Diplomarbeit

Bewertung der Prozessqualität von Open Source-Entwicklungsprozessen anhand von Software-Prozessdaten

von

Henning Schaefer

Vorgelegt der

Fakultät für Mathematik, Informatik und Naturwissenschaften der
Rheinisch-Westflischen Technischen Hochschule Aachen

im Januar 2009

Angefertigt am
Lehr- und Forschungsgebiet Informatik III
Prof. Dr. rer. nat. Horst Lichter

Gutachter:
Prof. Dr. rer. nat. Horst Lichter
Prof. Dr.-Ing. Manfred Nagl

Betreuer:
Dipl.-Inform. Holger Schackmann

Hiermit versichere ich, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie Zitate kenntlich gemacht habe.

Aachen, den 23. Januar 2009

Henning Schaefer

Zusammenfassung

Neben der Messung der Qualität von Software-Entwicklungsprozessen mit Hilfe von Audits und direkten Prozessanalysen stehen inzwischen auch weitere Quellen und Hilfsmittel zur Ermittlung und Bewertung verschiedener Aspekte der Prozessqualität zur Verfügung.
Diese Arbeit befaßt sich mit der Konzeption, Modellierung und Auswertung von Prozessmetriken anhand der Datenbanken eines Change-Request-Management-Systems basierend auf den Software-Werkzeugen *Bugzilla* und *BugzillaMetrics* sowie der Bewertung des Entwicklungsprozesses des Open Source-Softwareprojekts *Eclipse*.

Inhaltsverzeichnis

1 Einleitung **3**

1.1 Gegenstand dieser Arbeit 4

1.2 Bezeichnungen und Konventionen 4

1.3 Bugzilla 6

1.4 BugzillaMetrics 7

1.5 Das Eclipse-Projekt 8

2 Modellierung von Prozessqualität **11**

2.1 Modellierung von Qualität 12

2.2 Identifikation von Qualitätsmodellen 25

2.3 Quality Benchmark Levels 27

3 Merkmale der Open Source-Entwicklung **31**

3.1 Strukturelle Unterschiede 31

3.2 Die Entwickler-Gemeinde 32

3.3 Die Anwender-Gemeinde 34

3.4 Unterschiede bei der Prozess-Gestaltung 35

4 Metriken auf Bug-Tracking-Datenbanken **37**

4.1 Wesen der Information in einer Bug-Tracking-Datenbank 37

4.2 Randbedingungen 38

4.3 Entwicklung von Metriken mit Bugzilla Metrics 40

5 Evaluierung des Eclipse-Prozesses 47

5.1 Der Eclipse-Entwicklungsprozess 48

5.2 Ein Qualitätsmodell für Eclipse 54

5.3 Unterschiedliche Sichtweisen 59

5.4 Metriken für das Eclipse-Qualitätsmodell 61

5.5 Vermessung der Eclipse-Projekte 84

6 Abschließende Betrachtungen 115

6.1 Zusammenspiel mit anderen Auswertungsverfahren 115

6.2 Aufgetretene Schwierigkeiten 118

6.3 Ausblick 119

6.4 Fazit 121

Kapitel 1

Einleitung

Bereits seit längerer Zeit erfreuen sich Auswertungen jeglicher Art im Zusammenhang mit der Entwicklung von Softwareprodukten wachsender Beliebtheit. Hierbei kommen sowohl subjektive Einschätzungen (in Form von Audits) wie auch objektive Messungen am zu betrachtenden Objekt (sogenannte *Metriken*) zum Einsatz.

Meist werden derartige Messungen zur Sicherstellung der Qualität des Endprodukts sowie zur Verbesserung des Entwicklungsprozesses vorgenommen. Beispielhaft seien hier Messungen am Programmquelltext (z.B. um die Einhaltung einer vorgegebenen Codegüte zu überprüfen) oder am Binärcode des Programms (sogenannte *Profiler* zur Messung der Laufzeit- und Speichereffizienz einzelner Programmkomponenten) genannt.

