ereignisse, die ein bestimmtes Merkmal besitzen, bezeichnet man als Grundgesamtheit. So bilden alle Entwurfseinheiten innerhalb einer Zyklengruppe eine Grundgesamtheit. Bei statistischen Untersuchungen ist es in der Regel aus

Kostengründen nicht möglich, eine bestimmte Grundgesamtheit vollständig zu untersuchen. Deshalb wird versucht, aus Eigenschaften von Teilmengen einer Grundgesamtheit die Wahrscheinlichkeit für das Auftreten eines bestimmten Merkmals in der Grundgesamtheit zu schätzen und die Signifikanz des Schätzwertes zu beurteilen.

Eine aus einer Grundgesamtheit ausgewählte Menge mit n Elementen heißt Stichprobe. Die Elemente der Stichprobe bilden eine Messreihe X. Die Anzahl n der Elemente gibt den Umfang der Stichprobe an, in dieser Arbeit als Stichprobenumfang bezeichnet. Jedes einzelne Element der Stichprobe heißt Stichprobenwert.

Um aus Eigenschaften der Stichprobe mit einer gewissen Sicherheit auf Eigenschaften der Grundgesamtheit schließen zu können, muss die Stichprobe repräsentativ für die Grundgesamtheit sein. Eine Stichprobe gilt als repräsentativ, wenn sie annähernd so wie die Grundgesamtheit zusammengesetzt und der Stichprobenumfang hinreichend groß ist. Trotz eines großen Stichbrobenumfangs bleibt die Entscheidung stets mit einer gewissen Unsicherheit behaftet. Daher ist es wichtig, die Wahrscheinlichkeiten für Fehlentscheidungen abschätzen zu können.

Bei der Überprüfung einer Hypothese können zwei Fehlentscheidungen getroffen werden. Eine falsche Hypothese kann fälschlicherweise angenommen werden. Dieser Fehler wird als Fehler erster Art bezeichnet. Oder es kann auch passieren, dass eine in Wirklichkeit richtige Hypothese fälschlicherweise abgelehnt wird. Dieser Fehler wird als Fehler zweiter Art bezeichnet.

Die Wahrscheinlichkeit für den Fehler erster Art nennt man Irrtumswahrscheinlichkeit α. Sie wird meist vor Beginn des Tests festgelegt oder aus Testdaten berechnet und häufig auch als Signifikanzniveau α. In Abhängigkeit von den konkreten Gegebenheiten und den Ansprüchen an die Sicherheit der Testergebnisse sind Irrtumswahrscheinlichkeiten von 1 % ($\alpha = 0.01$) bis 10 % ($\alpha = 0.1$) allgemein üblich.

Bei der Untersuchung von Hypothesen wird üblicherweise versucht, die Wahrscheinlichkeit für den Fehler erster Art zu minimieren. Dafür wird eine zu der als richtig vermuteten Hypothese $H1$ gegensätzliche Hypothese $H0$ formuliert und untersucht. $H0$ wird als Nullhypothese, und $H1$ als Alternativhypothese bezeichnet.

Weisen die Untersuchungsergebnisse der Stichprobe große Abweichungen von der Nullhypothese auf, so spricht man von einem signifikanten Unterschied zwischen der Nullhypothese und der Stichprobe. Die Nullhypothese wird in diesem Fall abgelehnt und die Alternativhypothese angenommen. Die Testentscheidung über eine Ablehnung der Nullhypothese wird anhand eines Ablehnungsbereiches ermittelt. Dieser Ablehnungsbereich ist abhängig von der gewählten Irrtumswahrscheinlichkeit und dem Stichprobenumfang. Die gewählte Hypothese $H0$ wird abgelehnt, wenn gilt:

$$t \leq -t_{1-\alpha} \text{ mit } F(t_{1-\alpha}) = 1 - \alpha$$

$t_{1-\alpha}$ ist dabei ein Quantil der t-Verteilung, welches für das gewünschte α und den vorliegenden Stichprobenumfang ($n_D + n_{\bar{D}} - 2$ Freiheitsgrade) der Tabelle im

Anhang entnommen werden kann. In allen Analysen im Rahmen dieser Arbeit wird die Irrtumswahrscheinlichkeit auf 0.01 festgelegt.

Bei der statistischen Hypothesenprüfung muss immer bedacht werden, dass diese lediglich zu Wahrscheinlichkeitsangaben führt darüber, wie gut oder schlecht das empirische Ergebnis mit der Nullhypothese vereinbar ist. Es können jedoch keine absolut sicheren Aussagen auf der Basis statistischer Prüfungen getroffen werden.

5.2 Vergleich zweier Parameter

Bei diesem Verfahren werden die unbekannten Parameter zweier verschiedener Grundgesamtheiten miteinander verglichen und es wird geprüft, ob deren Differenz gleich, kleiner oder größer als ein vorher bestimmter Wert d ist. In dieser Arbeit wird überprüft, ob die Differenz der Erwartungswerte von zwei Mengen von Entwurfseinheiten größer als null ist. Basis der Überprüfungen ist die Nullhypothese. Es sind genau zwei Prüfergebnisse möglich sind: die Nullhypothese ist abzulehnen oder die Nullhypothese kann nicht abgelehnt werden.

Als Näherung für den Erwartungswert wird der Mittelwert über die Werte aller Entwurfseinheiten einer Grundgesamtheit gebildet, wobei die einzelnen Entwurfseinheiten die Stichproben darstellen. Die Grundegesamtheiten der Entwurfseinheiten werden gebildet, indem diese anhand eines bestimmten Merkmals in zwei Gruppen aufgeteilt werden. So werden beispelsweise bei der Untersuchung der ersten Hypothese die Entwurfseinheiten, je nach dem ob sie einen höheren oder einen niedrigeren Wert des Collective Code Ownership als der Wert des Medians haben, in die erste oder die zweite Gruppe eingeteilt. Die zu vergleichenden Erwartungswerte eines Attributes seien also wie folgt definiert:

$$\mu_D = \frac{1}{|D|} \cdot \sum_{a \in D} attribut(a)$$

$$\mu_{\bar{D}} = \frac{1}{|\bar{D}|} \cdot \sum_{a \in \bar{D}} attribut(a)$$

Dann wäre die zu überprüfende Hypothese $H1$:

$$H1: \ \mu_{\bar{D}} - \mu_D < 0$$

Und die Nullhypothese $H0$:

$$H0: \ \mu_{\bar{D}} - \mu_D \geq 0$$

Um zu prüfen, ob die Hypothese $H0$ mit dem gewünschten Signifikanzniveau abgelegt werden kann, muss der Testwert t berechnet werden und anschließend mit dem kritischen Wert $t_{1-\alpha}$ verglichen werden. Für die Berechnung des Testwertes werden die Varianzen s^2der beiden Grundgesamtheiten benötigt. Mit dem Stichprobenumfang n und dem Mittelwert $\bar{x}$ der Stichprobenwerte ist die Varianz einer Grundgesamtheit wie folgt definiert:

$$s^2 = \frac{1}{n-1} \cdot \sum (x_i - \bar{x})^2$$

Die Gesamtvarianz s^2_{ges} der beiden Stichproben ist:

$$s^2_{ges} = \frac{1}{n_D + n_{\bar{D}} - 2} \cdot ((n_D - 1) \cdot s^2_D + (n_{\bar{D}} - 1) \cdot s^2_{\bar{D}})$$

Der Testwert t wird aus diesen Daten wie folgt berechnet:

$$t = \frac{\mu_D - \mu_{\bar{D}} - d}{\sqrt{s^2_{ges}}} \cdot \sqrt{\frac{n_D \cdot n_{\bar{D}}}{n_D + n_{\bar{D}}}}$$

Die Hypothese H0 kann also abgelehnt und H1 angenommen werden, wenn gilt:

$$t < t_{o.99}$$

Diese Entscheidung ist zu 99 Prozent abgesichert. Wenn ausreichend große Mengen an Daten untersucht werden, werden beim Parametervergleich aufgrund einer hohen Datenstreuung die Wrikungen störender Faktoren ausgeglichen, so dass die Tendenz mit einer hohen statistischen Signifikanz nicht auf Wirkung anderer Faktoren zurückzuführen ist. Inwieweit der Einfluss des untersuchten Indikators jedoch im Vergleich zu anderen Einflüssen entscheidend ist, kann auf diese Weise nicht festgestellt werden, da die Gesamtvarianz der abhängigen Größe zwar zur Berechnung des Signifikanzniveaus, jedoch nicht zur Ergänzung des Ergebnisses herangezogen wird.

5.3 Korrelationsanalyse

Der Pearson-Korrelationskoeffizient[1] gibt an, wie hoch der funktionale Zusammenhang zwischen zwei Variablen ist. Die Variablen sind in diesem Fall Messwerte von zwei verschiedenen Maßen. Um diesen Koeffizienten zu berechnen, kann die folgende Formel verwendet werden:

$$r = r(x,y) = \frac{n \sum_{i-1}^{n} x_i y_i - \sum_{i-1}^{n} x_i \sum_{i-1}^{n} y_i}{\sqrt{\left(n \sum_{i=1}^{n} x_i^2 - \left(\sum_{i-1}^{n} x_i\right)^2\right)\left(n \sum_{i=1}^{n} y_i^2 - \left(\sum_{i-1}^{n} y_i\right)^2\right)}}$$

Dabei werden beispielsweise die Anzahl der Entwickler x und die Anzahl der Änderungen y für jede der n Entwurfseinheiten aus der Datenbank ausgelesen und in diese Formel als x_i und y_i für die i-te Entwurfseinheit eingesetzt. Ein Korrelationkoeffizient kann einen Wert zwischen -1 und 1 annehmen. Dabei ist der funktionale Zusammenhang am stärksten, wenn der Betrag des Korrelationskoeffizienten 1 ist. Dann lässt sich der Zusammenhang als eine lineare Funktion der Form $y = a \cdot x + b$ darstellen. Dieser Fall liegt beispielsweise vor, wenn eine Korrelation einer Variablen mit sich selbst berechnet wird. Ein negativer Wert

[1] In dieser Arbeit wird der Pearson-Korrelationskoeffizient verwendet. Im Folgenden wird vereinfachend von einem Korrelationskoeffizienten anstelle von Pearson-Korrelationskoeffizienten besprochen.

sagt aus, dass der Zusammenhang gegenläufig ist. Bei der Interpretation dieses Wertes ist Vorsicht geboten, da es auch vorkommen kann, dass der Wert des Korrelationskoeffizienten auf einen nicht existenten Zusammenhang hindeutet. Wie hoch die Wahrscheinlichkeit ist, dass dieser Wert durch Zufall entstanden ist, wird durch das Signifikanzniveau einer Berechnung gemessen. Dieses muss stets zusammen mit dem Koeffizienten angegeben werden, da dieser allein nicht interpretiert werden kann. Das Signifikanzniveau wird in Prozent oder als eine Dezimalzahl zwischen null und eins angegeben.

Um zu berechnen, ob der Korrelationkoeffizient ein bestimmtes Signifikanzniveau hat, muss der folgende Testwert berechnet werden:

$$t = \frac{r}{\sqrt{1-r^2}} \cdot \sqrt{n-2}$$

Dieser Testwert wird mit dem kritischen Wert verglichen, der aus der Tabelle im Anhang abgelesen werden kann. Wenn der Testwert größer ist als der kritische Wert, dann kann die Nullhypothese abgelehnt werden, die besagt, dass die untersuchten Variablen nicht miteinander korrelieren.

Je höher die Zahl n der Entwurfseinheiten und der Korrelationskoeffizient r, desto höher ist das mögliche zu erreichende Signifikanzniveau.

Kapitel 6

Maße und Indikatoren in der Fallstudie

Um die Maße für die Bewertung der Prinzipien Collective Code Ownership und Zyklenfreiheit definieren zu können, werden hier die in Kapitel zwei formulierten Hypothesen erneut aufgeführt:

Hypothese 1: Entwurfseinheiten mit hohem Ausmaß an praktiziertem Collective Code Ownership weisen eine Tendenz zu einer höheren Fehleranfälligkeit auf als die Entwurfseinheiten mit einem geringen Ausmaß an praktiziertem Collective Code Ownership

Hypothese 2: Entwurfseinheiten in Zyklen weisen eine Tendenz zu einer höheren Fehleranfälligkeit auf als die Entwurfseinheiten außerhalb von Zyklen

Hypothese 3: Entwurfseinheiten in größeren Zyklengruppen weisen eine Tendenz zu einer höheren Fehleranfälligkeit auf als die Entwurfseinheiten in kleineren Zyklengruppen

Hypothese 4: Entwurfseinheiten in stärker verflochtenen Zyklengruppen weisen eine Tendenz zu einer höheren Fehleranfälligkeit auf als die Entwurfseinheiten in weniger verflochtenen Zyklengruppen

Für die Untersuchung der Hypothesen müssen also die nachfolgenden Merkmale messbar sein:

- Fehleranfälligkeit einer Entwurfseinheit
- Praktiziertes Collective Code Ownership
- Zugehörigkeit einer Entwurfseinheit zu einer Zyklengruppe
- Größe einer Zyklengruppe

- Verflochtenheit einer Zyklengruppe

Während Zugehörigkeit zu einer Zyklengruppe und Größe der Zyklengruppen einfach zu ermitteln sind, ist die Berechnung der übrigen Größen aufwendig. Das liegt daran, dass für die Fehleranfälligkeit der Aufwand einer Änderung und für die Verflochtenheit die Anzahl der aufzulösenden Abhängigkeiten gemessen werden müssen. Der Aufwand einer Änderung und die Anzahl der aufzulösenden Abhängigkeiten können nur mit Schwierigkeiten ermittelt werden. Es ist erforderlich, vereinfachende Annahmen zu treffen, da der tatsächliche Aufwand von vielen Faktoren abhängt und eine Berücksichtigung aller dieser Faktoren zu aufwendig wäre. Es sind unterschiedliche vereinfachende Annahmen möglich. Es müssen deshalb Entscheidungen getroffen werden, welche der Annahmen in diesem konkreten Fall am sinnvollsten sind. Ohne eine Analyse der vorliegenden Daten ist eine Definition geeigneter Maße schwierig. Deshalb wurden die bereits gesammelten Daten vor der Definition der Maße analysiert. Anschließend werden die Maße für die Untersuchung der Hypothesen definiert.

6.1 Notwendige Vorüberlegungen

Die Gültigekeit der Aussagen artiger Studien hängt in einem hohem Maße von der geeigneten Festlegung der Maße ab. Deshalb ist dabei eine besondere Sorgfalt erforderlich. Wie man bei der Festlegung der Maße vorgehen soll, scheint jedoch eine schwierige Entscheidung zu sein.

Die Fehleranfälligkeit und praktiziertes Collective Code Ownership wurden in dieser Arbeit unter Berücksichtigung des Aufwands definiert. In vielen Studien über die Zuverlässigkeit von Systemen, unter anderem auch in einer der bereits erwähnten verwandten Studien, wird im Gegensatz dazu die Anzahl der Fehler in Entwurfseinheiten verwendet, um die Qualität des Quelltextes zu bewerten. Es wird dabei von einer Validierung von Maßen als Indikatoren der Qualität gesprochen. Beispielsweise wird in

- [Basili et al., 1996]
- [Denaro et al., 2002]
- [Khoshgoftaar and Allen, 2003]
- [Schröter et al., 2006a]
- [Schröter et al., 2006b]
- [Zimmermann and Nagappan, 2006]
- [Nagappan et al., 2006]
- [Nagappan and Ball, 2006]

angestrebt, die Effektivität der Maßnahmen zur Qualitätssicherung zu erhöhen, indem diejenigen Entwurfseinheiten indentifiziert werden, bei denen die Wahrscheinlichkeit für das Vorhandensein eines Fehlers besonders hoch ist. Diese Wahrscheinlichkeit orientiert sich an der Anzahl der Fehler in einer Entwurfseinheit. Es wird also angenommen, dass Entwurfseinheiten mit einer höheren Wahrscheinlichkeit für Fehler eine niedrigere Qualität besitzen als Entwurfseinheiten mit einer geringeren Wahrscheinlichkeit.

Dabei wird nicht berücksichtigt, dass größere Entwurfseinheiten zwar eine höhere Wahrscheinlichkeit für Fehler haben, dass deren Inspektion oder Test jedoch auch aufwendiger sind. Aus der Angabe dieser Wahrscheinlichkeit kann nicht auf die Effektivität der Maßnahmen zur Qualitätssicherung geschlossen werden, da nicht bekannt ist, ob bei der Qualitätssicherung unter Einsatz eines bestimmten zeitlichen Aufwands mehr Fehler in den Entwurfseinheiten mit höheren Wahrscheinlichkeiten für Fehler gefunden werden als in Entwurfseinheiten mit einer geringeren Wahrscheinlichkeit.

Die Aussage, dass zwischen einem bestimmten Maß und der Anzahl der Fehler ein starker Zusammenhang besteht, reicht also nicht aus, um dieses Maß als einen guten Indikator für Qualität des Quelltextes zu bestätigen. Ein Maß kann erst dann als ein Indikator für niedrige Qualität angesehen werden, wenn durch eine bewusste Verringerung des Messwertes, beispielsweise der Anzahl der Entwickler, eine Verringerung der Fehleranzahl im gesamten System erreicht wird. Wenn jedoch sowohl die Anzahl der Fehler als auch die Anzahl der Entwickler von der Größe der Entwurfseinheiten abhängen, wird eine Maßnahme zur Reduzierung der Anzahl der Entwickler nur dann zu einer Verringerung der Fehleranzahl in einer Entwurfseinheit führen, wenn dabei auch die Größe dieser Entwurfseinheit verringert wird. Wenn die Größe einer Entwurfseinheit durch eine Verschiebung von Quelltext aus dieser Entwurfseinheit in eine andere erreicht wird, bleibt der Umfang des gesamten Systems, und damit auch die Anzahl der Fehler in diesem, unverändert. Deshalb wäre in diesem Fall eine Orientierung an dem korrelierenden Maß bei der Entwicklung von Systemen nicht sinnvoll.

Zu der Entscheidung, bei der Betrachtung der Qualität den Aufwand zu berücksichtigen, kann auf eine logische Weise begründet werden. Denn nur in Verbindung mit diesem kann die Anzahl der Fehler einen Hinweis darauf geben kann, wie sorgfältig bei der Entwicklung vorgegangen wurde. Und das entspricht einer intuitiven Vorstellung von der Qualität des Quelltextes.

Es ist jedoch nicht immer ausreichend, eine logisch erscheinende Begründung für die Wahl der Maße anzugeben, um die Gültigkeit der Aussagen sicherzustellen. Die bei einer Untersuchung getroffenen Annahmen müssen anahnd der vorliegenden Daten überprüft werden. So ist die Größe der Entwurfseinheiten in dieser Studie nicht der einzige Faktor, dessen Vernachlässigung die Gültigkeit von Aussagen gefährden kann. Wenn zwei Maße sowohl untereinander als auch mit einer weiteren Größe korrelieren, kann nicht ausgeschlossen werden, dass der Zusammenhang zwischen den beiden Maßen durch diese dritte Größe bedingt ist. Die Korrelation zwischen den beiden Maßen wird in diesem Fall als Scheinkorrelation bezeichnet. Bei derartigen Untersuchungen kann man deshalb nicht sicher sein, dass es sich bei einem festgestellten Zusammenhang nicht um eine

Scheinkorrelation handelt. Auch im Falle gut fundierter Definitionen der Maße kann immer noch ein Faktor existieren, der zu einer Scheinkorrelation führt.

Die Gefahr für eine Scheinkorrelation kann dadurch verringert werden, dass man versucht, so viele weitere Faktoren wie möglich in die Betrachtung einzubeziehen. Die im Rahmen dieser Betrachtung festgestellten Zusammenhänge müssen bei der Festlegung der Maße berücksichtigt werden, indem sie in die vor der Betrachtung vorhandene Vorstellung des untersuchten Gegenstandsbereichs einbezogen werden. Dafür muss zunächst nach einer Erklärung der festgestellten Zusammenhänge gesucht werden. Anschließend muss die Verträglichkeit der vor dieser Betrachtung getroffenden Annahmen mit dieser Erklärung geprüft werden.

Im Rahmen dieser Studie wurde bei der Suche nach geeigneten Maßen untersucht, durch welche Faktoren die Korrelation zwischen der Anzahl der Fehler und der Anzahl der Entwickler bedingt sein kann. Als mögliche Faktoren wurden die Größe der Dateien, die Anzahl der Änderungen und das Alter der Dateien betrachtet. Die Ergebnisse der paarweisen Korrelation für alle Daten auf Dateiebene befinden sich im Anhang. Die Korrelationskoeffizienten sind signifikant mit $\alpha = 0.01$, also mit 99% Sicherheit.

Es wurde festgestellt, dass die Größe der Dateien und die Anzahl der Änderungen möglicherweise eine Scheinkorrelation verursachen. Aufgrund dieser Feststellung musste die Modellierung des Aufwands zur Implementieung einer bestimmten Version der Entwurfseinheit E angepasst werden.

Dieser setzt sich aus den Aufwänden der einzelnen Änderungen von E zusammen. Die den Aufwand einer Änderung bestimmenden Faktoren hängen in hohem Maße von der Art dieser Änderung ab. Der Aufwand bei der Erstellung einer Entwurfseinheit hängt beispielsweise stark von dem Umfang des erstellten Quelltextes ab. Nachträgliche Änderung können jedoch vom Umfang her geringfügig sein, gleichzeitig aber viel Zeit in Anspruch nehmen. Das liegt daran, dass es häufig viel aufwendiger ist, die Stelle im Quelltext zu finden, an der diese Änderung vorzunehmen ist, und zu klären, wie sie aussehen soll, als diese letztendlich vorzunehmen. Der Aufwand einer nachträglichen Änderung kann deshalb präziser durch den Umfang des zu analysierenden Quelltextes als durch den Umfang der Änderung charakterisiert werden.

Vor der Korrelationsanalyse wurde angenommen, der Anfwand zur Implementierung von E bestehe hauptsächlich in der ersten Änderung. Er hänge also hauptsächlich von der Größe von E ab. Diese Annahme wäre angemessen, wenn Entwurfseinheiten nach ihrer Implementierung nur wenig geändert worden wären. In diesem System ist es jedoch nicht der Fall. Zum einen wird aus den Daten deutlich, dass viele Entwurfseinheiten nach ihrer Erstellung mehrmals geändert werden. Zum anderen weist der Korrelationskoeffizient von 0.86 zwischen der Anzahl der Änderungen und der Anzahl der Fehler darauf hin, dass viele Fehler erst nach der Erstellung einer Entwurfseinheit entstehen. Das hat zur Folge, dass nachträgliche Änderungen bei der Modellierung des Aufwands nicht außer Acht gelassen werden können.

Deshalb wird in dieser Arbeit versucht, sowohl den Aufwand der Änderungen erster Art, als auch den Aufwand der Änderungen zweiter Art zu berücksichti-

gen. Es wird also angenommen, dass nach der Erstellung einer Entwurfseinheiten kleine Änderungen stattfinden. Dabei wird für den Umfang des zu analysierenden Quelltextes angenommen, dass er von dem Umfang der zu ändernden Entwurfseinheit abhängt.

Aufgrund der getroffenen Annahmen gilt also sowohl für die Erstellung einer Entwurfseinheit, als auch für deren nachträgliche Änderung, dass der Aufwand dieser Änderungen von der Größe der Entwurfseinheit abhängt. Aufbauend auf diesen Betrachtungen können in den nachfolgenden Abschnitten entsprechned den getroffenen Annahmen die Maße definiert werden.

6.2 Fehleranfälligkeit

In Kapitel zwei wurde die Fehleranfälligkeit einer Systemversion X definiert als die Gesamtschwere ihrer Fehler pro Einheit des minimalen erforderlichen Aufwandes:

$$F(X) = \frac{Q(X)}{K(X)}$$

Analog dazu kann die Fehleranfälligkeit einer Version von einer Entwurfseinheit E definiert werden als

$$F(E) = \frac{Q(E)}{K(E)}$$

wobei $Q(E)$ die Gesamtschwere aller in dieser Version enthaltener Fehler ist und $K(E)$ die der für ihre Erstellung erforderliche minimale Aufwand ist.

Der Aufwand zur Erstellung einer Systemversion soll gleich der Summe von Aufwänden aller seiner Entwurfseinheiten E_i sein. Der Aufwand zur Erstellung einer Version von einer Entwurfseinheit setzt sich aus den Aufwänden der einzelnen Änderungen A_{ij}, die für die Implementierung der letzten Version erforderlich waren.

$$K(X) = \sum_{i \in I} K(E_i) = \sum_{i \in I} \sum_{j \in J} K(A_{ij})$$

Zusammenfassend kann die Fehleranfälligkeit eines Systems geschrieben werden als:

$$F(X) = \frac{\sum_{i \in I} Q(E_i)}{\sum_{i \in I} \sum_{j \in J} K(A_{ij})}$$

Um ein Maß für die Fehleranfälligkeit $F(E)$ der Version E einer Entwurfseinheit festzulegen, müssen zunächst Maße für $Q(E)$ und $K(E)$ bestimmt werden. Als ein Maß für $Q(E)$ wird in dieser Arbeit die Anzahl $numFaults(E)$ der in dieser Version der Entwurfseinheit enthaltenen Fehler verwendet.