Zur Ermittlung der Quelltext-Güte und der Effizienz des ausführbaren Programms ist bereits eine große Anzahl entsprechender Software-Werkzeuge verfügbar; hingegen finden Einordnungen der Prozessqualität oft ohne eine entsprechende Tool-Unterstützung in Form von Mitarbeitergesprächen und Prozess-Audits statt, da es an zuverlässigen Quellen zur automatischen Bewertung der Prozessqualität fehlt.

Die Schwierigkeit bei der Ermittlung von aussagekräftigen Angaben zur Qualität von Entwicklungsprozessen besteht meist darin, objektive und vergleichbare Kriterien zur Einstufung des Prozesses zu ermitteln. Der verwendete Prozess und dessen Umsetzung wirkt sich jedoch nicht nur auf die Produktqualität aus; im Verlauf eines Software-Lebenszyklus fallen neben dem Endprodukt noch weitere Artefakte an, die sowohl vom Prozess als auch seiner Umsetzung beeinflußt werden und somit ebenfalls zur Bewertung der Prozessqualität herangezogen werden können.

Diese Artefakte umfassen unter anderem Datenarchive aus *Configuration Management*-Systemen wie *CVS*, archivierte E-Mail-Kommunikation oder auch Daten aus sogenannten *Change Request Management*-Systemen [KCM07]. Die Auswertung der in diesen Archiven vorgehaltenen Daten läßt unter gewissen Voraussetzungen Rückschlüsse auf die qualitative Entwicklung des Software-Entwicklungsprozesses und dessen Einhaltung zu:

> These repositories hold a wealth of information and provide a unique view of the actual evolutionary path taken to realize a software system. Often these data exist for the entire duration of a project and can represent thousands of versions with years of details about the development. These data include such things as individual versions of the system, the changes, and metadata about the changes (e.g., who made the change, why the change was made, when the change was done, etc.).[KCM07]

1.1 Gegenstand dieser Arbeit

Diese Arbeit befaßt sich mit der Auswertung eines dieser Artefakte. Anhand der in einem Change-Request-Management-System (CRM) erfassten Daten werden Rückschlüsse auf Qualität und Einhaltung des Entwicklungsprozesses gezogen. Hierzu erfolgen zunächst allgemeine Beobachtungen zu den Zusammenhängen zwischen den Inhalten der Datenbank des CRM-Systems und der Prozessqualität. Sodann erfolgt die Beschreibung eines Qualitätsmodells zur werkzeuggestützten Auswertung der erfassten Daten sowie die Entwicklung entsprechender Meßverfahren. Abschließend werden diese Meßverfahren auf einen realen Prozess aus dem Open Source-Umfeld angewandt und ausgewertet.

Da es sich bei dem beispielhaft betrachteten Projekt um ein Open Source-Entwicklungsprojekt handelt dessen Vorgehensweise sich in Teilen erheblich von klassischen Prozessverfahren unterscheidet, wird in einem separaten Abschnitt auf die Besonderheiten und speziellen Anforderungen von Open Source-Entwicklungsprozessen an die Auswertungsmethodik eingegangen.

Zunächst wird in Kapitel 2 auf die Bedeutung des Begriffs ‘Prozessqualität’ eingegangen und es werden Möglichkeiten aufgezeigt, Prozessqualität anhand unterschiedlicher Anforderungen zu modellieren. Anschließend folgt in Kapitel 3 eine Diskussion der Besonderheiten, die im Rahmen der Auswertung von Open Source-Entwicklungsprojekten zu beachten sind. Das Kapitel 4 befaßt sich mit der Entwicklung von Metriken zur Auswertung von Bug-Tracking-Datenbanken mit Hilfe von Bugzilla Metrics und den Rahmenbedingungen für den Einsatz dieses Werkzeugs. Schließlich folgt in Kapitel 5 eine vergleichende Auswertung der verschiedenen Produkte des Eclipse-Projekts anhand der in den vorhergehenden Kapiteln entwickelten Methodiken, bevor in Kapitel 6 eine Zusammenfassung der Thematik und ein Ausblick auf zukünftige Entwicklungen gegeben wird.