$K(E)$ hängt von der Anzahl der Änderungen $numChanges(E)$, die zur Erstellung von E erforderlich waren, sowie den Äufwänden der einzelnen Änderungen ab. Die Anzahl der Änderungen kann den Daten direkt entnommen werden. Der Aufwand einer Änderung wird in dieser Arbeit durch die Größe $length(E)$ der geänderten Entwurfseinheit bestimmt. Als Maß für die Fehleranfälligkeit einer Entwurfseinheit wird also in dieser Arbeit der folgende Ausdruck verwendet:

$$relNumFaults(E) = \frac{numFaults(E)}{length(E) \cdot numChanges(E)}$$

Wertebereich und Skalentyp

Die Fehleranfälligkeit einer Entwurfseinheit ist eine positive rationale Zahl. Da eine Aussage über das Verhältnis zweier Entwurfseinheiten bezüglich ihrer Fehleranfälligkeit möglich sein soll, muss diesem Maß eine Verhältisskala zu Grunde liegen.

Erforderliche Daten

Um $relNumFaults(E)$ zu berechnen, müssen die folgenden Daten erhoben werden:

- Anzahl der Fehler
- Größe
- Anzahl der Änderungen

Dabei werden nur Änderungen gezählt, die nicht mit dem Ziel durchgeführt wurden, einen Fehler zu beheben. Die Größe der Entwurfseinheiten wird in Zeichen gemessen. Es werden dabei keine Zeichen gezählt, die zu einem Kommentar gehören. Was unter einem Fehler verstanden wird, wurde bereits in Kapitel zwei beschrieben.

6.3 Praktiziertes Collective Code Ownership

Das praktizierte Collective Code Ownership für eine bestimmte Version E einer Entwurfseinheit wurde in Kapitel zwei definiert als die Anzahl der Entwickler, die ihre Änderungsrechte für diese wahrgenommen haben, pro Einheit des für ihre Implementierung erforderlichen Aufwands. Wenn der erforderliche Aufwand wie im vorigen Abschnitt berechnet wird, kann das praktizierte Collective Code Ownership $PCCO(E)$ folgendermaßen ermittelt werden:

$$PCCO(E) = \frac{numAuthors(E)}{\sum_{j \in J} K(A_j)}$$

Dabei ist $numAuthors(E)$ die Anzahl der Entwickler, welche die Entwurfseinheit E geändert haben. Für das ganze System ist das praktizierte Collective Code Ownership dann wie folgt definiert:

$$PCCO(X) = \frac{\sum_{i \in I} numAuthors(E_i)}{\sum_{i \in I} \sum_{j \in J} K(A_{ij})}$$

Wenn das im vorigen Abschnitt beschriebene Maß für den Aufwand verwendet wird, kann der folgende Ausdruck als Maß für das Collective Code Ownership verwendet werden:

$$relNumAuthors(E) = \frac{numAuthors(E)}{length(E) \cdot numChanges(E)}$$

Dabei ist $numAuthors(E)$ die Anzahl der Autoren, welche sich an der Entwicklung dieser Version der Entwurfseinheit beteiligt haben.

Wertebereich und Skalentyp

Das praktizierte Collective Code Ownership einer Entwurfseinheit ist eine positive rationale Zahl. Da eine Aussage über das Verhältnis zweier Entwurfseinheiten bezüglich dieser Größe möglich sein soll, muss diesem Maß eine Verhältisskala zu Grunde liegen.

Erforderliche Daten

Für dieses Maß müssen die folgenden Daten erhoben werden:

- Anzahl der Autoren
- Größe
- Anzahl der Änderungen

Dabei werden nur Änderungen gezählt, die nicht mit dem Ziel durchgeführt wurden, einen Fehler zu beheben. Die Größe der Entwurfseinheiten wird in Zeichen gemessen. Es werden keine Zeichen gezählt, die zu einem Kommentar gehören.

6.4 Zugehörigkeit zu einer Zyklengruppe

Die Zugehörigkeit zu einer Zyklengruppe für eine Entwurfseinheit ist ein direkt messbares Merkmal. Es kann nur zwei Werte annehmen. Der Wert ist eins, wenn die betrachtete Entwurfseinheit mehr als einen Nachbarn in der zugehörigen Zyklengruppe hat, wenn also die Zyklengruppe nicht nur aus der Entwurfseinheit selbst besteht. Andernfalls ist sie null.

Bezeichnung: $isInCycle(E)$

Wertebereich: $0, 1$

Skalentyp: Ordinalskala

6.5 Größe der Zyklengruppen

Unter Größe der Zyklengruppen wird in dieser Arbeit die Anzahl der sie bildenden Entwurfseinheiten verstanden. Diese kann auch direkt gemessen werden. Wenn diese Entwurfseinheit in keiner Zyklengruppe enthalten ist, ist dieser Wert gleich eins.

Bezeichnung: $cycleSize(Z)$

Wertebereich: N

Skalentyp: Absolutskala

6.6 Verflochtenheit der Zyklengruppen

Die Verflochtenheit einer Zyklengruppe gibt an, wie aufwendig ihre Auflösung ist. Diese Eigenschaft hängt davon ab, wie viele Abhängigkeiten zwischen den Entwurfseinheiten aufgelöst werden müssen und wie schwer es ist, sie aufzulösen.

$$cycleDensity(Z) = \sum_{i \in I} A(_i)$$

I ist dabei die Menge aller Indizes der in dieser Zyklengruppe vorhandenen Abhängigkeiten.

Es ist leider nicht möglich, anhand der vorliegen Daten ein präzises Maß für diese Eigenschaft zu definieren. Zum einen liegen nicht genügend Daten vor, um die Stärke der einzelnen Abhängigkeiten zu ermitteln. Deshalb wird in dieser Arbeit nur deren Anzahl betrachtet. Zum anderen ist es zu aufwendig, die Anzahl der aufzulösenden Abhängigkeiten genau zu berechnen. Dafür ist ein komplexer Algorithmus erforderlich. Es geht über den Rahmen dieser Diplomarbeit hinaus, einen derarigen Algorithmus zu entwerfen. Deshalb sind stark vereinfachende Annahmen erforderlich, um die Verflochtenheit zu bestimmen.

Wertebereich und Skalentyp

Die Verflochtenheit einer Zyklengruppe ist eine positive rationale Zahl. Es soll eine Aussage über das Verhältnis zweier Zyklengruppen bezüglich dieser Größe möglich sein. Deshalb muss einem Maß für die Verflochtenheit eine Verhältisskala zu Grunde liegen.

6.7 Anzahl der aufzulösenden Abhängigkeiten

Wenn sich zwei Entwurfseinheiten gegenseitig benutzen, muss bei der Auflösung der sie enthaltenden Zyklengruppe einer der beiden Abhängigkeiten aufgelöst werden. Deshalb stellt die Anzahl derartiger Abhängigkeiten eine mögliche untere Schranke für den gesuchten Wert dar. Die Abhängigkeit zwischen zwei sich gegenseitig benutzenden Entwurfseinheiten wird im Folgenden als eine bidirektionale Abhängigkeit bezeichnet.

Die Bidirektionalität $bidirDensity(Z)$ einer Zyklengruppe Z wird als Anzahl $numBidirDeps(Z)$ der bidirektionalen Abhängigkeiten in dieser Zyklengruppe pro Größeneinheit definiert:

$$bidirDensity(Z) = \frac{numBidirDeps(Z)}{size(Z)}$$

Die Anzahl der aufzulösenden Abhängigkeiten hängt auch davon ab, wie dicht das Netz aus Abhängigkeiten in einer Zyklengruppe ist. Diese Größe wird im folgenden als Abhängigkeitsdichte bezeichnet. Sie ist also ein weiterer möglicher Indikator für die Anzahl der aufzulösenden Abhängigkeiten. Die Abhängigkeitsdichte $depDensity(Z)$ kann als Anzahl $numDeps(Z)$ der Abhängigkeiten in einer Zyklengruppe Z pro Größeneinheit der Zyklengruppe ermittelt werden:

$$depDensity(Z) = \frac{numDeps(Z)}{size(Z)}$$

Da nicht klar ist, welcher der beiden Indikatoren besser geeignet ist, und die für eine Analyse beider Indikatoren erforderlichen Daten vorliegen, weden beide Indikatoren untersucht.

Erforderliche Daten

Um die Indikatoren *depDensity* und *bidirDensity* für eine Zyklengruppe zu bestimmen, müssen die folgenden Daten erhoben werden:

- Größe der Zyklengruppe
- Anzahl der bidirektionalen Abhängigkeiten
- Azahl der Abhängigkeiten

Die Größe der Zyklengruppen wird dabei in Dateien gemessen.

Kapitel 7

Datenaufbereitung

Es werden gekoppelte Daten der Fehlerdatenbank Bugzilla und des Versionsverwaltungssystems CVS verwendet. Da Bugzilla seine Daten unabhängig von CVS verwaltet, muss die Verbindung zwischen den Fehlerdaten und den Änderungsdaten nach der Datenerhebung durchgeführt werden. Dafür wird in [Fischer et al., 2003] ein Verfahren vorgestellt. In diesem wird zu jedem Fehler die korrigierende Änderung anhand von Kommentaren, Autoren und weiteren Attributen gesucht. Als das wichtigste Zuordnungskriterium dient dabei die den Fehler eindeutig identifizierende Nummer. Diese muss bei jeder Änderung zur Behebung dieses Fehlers im Kommentar angegeben werden, damit eine Verbindung zwischen einer Änderung und dem Fehler hergestellt werden kann.

Auf diese Weise kann die den Fehler behebende Änderung und dadurch auch die Stelle im Quelltext ermittelt werden, an der ein bestimmter Fehler beseitigt wurde. So wird ein Fehler den zugehörigen Entwurfseinheiten zugeordnet.

Über die Daten der Fehlerdatenbank und des Versionsverwaltungssystems hinaus werden Daten benötigt, die sich auf die Struktur des Quelltextes beziehen. Diese Daten können nicht den Logeinträgen des CVS entnommen werden. Um sie zu bekommen, muss eine Version des Quelltextes analysiert werden. Eine derartige Analyse wird im Folgenden als *Strukturanalyse* bezeichnet. Diese kann mit Hilfe unterschiedlicher Werkzeuge erfolgen. In dieser Studie wird wie auch in vielen weiteren Studien der Quelltext zum Zeitpunkt eines bestimmten Release (3.2.1) als Grundlage für die Strukturanalyse genommen.

Wenn man auf der Ebene von Methoden und Klassen Analysen durchführen möchte, benötigt man einen weiteren Teilschritt, in dem die Zugehörigkeit der Änderungsdaten zu Methoden und Klassen ermittelt wird, da CVS Änderungen für Dateien aufzeichnet. Dazu kann der in [Zimmermann and Weißgerber, 2004] vorgeschlagene Algorithmus benutzt werden, wenn keine Werkzeugunterstützung vorhanden ist. In dieser Studie wurde auf eine Untersuchung auf weiteren Ebenen aus zeitlichen Gründen verzichtet.

7.1 Benötigte direkte Maße

Um die zur Untersuchung der Hypothesen benötigen Maße und Indikatoren bestimmen zu können, ist eine Reihe direkter Maße benötigt, die in diesem Abschnitt beschrieben werden. Bei deren Beschreibung wird zwar allgemein von Entwurfseinheiten gesprochen, die Messungen wurden jedoch nur auf der Dateiebene durchgeführt. Das wurde durch den hohen Aufwand bedingt, der mit einer Verarbeitung der CVS-Daten auf feingranulareren Ebenen wie Klassen oder Methoden.

numFaults - Anzahl der Fehler in einer Entwurfseinheit

Was in dieser Studie unter einem Fehler verstanden wird, ist in Kapitel zwei beschrieben. Dieses Maß bestimmen zu können, sind einige aufwendige Vorbereitungsschritte erforderlich, später in diesem Kapitel beschrieben werden. Es werden dafür die Daten aus CVS sowie Bugzilla benötigt.

Bezeichnung: $numFaults(E)$

Wertebereich: N

Skala: absolut

size - Größe einer Entwurfseinheit

Unter der Größe einer Entwurfseinheit wird hier die Anzahl der Zeichen verstanden. Dieses Maß für Größe der Entwurfseinheiten erscheint genauer als die Anzahl der Anweisungen oder Zeilen, da diese ebenfalls unterschiedlich lang sein können. Es werden keine Zeichen gezählt, die zu Kommentaren gehören. Es wird dabei vernachlässigt, dass Kommentare auch gelesen und gepflegt werden müssen. Die Anzahl der Zeichen kann mit Hilfe eines Werkzeugs zur Strukturanalyse erhoben werden, wenn die zu untersuchende Version der Entwurfseinheit vorliegt.

Bezeichnung: $length(E)$

Wertebereich: N

Skala: absolut

numChanges - Anzahl der Änderungen einer Entwurfseinheit

Hierbei handelt es sich um die gesamte Anzahl der Änderungen, dier erforderlich sind, um die aktuelle Version der betroffenen Entwurfseinheit zu implementieren. Die Änderungen zur Korrektur von Fehlern werden dabei nicht als erforderlich für die Implementierung angesehen. Diese Entscheidung ist durch den folgenden Sachverhalt begründet. Da Fehler häufig Änderungen zu ihrer Korrektur nach sich ziehen, hängt die Anzahl der Änderungen unter anderem auch von der Anzahl der Fehler ab. Für die Untersuchung der Hypothesen ist jedoch

relevant, wie stark die Entstehung von Fehlern durch eine hohe Anzahl von Änderungen versuchsacht wird. Mit Hilfe der hier verwendenten statistischen Analyseverfahren ist es allerdings nicht möglich, die Richtung eines Zusammenhangs anhand von Daten herzuleiten. Es kann nur festgestellt werden, inwieweit ein Zusammenhang zwischen zwei Größen besteht.

Deshalb ist es erwünscht, nur die Daten einer Untersuchung zu unterziehen, die den Einfluss der Änderungen auf die Fehler wiederspiegeln. Dabei wird die Tatsache vernachlässigt, dass Änderungen zur Behebung von Fehern ebenfalls zu neuen Fehlern führen können. Der Zusammenhang fällt dadurch etwas geringer aus als der tatsächliche Einfluss der Änderungen auf die Anzahl der Fehler. In dieser Studie wird jedoch vorgezogen, einen weniger bedeutenden Zusammenhang als einen Zusammenhang mit einer unklaren Richtung als Basis für Schlussfolgerungen zu verwenden, da man auf diese Weise das Risiko für falsche Entscheidungen verringert. Da die Änderungen zur Fehlerbehebung in den Daten als solche ohnehin zu erkennen sind, kann diese Einschränkung problemlos umgesetzt werden.

Um die Anzahl der Änderunge ohne die eben getroffene Einschränkung bestimmen zu können sind im Prinzip nur die Daten aus CVS erforderlich. Wegen der Einschränkung werden jedoch darüber hinaus Daten aus Bugzilla benötigt. Es versuchsacht jedoch keinen zusätzlichen Aufwand, da diese Daten für weitere benötigte Maße, $numFaults$ beispielsweise, erforderlich sind.

Bezeichnung: $numChanges(E)$

Wertebereich: N

Skala: absolut

numAuthors - Anzahl der Entwickler einer Entwurfseinheit

Dieses Maß gibt an, wie viele Entwickler diese Entwurfseinheit set deren Erstellung geändert haben. Diese Daten können aus CVS extrahiert werden.

Bezeichnung: $numAuthors(E)$

Wertebereich: N

Skala: absolut

isInCycle - Zugehörigkeit einer Entwurfseinheit zu einer Zyklengruppe

Dieses Maß gibt an, ob diese Entwurfseinheit in einer Zyklengruppe enthalten ist. Es kann also nur zwei Werte annehmen. Es ist eins, wenn die betrachtete Entwurfseinheit mindestens einen Nachbar in der zugehörigen Zyklengruppe hat, wenn also die Zyklengruppe nicht nur aus der Entwurfseinheit selbst besteht. Andernfalls ist dieser Wert null.

Bezeichnung: $isInCycle(E)$

Wertebereich: $0, 1$

Skala: nominal

cycleSize - Größe einer Zyklengruppe

Dieses Maß gibt für eine Zyklengruppe die Anzahl der Entwurfseinheiten an, die in dieser enthalten sind. Die dafür erforderlichen Daten können mit Hilfe eines Werkzeugs zur Strukturanalyse erhoben werden.

Bezeichnung: $cycleSize(E)$

Wertebereich: N

Skala: absolut

numBidirDeps - Anzahl der bidirektionalen Abhängigkeiten in einer Zyklengruppe

Dieses Maß gibt für eine Zyklengruppe die Anzahl der Paare von Entwurfseinheiten an, die voneinander abhängig sind. Sie wird mit Hilfe eines Werkzeugs zur Strukturanalyse gemessen.

Bezeichnung: $numBidirDeps(Z)$

Wertebereich: N

Skala: absolut

numDeps - Anzahl der Abhängigkeiten in einer Zyklengruppe

Dieses Maß gibt die Anzahl der Abhängigkeiten innerhalb einer Zyklengruppe an. Dabei werden bidirektionale Abhängigkeiten als zwei unterschiedliche Abhänggkeiten betrachtet. Diese Größe wird mit Hilfe eines Werkzeugs zur Strukturanalyse gemessen.

Bezeichnung: $numDeps(Z)$

Wertebereich: N

Skala: absolut

7.2 Datenbeschaffung

Für die Untersuchung der Hypothesen werden also Daten aus CVS und Bugzilla, sowie durch ein Werkzeug für die Strukturanalyse gesammelte Daten benötigt. Es wurden zunächst die Daten aus CVS und Bugzilla extrahiert. Dabei wurden alle vorhandenen Daten beschaffen, da zum einen der Aufwand für eine teilweise Beschaffung genauso hoch gewesen wäre, und zum anderen weil es auch die

Möglichkeit besteht, dass diese Daten sich im Rahmen der Analyse als nützlich erweisen. Der zusätzliche Bedarf an Speicherplatz war dabei unproblematisch.

Da der Prozess der Datenbeschaffung für die drei Datenquellen CVS, Bugzilla und Strukturanalyse sehr verschieden ist, wird dieser im weiteren in jeweils einem Abschnitt ausführlich beschrieben.

7.2.1 Kopieren der Log-Einträge aus CVS

Um alle zum Eclipse-SDK zugehörigen CVS-Logeinträge kopieren zu können, musste dieses zunächst ausgecheckt werden. Das SDK besteht aus mehreren Modulen, die einzeln ausgecheckt wurden. Dafür wurden die Namen aller dazugehörigen Module in einer Textdatei abgespeichert. Diese wurde von einem Bash-Skript, *createMysqlDatafiles.sh* (siehe Anhang), eingelesen. Das Skript hat alle in der Textdatei vorkommenden Module in jeweils unterschiedliche Verzeichnisse ausgecheckt und anschließend in jedem Verzeichnis den CVS-Befehl *cvs log* ausgeführt. Dieser Befehl gibt alle in CVS gespeicherten Logeinträge zu dem in dieses Verzeichnis ausgecheckten Modul zurück. Die Ausgabe dieses Befehls wurde in eine weitere Textdatei, cvs.log, gespeichert. Auf diese Weise wurden alle im CVS gespeicherten Informationen über die Dateien des SDK in einer Datei abgelegt.

7.2.2 Kopieren der Daten aus Bugzilla

Die Fehlerdaten wurden manuell von der Internetseite [Eclipse Team, 2006] heruntergeladen. Um auf die Fehlerdaten zugreifen zu können, war zunächst eine Registrierung erforderlich. Nach einer erfolgreichen Anmeldung waren alle Fehlerdaten der letzten fünf Jahre sowohl als eine HTML-Tabelle als auch eine trennzeichenbasierte Textdatei verfügbar. In diesem Fall wurde letztere Alternative gewählt. Die Bedienung von Bugzilla ist in [The Bugzilla Team, 2005] beschrieben.

Um alle erforderlichen Fehlerdaten herunterzuladen, mussten die drei Komponenten von Eclipse SDK (Platform, JDT und PDE) getrennt bearbeitet werden, da eine gleichzeitige Auswahl aller Komponenten nicht möglich war. Für die Komponente Platform mussten die Daten in mehreren Schritten heruntergeladen werden, da der Server die Anfrage sonst nicht bearbeiten konnte. Deshalb wurden die Daten für Zeitabstände von jeweils einem Jahr heruntergeladen.

7.2.3 Strukturanalyse

Um bestimmte Messwerte einer Entwurfseinheit, beispielsweise ihre Größe, zu bestimmen, musste die Release-Version 3.2.1 von Eclispe aus dem CVS ausgecheckt werden. Der Quelltext dieser Version wurde einer Strukturanalyse unterzogen. Während einige Strukturinformationen mit verhältnismäßig einfachen Werkzeugen beschaffen werden können, braucht man für andere, beispielsweise die Zyklengröße, ein Werkzeug, das über eine große Auswahl von Maßen verfügt oder einem die Möglichkeit gibt, neue Maße zu definieren.

In dieser Arbeit wurde das Werkzeug *Sotograph* verwendet, um die erofrderlichen Informationen über die Struktur des Quelltextes zu bekommen. Der *Sotograph* ist in [Software Tomography Ltd, 2004, Software Tomography Ltd, 2005] ausführlich beschrieben. Alle für diese Studien verwendeten Maße bis auf die Anzahl der bidirektionalen Abhängigkeiten waren vordefiniert. Für das eben genannte Maß wurde von Carola Lilienthal eine Sotograph-Anfrage zur Verfügung gestellt.

Einige der benötigten Maße waren nur auf der Klassenebene verfügbar. Deshalb mussten die Messwerte für Dateien mit mehreren Klassen nach dem Messen berechnet werden, indem die Werte aller Klassen addiert wurden.

Alle gemessenen Daten wurden als trennzeichenbasierte Textdateien sowie Excel-Dateien exportiert. Für diese Studie wurden folgende Daten im Sotograph erhoben:

numChars Größe der Dateien in Zeichen

numFilesInCycle Anzahl der Dateien in der zu einer Datei zugehörigen Zyklengruppe

numBidirRefs Anzahl der bidirektionalen Abhängigkeiten für Klassen, wird für die Berechnugn der Bidirektionalität benutzt

numDirNeighborsInCycle Anzahl direkter Nachbarn im Zyklus für Dateien, wird für die Brechnung der Abhängigkeitsdichte benutzt

7.3 Einparsen und Speichern der Daten

Nachdem alle Daten aus unterschiedlichen Quellen vollständig beschafft worden waren, lagen diese in Form von Textdateien vor. Das Format der Daten aus CVS, Bugzilla und einem Werkzeug zur Strukturanalyse war jeweils unterschiedlich. Der nächste Bearbeitungsschritt bestand darin, die Daten in eine Form umzuwandeln, die einen flexiblen und einfachen Zugriff erlaubt. Um eine hohe Flexibilität bei der Auswertung zu ermöglichen, wurden alle Daten in einer MySQL-Datenbank abgelegt.

Eine Speicherung der Daten in einer Datenbank macht den Zugriff auf diese Daten deutlich einfacher. So sind zur Berechnung einiger Maße sowohl Daten aus CVS als auch Daten aus Bugzilla sowie Daten über die Struktur des Quelltextes erforderlich. Eine derartige Berechnung wäre sehr aufwendig, wenn die Daten in unterschiedlichen Dateien mit unterschiedlichem Format abgelegt wären. Wenn alle Daten jedoch in einer Datenbank vorliegen, ist die Berechnung relativ einfach mit Hilfe von SQL-Anfragen möglich.

Zum Speichern der Daten wurde eine Reihe von Tabellen angelegt. So wurden zunächst eine Tabelle *files* zum Speichern der dateibezogenen Informationen, eine Tabelle *change_sets* für die Änderungen, eine Tabelle *bugs* für die Fehlereinträge, eine Tabelle *usages* für Abhängigkeiten zwischen den Dateien und eine Tabelle *bidirectionals* für bidirektionale Abhängigkeiten zwischen diesen angelegt. Nach der Verarbeitung wurden die zu untersuchenden Daten in

zwei Tabellen gespeichert: *files* und *cycles*. Diese Tabellen wurden als Grundlage für die Anwendung statistischer Verfahren herangezogen. Sie wurden in Form von Excel-Tabellen exportiert, um mit Hilfe von EViews analysiert werden zu können. Eine vollständige Beschreibung des Datenbankschemas befindet sich im Anhang.

Um die Daten in der Datenbank zu speichern, wurden diese zunächst aus den ursprünglichen Textdateien eingelesen. Es wurde eine Reihe von Hilfsskripten in Perl geschrieben, die dem Einlesen der Textdateien und deren Speicherung in der Datenbank dienen. Perl eignet sich zum Einparsen von Daten in unterschiedlichen Formaten sehr gut, da es über eine Reihe nützlicher Funktionen zur Arbeit mit regulären Ausdrücken verfügt. Die Skripte wurden so implementiert, dass eine Reihe von Einstellungen als Parameter übergeben werden können. So können beispielsweise die Namen der Tabellen und der Datenbank beim Aufruf des Skripts eingestellt werden. Alle Perl-Skripte befinden sich auf der beigefügten CD-ROM.

Die Daten von CVS und Bugzilla wurden in diesem Schritt anders behandelt als die durch eine Strukturanalyse ermittelten Messwerte, da das Format der Datenquellen im Fall von CVS und Bugzilla etwas komplizierter ist. Sie wurden in zwei Durchläufen bearbeitet. Zunächst wurden die CVS-Logdatei[1] sowie die Bugzilla-Dateien[2] eingelesen und in Form von SQL-Anweisungen als Textdateien gespeichert. Nach diesem Verarbeitungsschritt lagen Daten von CVS und Bugzilla im gleichen Format vor und konnten auf gleiche Art behandelt werden. Die Textdateien[3] mit den SQL-Anweisungen wurden in einem weiteren Schritt von einem SQL-Interpreter ausgeführt.