1.2 Bezeichnungen und Konventionen

Im Fachgebiet der Softwarekonstruktion finden je nach angewandtem Prozess bzw. verwendeter Software-Hilfsmittel die unterschiedlichsten Begrifflichkeiten Anwendung.

Daher ist es erforderlich, einige Bezeichnungen und Konventionen näher zu erläutern und für die Verwendung in dieser Arbeit festzulegen.

1.2.1 Change Request Management

Unter dem Begriff *Change Request Management* versteht man im Allgemeinen die Verwaltung von Änderungswünschen an einer bestehenden Software-Konfiguration. Im täglichen Sprachgebrauch wird hiermit insbesondere ein System zur Erfassung von Fehlerberichten und Erweiterungswünschen bezüglich eines Softwareprodukts verstanden (ein sogenanntes *Bug Tracking*-System). Diese Arbeit orientiert sich beim Gebrauch des Begriffs *Change Request Management* an dieser alltäglichen Definition; mit *Change Request Management*-Systemen sind im Folgenden durchgängig ausschließlich Bug-Tracking-Systeme gemeint.

Außerdem werden die Begriffe *Change-Request-Management-System* (*CRM*) sowie *Bug Tracking System* synonym verwendet.

1.2.2 Bezeichnung von Change Requests

Je nach Hintergrund des Prozesses und verwendetem Software-Werkzeug finden sich zur Beschreibung eines sogenannten *Change Requests*, der Anforderung einer Änderung am Softwareprojekt, verschiedene Bezeichnungen; darunter sind *Change Request* (bzw. *CR*), *Case*, *Issue* oder *Bug* sehr gebräuchlich. Im Umfeld der verwendeten Software-Werkzeuge `Bugzilla` und `BugzillaMetrics` finden sich die beiden Begriffe *Bug* und *Case*. Um eine Verwechslung mit der Abkürzung *CASE* für *Computer Aided Software Engineering* zu vermeiden, wird in dieser Arbeit überwiegend die Bezeichnung ***Bug*** Verwendung finden. Aus sprachlichen Gründen werden die deutschen Begriffe ***Fall*** sowie ***Fehlerbericht*** synonym zu dieser Bezeichnung verwandt. Sofern explizit zwischen Einträgen, die einen tatsächlichen Fehler berichten und solchen, die einen Erweiterungswunsch beinhalten unterschieden werden muß, ist dies explizit angegeben.

Die Verwendung der Worte *Bug* und *Fehlerbericht* für Einträge im *CRM*-System kann zu Mißverständnissen führen, da unter diesen Begriffen im Allgemeinen nur ein Change Request zur Behebung eines Fehlers verstanden wird; daher wird an dieser Stelle ausdrücklich darauf hingewiesen, daß die Begriffe *Bug* bzw. *Fehlerbericht* neben Change Requests zur Fehlerbehebung in dieser Arbeit unbedingt auch Change Requests zur Erweiterung der Produktfunktionalität umfassen. Aus diesem Grund wird auch dann von der *Behebung eines Bugs* gesprochen, wenn die gewünschte Erweiterung umgesetzt wurde.

1.2.3 Der Begriff *Metrik*

Im folgenden wird der Begriff *Metrik* ausschließlich für die Definition eines Meßverfahrens und gegebenenfalls dessen Implementierung z.B. in Form der XML-Metrikdefinition in *BugzillaMetrics*, verwandt; für die in einem Meßverfahren ermittelten Werte wird zu einem späteren Zeitpunkt eine separate Begrifflichkeit eingeführt (siehe hierzu Kapitel 2.1).

1.2.4 Der Begriff *Prozess*

Der Begriff *Prozess* wird im Allgemeinen für eine festgelegte Abfolge von Ereignissen und Tätigkeiten im Rahmen eines Entwicklungsprojekts verwandt. Die Norm *DIN EN ISO 8402* definiert einen Prozess als einen

> [...] Satz von in Wechselbeziehungen stehenden Mitteln und Tätigkeiten, die Eingaben in Ergebnisse umgestalten, wobei unterschiedliche Mittel angewendet werden

Das Eclipse-Projekt faßt eine Vielzahl von Beschreibungen solcher Tätigkeiten unter dem Begriff *Prozess* zusammen. Beispielhaft zu nennen seien hier die Prozesse zur Aufnahme neuer Teilprojekte in das Eclipse-Gesamtprojekt (*Incubation*), der *Roadmap Process* [EDP08], der Review-Prozess zum Projekt-Lebenszyklus und der Prozess zur Begutachtung von Beiträgen die möglicherweise die Urheberrechte Dritter oder die Eclipse-Lizenz verletzen.

Um Verwechslungen zu vermeiden meint der Begriff *Prozess* ausschließlich diejenigen Vorgaben, die mit der projektinternen Fortentwicklung deŕ Produkte in Zusammenhang stehen. Prozesse, die über die reine Entwicklungstätigkeit eines Projekts hinausgehen, sind von diesem Begriff ausdrücklich nicht erfaßt.

1.3 Bugzilla

Das Software-Werkzeug `Bugzilla` [Mozb] ist ein Bug Tracking System, das die Erfassung von Bugs sowie die Überwachung der Behebung derselben ermöglicht. Das System wurde in `PERL` entwickelt und unterstützt das verbreitete Datenbanksystem `MySQL`. Die Bedienung durch den Anwender erfolgt über eine Web-basierte Benutzerschnittstelle, die sich in jedem Web-Browser darstellen läßt. Durch diese Architektur läßt sich Bugzilla auf den verschiedensten Betriebssystemumgebungen nutzen.

Bugzilla ist fähig, mehrere Projekte und Subprojekte in einer dreistufigen hierarchischen Einstufung zu verwalten. Jeder Bug innerhalb des Systems besitzt mehrere Eigenschaftswerte bezüglich Schwere, Priorität, zugeordnetem Benutzer, der Programmversion auf die sich der Bug bezieht, dem momentanen Bearbeitungszustand sowie

verschiedene andere Angaben. Wird ein Bug nach der Erfassung im System geändert, so wird ein vollständiges Änderungsjournal für jeden einzelnen Bug geführt; mit Hilfe eines solchen Journals, das jede Änderung eines Datenbankeintrags protokolliert, läßt sich der Verlauf der Bearbeitung jederzeit rekonstruieren. Erst diese Funktionalität macht sinnvolle Auswertungen des Datenbankinhaltes überhaupt möglich, da eine rein statische Auswertung der Daten zu einem festgelegten Zeitpunkt häufig nicht aussagekräftig ist.

> Bugzilla can help you get a handle on the software development process. Successful projects often are the result of successful organization and communication. Bugzilla is a powerful tool that will help your team get organized and communicate effectively. [Moza]

Dieses Zitat verdeutlicht bereits den Einfluss, den eine sorgfältige Verwaltung und Organisation der Bugs auf die Prozessqualität besitzt. Die Auswertung der mit Hilfe eines Bug Tracking-Werkzeugs erstellten Datenbanken kann also wertvolle Hinweise auf den Entwicklungsprozess und dessen Güte liefern.

1.4 BugzillaMetrics

Eine manuelle Auswertung aller Bugs in der Datenbank eines Bug Tracking-Werkzeugs ist aufgrund der meist sehr großen Anzahl von Einträgen in solch einem System nicht praktikabel. Darüber hinaus wird für eine aussagekräftige Aussage aufgrund statistischer Effekte eine gewisse Größe des Datenbestands benötigt (siehe hierzu auch Abschnitt 2.1.4).

Im Rahmen einer Diplomarbeit [Gra07] wurde daher an der RWTH Aachen das Software-Werkzeug `BugzillaMetrics` [G$^+$] entwickelt, das eine automatisierte Auswertung der verschiedensten Metriken auf Bugzilla-Bug-Datenbanken ermöglicht.