Die durch Strukturanalyse gewonnenen Daten lagen in einem tabellenartigen Format vor – in jeder Zeile war genau ein Datensatz enthalten. Deshalb konnten diese Daten direkt nach dem Einlesen in die Datenbank geschrieben werden[4].

7.4 Verlinkung der Fehlereinträge mit den Änderungen

Nachdem alle Daten in der Datenbank abgespeichert wurden, muss eine Verbindung zwischen den Änderungsvorgängen an den einzelnen Entwurfseinheiten und den Einträgen der Fehlerdatenbank hergestellt werden. Dazu werden die Änderungskommentare benutzt. Wenn ein Fehler beseitigt wird und die dazugehörige Änderung eingecheckt wird, muss die ID des beseitigten Fehlers im Änderungskommentar angegeben werden. Oft werden diese in Verbindung mit bestimmten Wörtern oder Zeichen eingetragen. So steht oft ein '#' vor einer Fehler-ID. Deshalb werden alle Kommentare nach einem bestimmten Muster durchsucht, um festzustellen, ob es sich dabei um einen Bugfix, also eine Änderung zur Fehlerbehebung handelt. Es wurde eine Reihe von Mustern formuliert,

[1] Dafür wurde das Skript cvslog2mysqlConverter.pl benutzt.
[2] Dafür wurde das Skript buglist2mysqlConverter.pl benutzt.
[3] Dafür wurde das Skript mysqlProcessor.pl benutzt.
[4] Dazu wurde das Skript parseSotographData.pl benutzt.

die eine Zahl im Kommentar als eine Fehler-ID beziehungsweise eine Nicht-ID identifizieren. Diese werden in der Tabelle *Muster zur Erkennung der Bug-IDs* aufgelistet.

Muster zur Erkennung der Bug-IDs

Musterbeschreibung	Wahrscheinlichkeitsstufe
eine Zahl ist von einem oder mehr Punkten umgeben	0
eine Zahl ist von einem oder mehr Buchstaben umgeben	0
eine Zahl ist von einem oder mehr Unterstrichen umgeben	0
comments mit merge	0
Zahlen 2000 bis 2000 von Leerzeichen umgeben oder vor WIN	0
eine ein-, zwei- oder dreistellige Zahl mit Leerzeichen	1
fünf- oder sechsstellige Zahl bis zur maximalen Bug ID mit Leerzeichen	2
comments mit patch	3
comments mit #	3
comments mit defect	3
comments mit fix	3
comments mit bug (aber nicht debug) ('bug%', '% bug%', '%.bug%'...)	3

Jedem Muster wird eine Wahrscheinlichkeitsstufe dafür zugeordnet, dass die damit verbundene Zahl tatsächlich eine ID ist. So ist die Wahrscheinlichkeit dafür höher, wenn vor einer Zahl ein '#' steht, als wenn die Zahl dreistellig ist und durch keine weiteren Merkmale auffällt. Die niedrigste Stufe deutet darauf hin, dass die betroffene Zahl mit einer sehr hohen Wahrscheinlichkeit keine ID ist. Die Muster wurden bei einer manuellen Untersuchung der Kommentare erkannt.

Die Bugfixes werden in einer weiteren Tabelle in der Datenbank abgelegt. Alle als Fehler-ID erkannten Zahlen werden zur Verlinkung der zugehörigen Fehler mit der bearbeiteten Änderung benutzt, indem in einer weiteren Tabelle, bugfixeslinks, ein ID-Paar gespeichert wird. Dieses Paar besteht aus der ID der Änderung und der Fehler-ID.

Eine derartige Verbindung ist jedoch nur für bereits behobene Fehler möglich, da man den Fehler sonst keiner Entwurfseinheit zuordnen kann. Es wird vereinfachend angenommen, dass die Fehler zum Zeitpunkt der Untersuchung vollständig behoben wurden, wenn es einen Änderungsvorgang mit der entsprechenden Fehler-ID gibt. Teilweise behobene Fehler können als solche nicht auf eine sinnvolle Weise berücksichtigt werden, da die dafür benötigten Daten nicht vorliegen. Es werden nur Entwurfseinheiten betrachtet, die sowohl zum Anfangszeitpunkt als auch zum End-Zeitpunkt des untersuchten Zeitraums im Entwicklungsprojekt vorhanden waren, da sonst keine Messung möglich ist.

7.5 Import der Daten in EViews

Zum Schluss liegen alle Daten in der Datenbank vor und können beliebig umstrukturiert werden. Für eine statistische Analyse werden diese anschließend als eine trennzeichenbasierte Textdatei aus der Datenbank exportiert und in EViews

importiert. EViews ist ein Werkzeug für statistische Analysen und Prognosen. Anhand der vorliegenden Messwerte wurden die abzuleitenden Messwerte mit Hilfe von SQL-Anfragen berechnet. Die dazu benutzten Anfragen sind in den einzelnen Perl-Skripten auf der beigefügten CD-ROM zu finden. Der Export der Daten wurde über die Web-Schnittstelle PHPMyAdmin durchgeführt. Wie Daten aus einer trennzeichenbasierten Datei in EViews importiert werden können, wird in [Quantitative Micro Software, 2002] beschrieben.

Kapitel 8

Untersuchung der Hypothesen

Für die Analyse aller Hypothesen außer der zweiten kann sowohl Korrrelation als auch Parametervergleich herangezogen werden. Bei der Untersuchung der zweiten Hypothese kann die Korrelation nicht berechnet werden, da dem Maß *isInCycles* ein nominaler Skalentyp zu Grunde liegt. Die Korrelation ist jedoch erst ab dem Intervallniveau zulässig.

Im folgenden werden die Hypothesen jeweils mit Hilfe des Verfahrens Parametervergleich überprüft und wenn eine deutliche Tendenz zu einer höheren Fehleranfälligkeit zu erkennen ist, werden die Korrelationskoeffizienten berechnet. Wie im Laufe der Analyse deutlich wird, sind jedoch die untersuchten Indikatoren nicht die wichtigsten für die Fehleranfälligkeit, auch wenn eine deutliche Tendenz zu einer höheren Fehleranfälligkeit vorliegt.

Für eine Untersuchung der Hypothesen mit Hilfe des Parametervergleichs müssen die Entwurfseinheiten in zwei Gruppen eingeteilt werden. Während die Aufteilung im Falle der zweiten Hypothese nur auf die eine Weise möglich ist, muss für die Untersuchung der übrigen Hypothesen entschieden werden, wo die Grenze zwischen den geringen und hohen Werten verlaufen soll. Die Aussagekraft der Untersuchungsergebnisse ist um so höher, je gleichmäßiger diese aufgeteilt werden. Aus diesem Grund wird hier der Median als Grenzwert für die Aufteilung benutzt.

Der Parametervergleich wurde automatisch durch das von mir in Perl geschriebene Skript *parameterVergleich.pl* (siehe CD-ROM) vorgenommen. Fr die Korrelationsanalyse wurde das statistische Werkzeug *EViews* verwendet.

8.1 Hypothese 1

Die erste Hypothese wird mit Hilfe eines Parametervergleichs und einer Korrelationsanalyse (siehe Kapitel fünf) untersucht. Sie lautet:

Hypothese 1: Entwurfseinheiten mit hohem Ausmaß an praktiziertem Collective Code Ownership weisen eine Tendenz zu einer höheren Fehleranfälligkeit auf als die Entwurfseinheiten mit einem geringen Ausmaß an praktiziertem Collective Code Ownership

Um diese mit Hilfe des Parametervergleichs zu überprüfen, werden die Entwurfseinheiten in zwei Gruppen D und $\bar{D}$ unterteilt. D ist die Gruppe mit den höheren Werten von $relNumAuthors$ und $\bar{D}$ die Gruppe mit den geringeren Werten. Die Berechnung des Wertes, der für diese Aufteilung benutzt wird, ist in Kapitel fünf beschrieben. In diesem Fall wird der Median des Maßes $relNumAuthors$ benutzt. Die für das Maß $relNumFaults$ in diesen Gruppen berechneten Erwartungswerte, Varianzen s^2 sowie der Stichprobenumfang n sind in der nachfolgenden Tabelle gegeben.

	$\bar{D}$	D
μ	5.03000439627301e-005	0.000527082842942674
s^2	1.04928802290816e-008	2.37848747847423e-007
n	6680	6681

Die zu überprüfende Hypothese wird bei dieser Betrachtung als die Alternativhypothese H1 eingesetzt. Sie ist richtig, wenn die komplementäre Hypothese H0 mit einer geringen Irrtumswahrscheinlichkeit abgelehnt wird. Die Hypothesen sind wie folgt formuliert:

$$H0: \ \mu_{\bar{D}} - \mu_D \geq 0$$
$$H1: \ \mu_{\bar{D}} - \mu_D < 0$$

Die Gesamtvarianz s^2_{ges} der beiden Stichproben ist:

$$s^2_{ges} = \frac{1}{n_D + n_{\bar{D}} - 2} \cdot ((n_D - 1) \cdot s^2_D + (n_{\bar{D}} - 1) \cdot s^2_{\bar{D}}) = 1.2417e - 007$$

Aus diesen Daten wird nun ein Testwert t berechnet, der für die Testentscheidung benötigt wird:

$$t = \frac{\mu_D - \mu_{\bar{D}}}{\sqrt{s^2_{ges}}} \cdot \sqrt{\frac{n_D \cdot n_{\bar{D}}}{n_D + n_{\bar{D}}}} = 78.2017$$

Für diese Berechnung ergibt der Wert des $t_{0,99}$-Quantils als 2.326. Damit liegt der Testwert t innerhalb des Ablehnungsbereiches:

$$78.2017 > 2.326 \Rightarrow t > t_{o,99} \Rightarrow \textbf{H0 abgelehnt}.$$

Die Hypothese H0 kann also abgelehnt und H1 angenommen werden. Es konnte statistisch bestätigt werden, dass Entwurfseinheiten mit hohem Ausmaß an praktiziertem Collective Code Ownership eine Tendenz zu einer höheren Fehleranfälligkeit aufweisen als die Entwurfseinheiten mit einem geringen Ausmaß an praktiziertem Collective Code Ownership.

Anschließend wird mit Hilfe einer Korrelationsanalyse untersucht, wie ausschlaggebend der Indikator $relNumAuthors$ des praktizierten Collective Code Ownership für die Fehleranfälligkeit ist im Vergleich zu anderen, hier nicht betrachteten Faktoren. Es wurde ein Korrelationskoeffizient in Höhe von $r = 0.3$ berechnet. Der Testwert dafür ist:

$t = \frac{r}{\sqrt{1-r^2}} \cdot \sqrt{n-2} = 11.4690 > 2.326$

Deshalb ist der Koeffizient signifikant. Die Analyse ergibt somit einen positiven, satistisch signifikanten Zusammenhang zwischen $relNumAuthors$ und dem Indikator $relNumFaults$ für Fehleranfälligkeit. Dieser ist aber zu schwach, um für sichere Aussagen benutzt zu werden.

8.2 Hypothese 2

Hypothese 2: Entwurfseinheiten in Zyklen weisen eine Tendenz zu einer höheren Fehleranfälligkeit auf als die Entwurfseinheiten außerhalb von Zyklen

Zur Untersuchung dieser Hypothese wird ein Parametervergleich (siehe Kapitel fünf) durchgeführt. Dafür werden die Entwurfseinheiten in zwei Gruppen D und $\bar{D}$ unterteilt. D ist die Gruppe von Entwurfseinheiten mit $isInCycle = 1$ und $\bar{D}$ die Gruppe mit $isInCycle = 0$. Die für das Maß $relNumFaults$ in diesen Gruppen berechneten Erwartungswerte, Varianzen s^2 sowie der Stichprobenumfang n sind in der nachfolgenden Tabelle gegeben.

	$\bar{D}$	D
μ	0.000416348777340606	0.000231683053035076
s^2	1.3356620303832e-005	5.9791436260994e-006
n	4125	9237

Die zu überprüfende Hypothese wird bei dieser Betrachtung als die Alternativhypothese H1 eingesetzt. Sie ist richtig, wenn die komplementäre Hypothese H0 mit einer geringen Irrtumswahrscheinlichkeit abgelehnt wird. Die Hypothesen sind wie folgt formuliert:

$$H0: \ \mu_{\bar{D}} - \mu_D \geq 0$$
$$H1: \ \mu_{\bar{D}} - \mu_D < 0$$

Die Gesamtvarianz s^2_{ges} der beiden Stichproben ist:

$$s^2_{ges} = \frac{1}{n_D + n_{\bar{D}} - 2} \cdot ((n_D - 1) \cdot s^2_D + (n_{\bar{D}} - 1) \cdot s^2_{\bar{D}}) = 8.2564e-006$$

Aus diesen Daten wird nun ein Testwert t berechnet, der für die Testentscheidung benötigt wird:

$$t = \frac{\mu_D - \mu_{\bar{D}} - d}{\sqrt{s^2_{ges}}} \cdot \sqrt{\frac{n_D \cdot n_{\bar{D}}}{n_D + n_{\bar{D}}}} = -3.4318$$

Für diese Berechnung ergibt der Wert des $t_{0,99}$-Quantils 2.326. Damit liegt der Testwert t innerhalb des Ablehnungsbereiches:

$-3.4318 < 2.326 \Rightarrow t < t_{o,99} \Rightarrow$**H0 nicht abgelehnt**.

Die Hypothese $H0$ kann nicht abgelehnt und $H1$ deshalb nicht angenommen werden. Es konnte also nicht statistisch bestätigt werden, dass Entwurfseinheiten in Zyklen eine Tendenz zu höherer Fehleranfälligkeit aufweisen als die Entwurfseinheiten außerhalb von Zyklen.

8.3 Hypothese 3

Hypothese 3: Entwurfseinheiten in größeren Zyklengruppen weisen eine Tendenz zu einer höheren Fehleranfälligkeit auf als die Entwurfseinheiten in kleineren Zyklengruppen

Zur Untersuchung dieser Hypothese werden Parametervergleich und Korrelationsanalyse (siehe Kapitel fünf) durchgeführt. Um diese mit Hilfe des Parametervergleichs zu überprüfen, werden die Zyklengruppen in zwei Gruppen D und $\bar{D}$ unterteilt. D ist die Gruppe mit den höheren Werten von $cycleSize$ und $\bar{D}$ die Gruppe mit den geringeren Werten. Die Berechnung des Wertes, der für diese Aufteilung benutzt wird, ist in Kapitel fünf beschrieben. In diesem Fall wird der Median des Maßes $cycleSize$ benutzt. Die für das Maß $relNumFaults$ in diesen Gruppen berechneten Erwartungswerte, Varianzen s^2 sowie der Stichprobenumfang n sind in der nachfolgenden Tabelle gegeben.

	$\bar{D}$	D
μ	0.000252622384379562	0.000193170928066038
s^2	4.94816946099899e-007	6.4551740794957e-007
n	120	121

Die zu überprüfende Hypothese wird bei dieser Betrachtung als die Alternativhypothese H1 eingesetzt. Sie ist richtig, wenn die komplementäre Hypothese H0 mit einer geringen Irrtumswahrscheinlichkeit abgelehnt wird. Die Hypothesen sind wie folgt formuliert:

$$H0: \ \mu_{\bar{D}} - \mu_D \geq 0$$
$$H1: \ \mu_{\bar{D}} - \mu_D < 0$$

Die Gesamtvarianz s^2_{ges} der beiden Stichproben ist:

$$s^2_{ges} = \frac{1}{n_D + n_{\bar{D}} - 2} \cdot ((n_D - 1) \cdot {s_D}^2 + (n_{\bar{D}} - 1) \cdot {s_{\bar{D}}}^2) = 5.6047e - 007$$

Aus diesen Daten wird nun ein Testwert t berechnet, der für die Testentscheidung benötigt wird:

$$t = \frac{\mu_D - \mu_{\bar{D}} - d}{\sqrt{s^2_{ges}}} \cdot \sqrt{\frac{n_D \cdot n_{\bar{D}}}{n_D + n_{\bar{D}}}} = -0.6138$$

Für diese Berechnung ergibt der Wert des $t_{0,99}$-Quantils 2.345. Damit liegt der Testwert t innerhalb des Ablehnungsbereiches:

$$-0.6138 < 2.345 \Rightarrow t < t_{o,99} \Rightarrow \textbf{H0 nicht abgelehnt}.$$

Die Hypothese $H0$ kann nicht abgelehnt und $H1$ deshalb nicht angenommen werden. Es konnte also nicht statistisch bestätigt werden, dass Entwurfseinheiten in größeren Zyklengruppen eine Tendenz zu höherer Fehleranfälligkeit aufweisen als die Entwurfseinheiten in kleineren Zyklengruppen.

8.4 Hypothese 4

Hypothese 4: Entwurfseinheiten in stärker verflochtenen Zyklengruppen weisen eine Tendenz zu einer höheren Fehleranfälligkeit auf als die Entwurfseinheiten in weniger verflochtenen Zyklengruppen

Zur Untersuchung dieser Hypothese werden Parametervergleich und Korrelationsanalyse (siehe Kapitel fünf) durchgeführt.

8.4.1 Bidirektionalität

Um diese mit Hilfe des Parametervergleichs zu überprüfen, werden die Zyklengruppen in zwei Gruppen D und $\bar{D}$ unterteilt. D ist die Gruppe mit den höheren Werten von *bidirDensity* und $\bar{D}$ die Gruppe mit den geringeren Werten. Die Berechnung des Wertes, der für diese Aufteilung benutzt wird, ist in Kapitel 5 beschrieben. In diesem Fall wird der Median des Maßes *bidirDensity* benutzt. Die für das Maß *relNumFaults* in diesen Gruppen berechneten Erwartungswerte, Varianzen s^2 sowie der Stichprobenumfang n sind in der nachfolgenden Tabelle gegeben.

	$\bar{D}$	D
μ	0.0002629901575	0.000184926235044248
s^2	5.21198840117079e-007	6.07382743336574e-007
n	120	121

Die zu überprüfende Hypothese wird bei dieser Betrachtung als die Alternativhypothese H1 eingesetzt. Sie ist richtig, wenn die komplementäre Hypothese H0 mit einer geringen Irrtumswahrscheinlichkeit abgelehnt wird. Die Hypothesen sind wie folgt formuliert:

$$H0: \ \mu_{\bar{D}} - \mu_D \geq 0$$
$$H1: \ \mu_{\bar{D}} - \mu_D < 0$$

Die Gesamtvarianz s^2_{ges} der beiden Stichproben ist:

$$s^2_{ges} = \frac{1}{n_D + n_{\bar{D}} - 2} \cdot ((n_D - 1) \cdot {s_D}^2 + (n_{\bar{D}} - 1) \cdot {s_{\bar{D}}}^2) = 5.6125e - 007$$

Aus diesen Daten wird nun ein Testwert t berechnet, der für die Testentscheidung benötigt wird:

$$t = \frac{\mu_D - \mu_{\bar{D}} - d}{\sqrt{s^2_{ges}}} \cdot \sqrt{\frac{n_D \cdot n_{\bar{D}}}{n_D + n_{\bar{D}}}} = -0.8101$$

Für diese Berechnung ergibt der Wert des $t_{0,99}$-Quantils 2.345. Damit liegt der Testwert t innerhalb des Ablehnungsbereiches:

$$-0.8101 < 2.345 \Rightarrow t < t_{o,99} \Rightarrow \textbf{H0 nicht abgelehnt}.$$

Die Hypothese $H0$ kann nicht abgelehnt und $H1$ deshalb nicht angenommen werden. Es konnte also nicht statistisch bestätigt werden, dass Entwurfseinheiten in stärter verflochtenen Zyklengruppen in Sinne der Bidirektionalität eine Tendenz zu höherer Fehleranfälligkeit aufweisen als die Entwurfseinheiten in weniger verflochtenen Zyklengruppen.

8.4.2 Abhängigkeitsdichte

Um diese mit Hilfe des Parametervergleichs zu überprüfen, werden die Zyklengruppen in zwei Gruppen D und $\bar{D}$ unterteilt. D ist die Gruppe mit den höheren Werten von *depDensity* und $\bar{D}$ die Gruppe mit den geringeren Werten. Die Berechnung des Wertes, der für diese Aufteilung benutzt wird, ist in Kapitel fünf beschrieben. In diesem Fall wird der Median des Maßes *depDensity* benutzt. Die für das Maß *relNumFaults* in diesen Gruppen berechneten Erwartungswerte, Varianzen s^2 sowie der Stichprobenumfang n sind in der nachfolgenden Tabelle gegeben.

	$\bar{D}$	D
μ	0.000269430634959016	0.000183593781570248
s^2	5.36792831475809e-007	5.48756869894453e-007
n	120	121

Die zu überprüfende Hypothese wird bei dieser Betrachtung als die Alternativhypothese H1 eingesetzt. Sie ist richtig, wenn die komplementäre Hypothese H0 mit einer geringen Irrtumswahrscheinlichkeit abgelehnt wird. Die Hypothesen sind wie folgt formuliert:

$$H0: \ \mu_{\bar{D}} - \mu_D \geq 0$$
$$H1: \ \mu_{\bar{D}} - \mu_D < 0$$

Die Gesamtvarianz s^2_{ges} der beiden Stichproben ist:

$$s^2_{ges} = \frac{1}{n_D + n_{\bar{D}} - 2} \cdot ((n_D - 1) \cdot {s_D}^2 + (n_{\bar{D}} - 1) \cdot {s_{\bar{D}}}^2) = 5.4275e - 007$$

Aus diesen Daten wird nun ein Testwert t berechnet, der für die Testentscheidung benötigt wird:

$$t = \frac{\mu_D - \mu_{\bar{D}} - d}{\sqrt{s^2_{ges}}} \cdot \sqrt{\frac{n_D \cdot n_{\bar{D}}}{n_D + n_{\bar{D}}}} = -0.9081$$

Für diese Berechnung ergibt der Wert des $t_{0,99}$-Quantils 2.345. Damit liegt der Testwert t innerhalb des Ablehnungsbereiches:

$$-0.9081 < 2.345 \Rightarrow t < t_{o,99} \Rightarrow \textbf{H0 nicht abgelehnt}.$$

Die Hypothese $H0$ kann nicht abgelehnt und $H1$ deshalb nicht angenommen werden. Es konnte also nicht statistisch bestätigt werden, dass Entwurfseinheiten in stärter verflochtenen Zyklengruppen in Sinne von Anhängigkeitsdichte eine Tendenz zu höherer Fehleranfälligkeit aufweisen als die Entwurfseinheiten in weniger verflochtenen Zyklengruppen.

Kapitel 9

Kritische Betrachtung der Ergebnisse

Im Verlauf der Untersuchung waren einige vereinfachende Annahmen erforderlich, um anhand der vorliegenden Daten die Hypothesen zu untersuchen. Diese werden an dieser Stelle nochmal aufgezählt:

- Fehler sind alle gleich schwer zu beheben
- Fehler treten dort auf wo sie behoben werden
- Richtung der Abhängigkeit entspricht der vermuteten Richtung
- Zeitliche Veräderung der Messwerte kann vernachlässigt werden
- Die verwendeten Skripte sind korrekt
- Auch im Falle gut fundierter Definitionen der Maße kann immer noch ein Faktor existieren, der zu einer Scheinkorrelation führt.
- Dabei wird für den Umfang des zu analysierenden Quelltextes angenommen, dass er von dem Umfang der zu ändernden Entwurfseinheit abhängt.
- Als ein Maß für $Q(E)$ wird in dieser Arbeit die Anzahl $numFaults(E)$ der in dieser Version der Entwurfseinheit enthaltenen Fehler verwendet.
- Es wird dabei vernachlässigt, dass Kommentare auch gelesen und gepflegt werden müssen.

Darüber hinaus wurdden folgende Sachverhalte nicht berücksichtigt.

Unberücksichtigte Veränderung der Messwerte

In allen Studien, die quelltextbezogene Maße benutzen, werden diese zum Zeitpunkt eines Release ermittelt. Hierbei muss hinterfragt werden, ob die betrachteten Fehler tatsächlich unter Bedingungen entstanden sind, die durch die ermittelten Messwerte charakterisiert werden. Eine Entwurfseinheit kann zum Zeitpunkt des Release beispielsweise eine hohe zyklomatische Komplexität aufweisen. Zum Zeitpunkt der Fehlerentstehung kann diese jedoch noch niedrig gewesen sein. So ist, beispielsweise, für eins der in [Nagappan et al., 2006] untersuchten Projekte bekannt, dass dieses System einem gründlichen Refactoring unterzogen wurde unter Berücksichtigung der gängigen Komplexitätsmaße. In diesem Fall ist es offensichtlich nicht sinnvoll, die Summe der seit dem ersten Release gefundenen Fehler in Bezug zu den neuen Messwerten zu setzen. Wenn es um einen längeren Entwicklungszeitraum geht, wie in [Ostrand et al., 2005], wird in einigen Studien ([Ostrand et al., 2005],[Schröter et al., 2006a]) für einzelne Release-Versionen eine gesonderte Messung vorgenommen, so dass die seit dem Release-Zeitpunkt gefundenen Fehler mit den Messwerten in Verbindung gesetzt werden. Eine Verringerung des betrachteten Zeitraums mildert zwar das Problem, da die Wahrscheinlichkeit für einen stark veränderten Messwert sinkt. Eine exakte Lösung wäre es, die Messwerte zu dem Entstehungszeitpunkt des Fehlers zu messen, davon den Mittelwert für einen bestimmten Zeitraum zu bilden und diesen dann mit der Anzahl der Fehler in der betroffenen Entwurfseinheit in Verbindung zu setzen. Eine eindeutige Bestimmung des Entstehungszeitpunktes ist jedoch schwierig und aufwendig, da Fehler lange Zeit unentdeckt im System verbleiben können. Schließlich ist die Annahme, die in einer Release-Version gefundenen Fehler seien auch tatsächlich in dieser Version entstanden, auch nicht immer korrekt. Allerdings haben einige Studien ([?],[?],[?]) gezeigt, dass intensiv benutzter Software die meisten Fehler in geänderten Entwurfseinheiten entdeckt werden. Dieses Ergebnis bestätigt die intuitive Vorstellung, dass bei einer hohen Anzahl der Benutzer der Umgang mit der Software sehr vielfältig ist und dadurch die meisten Fehler schnell entdeckt werden. Im Falle eines intensiv benutzten Softwareprodukts kann man deshalb für nur einmal geänderte Entwurfseinheiten die Wahrscheinlichkeit dafür als gering schätzen, dass ein Fehler nicht bei der letzten Änderung entstanden ist. In diesen Fall wäre der Zeitpunkt der letzten Änderung deshalb eine gute Annäherung an den tatsächlichen Wert. Für Entwurfseinheiten, die mehrmals geändert wurden, müsste jedoch ein Mittelwert der Messwerte gebildet werden, da man nicht beurteilen kann, welche der Änderungen den der Fehler verursacht hat.