Bugzilla Metrics verwendet eine XML-Beschreibungssprache zur Definition von Metriken und Ausgabeformat und berechnet basierend auf den in der Datenbank des Bugtrackers hinterlegten Daten sogenannte *Case Values*, die mit Hilfe der Metrikbeschreibung in unterschiedlicher Weise verrechnet werden können. So lassen sich verschiedene arithmetische (Summe, Differenz, Multiplikation und Division) und statistische Operationen (Mittelwert- und Durchschnittsbildung) auf die ermittelten Case Values anwenden und zu vom Benutzer festzulegenden Punkten auf einer Zeitachse berechnen.

Durch diese Funktionalität ist es nicht nur möglich, eine statische Analyse des Projekts zu einem einzelnen Zeitpunkt durchzuführen, sondern auch die Entwicklung des Projekts über einen größeren Zeitraum hinweg zu beobachten, um die Auswirkungen von Veränderungen auf die Projekt- und Prozessstruktur zu ermitteln.

Durch die Verwendung einer XML-Sprache zur Beschreibung benutzerdefinierter Metriken lassen sich die in Bugzilla Metrics definierten Standard-Metriken im Rahmen der Möglichkeiten, die das Werkzeug bietet, beliebig anpassen und erweitern; hierdurch wird es möglich, das Werkzeug auch bei speziellen Bedürfnissen und individuellen Prozessen zu verwenden.

Die Ausgabe des Programms erfolgt entweder über Text-Dokumente in unterschiedlichen Formaten (XML, CSV oder HTML), so daß sich die Ergebnisse leicht in andere Programme zur weiteren Auswertung importieren lassen, oder über eine graphische Diagrammdarstellung (Abbildung 1.1).