9.1 Gültigkeit von Maßen

Kausalitätsproblem

Es wurde ein Meilenstein verwendet und alle Dateine, die am Ende vorhanden waren. Die Strukturdaten wurde nur für die neuste Version (3.2.1) erhoben. Es stellt sich die Frage, inwieweit man dadurch weniger bedeutende Ergebnis-

se erzielt als wenn der Messprozess genauer an dem Zurechenbarkeitsprinzip ausgerichtet wird.

Modellierung des Aufwands

Die Schwierigkeit bei der Modellierung des Aufwandes besteht in der gleichzeitigen Berücksichtigung beider Arten von Änderungen, da nicht klar ist, in welchen Einheiten deren Aufwand gemessen wird. Da es jedoch ohnehin nicht möglich ist, den Umfang der von einer Änderunung erforderlichen Analysearbeit präzise zu schätzen, stellt dieser Sachverhalt ein vernachlässigbares Probelm dar. Es werden nur zwei arten unterschieden, da eine Unterscheidung zwischen weiteren Arten von Änderungen nicht zu einer höheren Präzision beitragen aufgrund von erforderlichen Vereinfachungen bei den späteren Betrachtungen.

Mangelnde Informationen über den Entwicklungsprozess

Ein weiteres Problem besteht darin, dass zu wenig Informationen über das Vorgehen bei der Entwicklung von Eclipse vorliegen. Ludwich und Lichter haben in [Ludewig and Lichter, 2007] das folgende Problem bei der Validierung von Thesen genannt. Um das Experiment reproduzierbar zu machen, müssen die zu untersuchenden Fragen sehr präzise formuliert werden. Die Bedingungen, unter deren das Experiment durchgeführt wurde, müssen mit vielen Einzelheiten aufgezeichnet werden. So würde es nicht ausreichen, die Anzahl der an einem Tag geschriebenen Codezeilen aufzuzeichnen. Es müsste auch beschrieben werden, welche Art von Code mit welchen Vorkenntnissen, in welcher Programmiersprache und unter welchen Arbeitsbedingungen beschrieben wurde. Eine genaue Aufzeichnung aller Einzelheiten des Vorgangs ist problematisch.

Mangelnde Eignung der Daten

Bei einer Beobachtung werden Daten aus bereits durchgeführten Vorgängen genutzt, um bestimmte Zusammenhänge zu überprüfen. Dabei fallen zwar keine zusätzlichen Kosten für die Durchführung der untersuchten Tätigkeiten an, es ist jedoch auch nicht möglich, die Bedingungen der Durchführung nachträglich zu beeinflussen. Man hat nur die Möglichkeit, sich zwischen unterschiedlichen bereits durchgeführten Projekten zu entscheiden. Das zu untersuchende Projekt muss so gewählt werden, dass es möglichst repräsentativ für die zu validierende These ist. Es ist jedoch problematisch, Daten im gewünschten Umfang und von gewünschter Qualität zu finden, um gültige Aussagen über die Validität einer These treffen zu können.

9.2 Interpretationsproblem

Korrelationskoeffizienten lassen sich schlecht interpretieren, da nicht klar ist, wie die Stärke des Zusammenhangs für einen bestimmten Koeffizienten konkret

aussieht. Es können nur sehr grobe Vorstellungen entstehen. Eine Visualisierung hilft jedoch dabei, sich die Bedeutung eines Koeffizienten vorzustellen.

Kapitel 10

Zusammenfassung

Diese Arbeit hatte die Zielsetzung, die Prinzipien Collective Code Ownership und Zyklenfreiheit im Hinblick darauf zu bewerten, wie sich diese auf der Fehleranfälligkeit eines Systems auswirken.

An Daten von Eclipse wurden vier Hypothesen untersucht. Dabei konnte mit Hilfe eines Parametervergleichs statistisch bestätigt werden, dass Entwurfseinheiten mit hohem Ausmaß an praktiziertem Collective Code Ownership eine Tendenz zu einer höheren Fehleranfälligkeit aufweisen als die Entwurfseinheiten mit einem geringen Ausmaß an praktiziertem Collective Code Ownership. Eine anschließende Korrelationsanalyse hat jedoch gezeigt, dass der Zusammenhang zwischen der Fehleranfälligkeit und dem praktizierten Collective Code Ownership verhältnismäßig schwach ist.

Weiter wurde untersucht, ob

- Entwurfseinheiten in Zyklen eine Tendenz zu einer höheren Fehleranfälligkeit aufweisen als die Entwurfseinheiten außerhalb von Zyklen,
- Entwurfseinheiten in größeren oder stärker verflochtenen Zyklengruppen eine Tendenz zu einer höheren Fehleranfälligkeit aufweisen als die Entwurfseinheiten in kleineren oder weniger verflochtenen Zyklengruppen.

Es konnte jedoch keine dieser Vermutungen durch einen Parametervergleich statistisch bestätigt werden.

Die Analyseergebnisse werden im Hinblick auf deren Gültigkeit kritisch geprüft. Dabei wird auf die aufgetretenen technischen Probleme sowie das Problem der Interpretation eingegangen. Da diese Probleme in allen Untersuchungen mehr oder weniger vorliegen, soll der Leser für die Gefahr einer Fehlinterpretation von Ergebnissen derartiger Studien sensibilisiert werden.

Literaturverzeichnis

[Auer and Miller, 2002] Auer, K. and Miller, R. (2002). *Extreme Programming Applied - Playing To Win*. Addison-Wesley Longman Publishing Co., Boston, MA, USA.

[Barnes, 2004] Barnes, N. (2004). Bugzilla database schema. Technical report, Ravenbrook Limited.

[Basili et al., 1996] Basili, V. R., Briand, L. C., and Melo, W. L. (1996). A validation of object-oriented design metrics as quality indicators. *IEEE Trans. Softw. Eng.*, 22(10):751–761.

[Basili and Weiss, 1984] Basili, V. R. and Weiss, D. M. (1984). A methodology for collecting valid software engineering data. *IEEE Trans. Software Eng.*, 10(6):728–738.

[Beck and Andres, 2004] Beck, K. and Andres, C. (2004). *Extreme Programming Explained*. Addison-Wesley Professional.

[Beck and Fowler, 2000] Beck, K. and Fowler, M. (2000). *Planning Extreme Programming*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

[Bennicke and Rust, 2005] Bennicke, M. and Rust, H. (2005). Messen im software-engineering und metrikbasierte qualitätsanalyse. Technical Report VISEK/024/D/1.2, Virtuelles Software Engineering Kompetenz Center.

[Bosch, 1990] Bosch, K. (1990). *Statistik für Nichtstatistiker*. R. Oldenbourg Verlag, München.

[Denaro et al., 2002] Denaro, G., Morasca, S., and Pezzè, M. (2002). Deriving models of software fault-proneness. In *SEKE '02: Proceedings of the 14th international conference on Software engineering and knowledge engineering*, pages 361–368, New York, NY, USA. ACM Press.

[Eclipse Team, 2006] Eclipse Team (2006). Bugzilla-system von eclipse. URL.

[Fenton and Neil, 1999] Fenton, N. E. and Neil, M. (1999). A critique of software defect prediction models. *IEEE Trans. Softw. Eng.*, 25(5):675–689.

[Fenton and Neil, 2000] Fenton, N. E. and Neil, M. (2000). Software metrics: roadmap. In *ICSE '00: Proceedings of the Conference on The Future of Software Engineering*, pages 357–370, New York, NY, USA. ACM Press.

[Fenton and Pfleeger, 1998] Fenton, N. E. and Pfleeger, S. L. (1998). *Software Metrics: A Rigorous and Practical Approach, Revised*. Course Technology.

[Fischer et al., 2003] Fischer, M., Pinzger, M., and Gall, H. (2003). Analyzing and relating bug report data for feature tracking. *wcre*, 0:90.

[Fowler, 2006] Fowler, M. (2006). http://www.martinfowler.com/bliki/codeownership.html. URL.

[Hartung et al., 1993] Hartung, J., Elpelt, B., and Klösener, K.-H. (1993). *Statistik*. Oldenburg, München, Wien.

[Henderson-Sellers, 1996] Henderson-Sellers, B. (1996). *Object-oriented metrics: measures of complexity*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA.

[Kan, 2002] Kan, S. H. (2002). *Metrics and Models in Software Quality Engineering*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

[Khoshgoftaar and Allen, 2003] Khoshgoftaar, T. M. and Allen, E. B. (2003). Ordering fault-prone software modules. *Software Quality Control*, 11(1):19–37.

[Leonhart, 2004] Leonhart, R. (2004). *Lehrbuch Statistik - Einstieg und Vertiefung*. Verlag Hans Huber, Bern.

[Ludewig and Lichter, 2007] Ludewig, J. and Lichter, H. (2007). *Software Engineering*. dpunkt.Verlag, Heidelberg.

[Nagappan and Ball, 2006] Nagappan, N. and Ball, T. (2006). Explaining failures using software dependences and churn metrics. Technical Report MSR-TR-2006-03, Microsoft Research.

[Nagappan et al., 2006] Nagappan, N., Ball, T., and Zeller, A. (2006). Mining metrics to predict component failures. In Osterweil, L. J., Rombach, H. D., and Soffa, M. L., editors, *ICSE*, pages 452–461. ACM.

[Neumann, 2005] Neumann, R. (2005). Identifikation von softwarequalitätsproblemen anhand historischer daten des Änderungsmanagements. Diplomarbeit, Brandenburgische Technische Universität Cottbus.

[Object Technology International, 2003] Object Technology International, I. (2003). Eclipse platform technical overview. white paper.

[Open Source Competence Group (OSCG), 2006] Open Source Competence Group (OSCG) (2006). http://www.oscg.ch/rsc/tools/idts/eclipse.html.

[Ostrand et al., 2005] Ostrand, T. J., Weyuker, E. J., and Bell, R. M. (2005). Predicting the location and number of faults in large software systems. *IEEE Transactions on Software Engineering*, 31(4):340–355.

[Quantitative Micro Software, 2002] Quantitative Micro Software (2002). *EViews User's Guide 4.0*. Quantitative Micro Software, LLC, United States of America.

[Rönz, 2007] Rönz, B. (2007). Computergestützte statisitk i. internet.

[Schröter et al., 2006a] Schröter, A., Zimmermann, T., Premraj, R., and Zeller, A. (2006a). If your bug database could talk... (short paper). In *Proceedings of the 5th International Symposium on Empirical Software Engineering. Volume II: Short Papers and Posters*, pages 18–20.

[Schröter et al., 2006b] Schröter, A., Zimmermann, T., and Zeller, A. (2006b). Predicting component failures at design time. In *Proceedings of the 5th International Symposium on Empirical Software Engineering*, pages 18–27.

[Software Tomography Ltd, 2004] Software Tomography Ltd (2004). *Dokumentation zu Sotograph*. Software Tomography Ltd.

[Software Tomography Ltd, 2005] Software Tomography Ltd (2005). *Dokumentation zu Sotograph*. Software Tomography Ltd.

[The Bugzilla Team, 2005] The Bugzilla Team (2005). *The Bugzilla Guide - 2.18 Release*.

[und H. Züllighoven, 2002] und H. Züllighoven, C. F. (2002). *Informaik-Handbuch*, chapter Softwaretechnik, pages 641–667. Hanser Verlag, München, 3. auflage edition.

[Vesperman, 2003] Vesperman, J. (2003). *Essential CVS*. O'Reilly Media, Inc.

[Zimmermann and Nagappan, 2006] Zimmermann, T. and Nagappan, N. (2006). Predicting subsystem failures using dependency graph complexities. Technical report, Microsoft Research.

[Zimmermann and Weißgerber, 2004] Zimmermann, T. and Weißgerber, P. (2004). Preprocessing cvs data for fine-grained analysis. In *Proceedings of the First International Workshop on Mining Software Repositories*, pages 2–6.

Anhang A

Inhaltsverzeichnis der CD-ROM

1. Index
2. Text der Diplomarbeit
3. Vortragsfolien
4. Literaturquellen
5. Perl-Quelltext
6. Bash-Skripte
7. Daten

Anhang B

Erstellte SQL-Schemata

B.1 cvsfiles

- fileid int(10)
- filepath varchar(255)
- filename varchar(255)
- head varchar(20)
- branch varchar(20)
- totalrevisions smallint
- description text

B.2 bugs

- bug_id int(10)
- opendate datetime
- changeddate datetime
- bug_severity varchar(20)
- priority varchar(10)
- rep_platform text
- assigned_to text
- assigned_to_realname text
- reporter text

- reporter_realname text
- bug_status varchar(10)
- resolution text
- classification varchar(20)
- product varchar(20)
- component varchar(20)
- version smallint
- op_sys varchar(20)
- votes smallint
- target_milestone varchar(20)
- qa_contact varchar(20)
- qa_contact_realname text
- status_whiteboard varchar(30)
- keywords text
- short_desc text
- short_short_desc text

B.3 change_sets

- changesetid int(10)
- filepath varchar(255)
- datetime datetime
- author varchar(20)
- revision varchar(20)
- state varchar(10)
- comment text
- linenum text

B.4 usages

- fileid1 int(10)
- fileid2 int(10)

B.5 bidirectionals

- fileid1 int(10)
- fileid2 int(10)

B.6 files

- fileid int(10)
- cyclegroup int(10)
- numAuthors int(10)
- numFaults int(10)
- numChanges int(10)
- age int(10)
- relNumFaults float
- relNumAuthors float
- numBidirs int(10)
- numDeps int(10)
- cycleSize int(10)

B.7 cycles

- cyclegroup int(10)
- cyclesize int(10)
- sumbidirs int(10)
- sumdeps int(10)
- bidirdensity float
- depsdensity float

Anhang C

Tabelle der t-Quantilverteilung

Tabelle C.1: Quantile $t_{1-\alpha;n}$ der t-Verteilung zur statistischen Sicherheit $1-\alpha$ im Abhängigkeit vom Freiheitsgrad n

n	Statistische Sicherheit 1-α						n
	0,90	0,95	0,975	0,99	0,995	0,999	
1	3,078	6,314	12,71	31,82	63,66	318,3	1
2	1,886	2,920	4,303	6,965	9,925	22,33	2
3	1,638	2,353	3,182	4,541	5,841	10,21	3
4	1,533	2,132	2,776	3,747	4,604	7,173	4
5	1,476	2,015	2,571	3,365	4,032	5,893	5
6	1,440	1,943	2,447	3,143	3,707	5,208	6
7	1,415	1,895	2,365	2,998	3,499	4,785	7
8	1,397	1,860	2,306	2,896	3,355	4,501	8
9	1,383	1,833	2,262	2,821	3,250	4,297	9
10	1,372	1,812	2,228	2,764	3,169	4,144	10
11	1,363	1,796	2,201	2,718	3,106	4,025	11
12	1,356	1,782	2,179	2,681	3,055	3,930	12
13	1,350	1,771	2,160	2,650	3,012	3,852	13
14	1,345	1,761	2,145	2,624	2,977	3,787	14
15	1,341	1,753	2,131	2,602	2,947	3,733	15
16	1,337	1,746	2,120	2,583	2,921	3,686	16
17	1,333	1,740	2,110	2,567	2,898	3,646	17
18	1,330	1,734	2,101	2,552	2,878	3,610	18
19	1,328	1,729	2,093	2,539	2,861	3,579	19
20	1,325	1,725	2,086	2,528	2,845	3,552	20
21	1,323	1,721	2,080	2,518	2,831	3,527	21
22	1,321	1,717	2,074	2,508	2,819	3,505	22
23	1,319	1,714	2,069	2,500	2,807	3,485	23
24	1,318	1,711	2,064	2,592	2,797	3,467	24
25	1,316	1,708	2,060	2,485	2,787	3,450	25
26	1,315	1,706	2,056	2,479	2,779	3,435	26
27	1,314	1,703	2,052	2,473	2,771	3,421	27
28	1,313	1,701	2,048	2,467	2,763	3,408	28
29	1,311	1,699	2,045	2,462	2,756	3,396	29
30	1,310	1,697	2,042	2,457	2,750	3,385	30
40	1,303	1,684	2,021	2,443	2,704	3,307	40
50	1,299	1,676	2,009	2,403	2,678	3,261	50
60	1,296	1,671	2,000	2,390	2,660	3,232	60
70	1,294	1,667	1,994	2,381	2,648	3,211	70
80	1,292	1,664	1,990	2,374	2,639	3,195	80
90	1,291	1,662	1,987	2,368	2,632	3,183	90
100	1,290	1,660	1,984	2,364	2,626	3,174	100
200	1,286	1,652	1,972	2,345	2,601	3,131	200
500	1,283	1,648	1,965	2,334	2,586	3,107	500
∞	1,282	1,645	1,960	2,326	2,576	3,090	∞

Anhang D

Ergebnisse der statistischen Analyse

D.1 Beschreibende statistische Werte für die Daten

D.2 Paarweise Korrelation aller Maße

Tabelle D.3: paarweise Korrelation der Maße auf der Dateiebene 1, $\alpha = 0.01$

	FEHLER	ZEICHEN	ALTER	REVISIONEN INSGESAMT	AENDERUNGEN OHNEFIXES
FEHLER	1	0,56	0,16	0,86	0,66
ZEICHEN	0,56	1	0,1	0,58	0,51
ALTER	0,16	0,1	1	0,32	0,38
REVISIONENINSGESAMT	0,86	0,58	0,32	1	0,95
AENDERUNGENOHNEFIXES	0,66	0,51	0,38	0,95	1
AUTOREN	0,52	0,33	0,5	0,66	0,65
AUTORENOHNEFIXES	0,42	0,29	0,56	0,64	0,68
NACHBARNIMZYKLUS	0,36	0,4	0,15	0,44	0,43
BIDIRECTIONALEABH	0,13	0,15	0,04	0,15	0,15
ZYKLENGRÖSSE	0,15	0,13	0,1	0,21	0,22

Tabelle D.1: Beschreibende statistische Werte für die Maße 1

	ALTER	AUTOREN	AUTORENOHNE FIXES	BIDIRECTIONALE ABH	FEHLER
Mean	1077,2	2,67	2,26	2,06	2,2
Median	1021	2	2	0	1
Maximum	2009	19	14	171	103
Minimum	19	1	0	0	0
Std. Dev.	568,81	1,96	1,79	5,4	5,1
Skewness	0,07	1,77	1,52	12,64	7,82
Kurtosis	1,79	7,6	6,44	323,47	96,92
Jarque-Bera	815,12	18684,4	11710,38	57279874	5024565.
Probability	0	0	0	0	0
Sum	14329996	35566	29999	27349	29269
Sum Sq. Dev.	4.30E+09	50931,15	42527,6	388579,5	346657,9
Observations	13303	13303	13303	13303	13303

Tabelle D.2: Beschreibende statistische Werte für die Maße 2

	AENDERUNGEN OHNEFIXES	ZEICHEN	ZYKLEN GRÖSSE	NACHBARN IMZYKLUS	REVISIONEN INSGESAMT
Mean	5,64	6209,02	124,87	3,59	15,32
Median	3	2501	0	0	8
Maximum	251	384738	916	406	640
Minimum	0	0	0	0	1
Std. Dev.	8,53	13296,86	253,24	9,83	25,16
Skewness	7,16	9,25	2,24	10,94	7,41
Kurtosis	108,51	149,27	6,91	270,14	99,23
Jarque-Bera	6283556.	12049256	19620,41	39821047	5254642.
Probability	0	0	0	0	0
Sum	75005	82598553	1661127.	47777	203757
Sum Sq. Dev.	968041,7	2.35E+12	8.53E+08	1284309.	8419460.
Observations	13303	13303	13303	13303	13303

Tabelle D.4: paarweise Korrelation der Maße auf der Dateiebene 2, $\alpha = 0.01$

	AUTOREN	AUTORENOHNE FIXES	NACHBARN IMZYKLUS	BIDIRECTIONALE ABH	ZYKLEN GRÖSSE
FEHLER	0,52	0,42	0,36	0,13	0,15
ZEICHEN	0,33	0,29	0,4	0,15	0,13
ALTER	0,5	0,56	0,15	0,04	0,1
REVISIONENINSGESAMT	0,66	0,64	0,44	0,15	0,21
AENDERUNGENOHNEFIXE	0,65	0,68	0,43	0,15	0,22
AUTOREN	1	0,93	0,31	0,12	0,3
AUTORENOHNEFIXES	0,93	1	0,3	0,11	0,3
NACHBARNIMZYKLUS	0,31	0,3	1	0,29	0,35
BIDIRECTIONALEABH	0,12	0,11	0,29	1	0,05
ZYKLENGRÖSSE	0,3	0,3	0,35	0,05	1

Anhang A

Bash-Skripte zur Datenbeschaffung

A.1 createMysqlDatafiles.sh

```
#!/bin/sh
if [ "$1" != "" ]; then
#pfad der skripte:
    binpath=‘echo $0 | sed ’s/\(.*\)\/.*/\1/’‘
#cvs-einstellungen
    if [ "$2" != "" ]; then
        cvsargs=$2
    else
        cvsargs=-d:pserver:anonymous@dev.eclipse.org:/home/eclipse

    fi

# anlegen der verzeichnisse cvs und bugzilla,
#sowie das output-verzeichniss mysqlfiles
    mkdir cvs
#bugzilla-verzeichnis wird vorausgesetzt.
#Es muss alle .csv-dateien enthalten
    mkdir mysqlfiles

# auschecken der module aus der Datei mit dem
#Namen $1 ins verzeichnis cvs und holen
# der log-eintraege in die Datei cvs.log

    cd cvs
    for i in ‘cat ../$1‘;
        do
```

```
        cvs $cvsargs co $i;
        done
    ../$binpath/get.cvslog.for.this.directory.sh
  # konvertieren der logdatei in eine mysql-Datei
  #und kopieren in das Ergebnis-Verzeichnis
  # Der Dateiname wird in der Datei mysqlfiles.txt protokoliert.

    perl ../$binpath/perl/cvslog2mysqlConverter.pl cvs.log
    cp cvs.log.mysql ../mysqlfiles/cvslog.mysql
    cd ..
    echo cvslog.mysql >> mysqlfiles/mysqlfiles.txt
  # alle bugzilla-csv-dateien konvertieren
  #und ins ergebnis-verzeichnis kopieren
  # die Dateinamen werden in der mysqlfiles.txt protokoliert.

    cd bugzilla
    for f in *.csv;
        do
        perl ../$binpath/perl/buglist2mysqlConverter.pl $f
        cp $f.mysql ../mysqlfiles/$f.mysql
        echo $f.mysql >> ../mysqlfiles/mysqlfiles.txt;
        done
  #alle backslashes durch slashes ersetzen

    cd ../mysqlfiles
    for i in *.mysql;
        do
        cat $i | sed 's/\\/\//g' > $i.tmp;
        cat $i.tmp > $i;
        done
  else echo "usage: eclipseCreateMysqlDatafiles.sh fileContainingTheModuleList
cvsarguments"
  fi
```

A.2 eclipse.checkout.modules.sh

```
  #!/bin/sh
  for i in `cat $1`;
      do
      cvs $2 co $i;
      done
```

A.3 eclispe.get.cvslog.sh

```
 #!/bin/sh
if [ "$1" != "" ]; then
    for dir in `cat $1`; do
    echo $dir
    if [ -d "$dir" ]; then
        cd $dir
        cvs log >> ../all.cvslog
        cd ..
    else
        echo "not all directories in the list do exist"
    fi
    done
else
    echo "usage: eclipse.get.cvslog.sh list.of.module.directories"
fi
```

A.4 eclipseCreateMysqlDatafiles.sh

```
 #!/bin/sh
#pfad der skripte:
binpath=`echo $0 | sed 's/\(.*\)\/.*/\1/'`
$binpath/createMysqlDatafiles.sh
modules.txt -d:pserver:anonymous@dev.eclipse.org:/home/eclipse
```

A.5 get.cvslog.for.this.directory.sh

```
 #!/bin/sh
for dir in `ls`; do
    if [ -d $dir ]; then
    cd $dir
    echo
    echo
    echo $dir
    cvs log >> ../cvs.log
    cd ..
fi; done
```