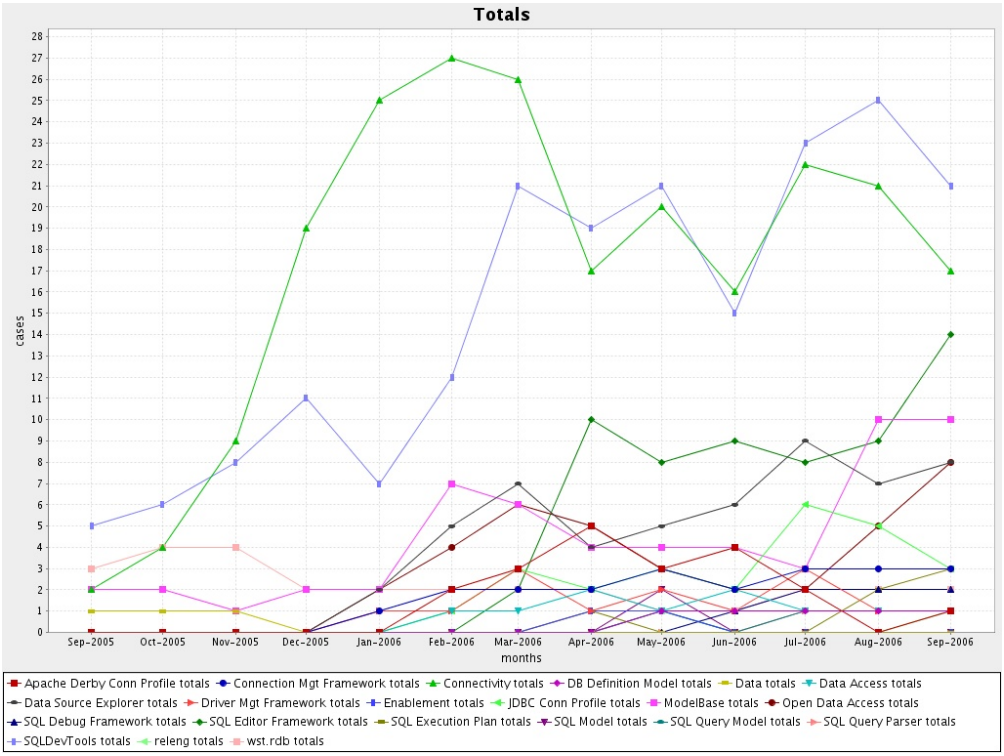

Abbildung 1.1: Graphische Darstellung der Auswertung

1.5 Das Eclipse-Projekt

Das Eclipse-Projekt [ECL] ist ein umfangreiches Open Source-Software-Entwicklungsprojekt das sich mit der Entwicklung der gleichnamigen Java-Entwicklungsumgebung, deren Komponenten und den zugrunde liegenden Technologien befaßt. Mitte der 1990er-Jahre durch `IBM` ins Leben gerufen wurde die Entwicklung bis 2004 durch `IBM` kontrolliert, bevor man sich veranlaßt sah, das Projekt in die Hände einer unabhängigen Institution zu übergeben, um die Offenheit des Projekts zu betonen [Cer05]. Die daraufhin gegründete gemeinnützige `Eclipse Foundation`, die das Projekt seit-

her verantwortlich leitet, wird von 12 Mitgliedsunternehmen unterstützt, die jeweils mindestens 8 Vollzeit-Entwickler und finanzielle Mittel in Höhe von bis zu 250000 US-Dollar jährlich bereitstellen. Darüberhinaus wird das Projekt von einer Vielzahl weiterer Unternehmen und unabhängiger Entwickler unterstützt, wodurch Eclipse zur größten nicht von `Microsoft` kontrollierten Software-Entwicklungsplattform wurde [Cer05].

Die Entwicklung des Eclipse-Projekts folgt einem offenen, agilen Entwicklungsprozess, dessen Untersuchung der Hauptgegenstand dieser Arbeit ist. Offenheit definiert sich in diesem Fall wie folgt:

> Eclipse is open to all; Eclipse provides the same opportunity to all. Everyone participates with the same rules; there are no rules to exclude any potential contributors which include, of course, direct competitors in the marketplace. [EDP08]

Aufgeteilt ist das Projekt in 11 sogenannte *Classifications*, die verschiedene verwandte Subprojekte des Gesamtprojekts abbilden und deren Entwicklung weitestgehend unabhängig voneinander abläuft. So kümmern sich die Projekte die in der Classification `Eclipse` zusammengefaßt sind um die Bereitstellung der grundlegenden Funktionalität des Gesamtprodukts während weitere Projekte, die in anderen *Classifications* zusammengefaßt sind, diese Grundfunktionalität auf vielfältige Weise erweitern.

Eine Besonderheit des Eclipse-Projekts ist das sogenannte *Simultaneous Release*, das zu allen *Major Releases*, die den einzelnen Versionsschritten des Projekts entsprechen, einen gemeinsamen Veröffentlichungszeitpunkt für die wichtigsten Subprojekte festlegt. Im Rahmen des *Simultaneous Release* wird demnach für alle beteiligten Projekte ein zeitliches Ziel definiert bis zu dem die geplante Funktionalität umzusetzen ist. Darüber hinaus ermöglicht das *Simultaneous Release* eine vereinfachte Auswertung für die im Rahmen dieser Arbeit vorgesehenen Untersuchungen, da für alle beteiligten Projekte der gleiche Auswertungszeitraum zugrunde gelegt werden kann.

Das *Simultaneous Release*, die im Vergleich zu vielen Open Source-Projekten umfangreich und gut dokumentierte Prozessdefinition sowie die Offenheit und Vielfalt des Projekts selbst machen das Eclipse-Projekt zu einem idealen Untersuchungsobjekt für die im Rahmen dieser Arbeit vorgesehenen Auswertungen.

Kapitel 2

Modellierung von Prozessqualität

Der Begriff *Qualität* wird durch das *Deutsche Institut für Normung* (DIN) wie folgt definiert:

> [Qualität ist] die Gesamtheit von Eigenschaften und Merkmalen eines Produktes, die sich auf dessen Eignung zur Erfüllung gegebener Erfordernisse beziehen
> *DIN 55350*, aus [SSM06]

Qualität, unabhängig davon, ob es sich bei dem betrachteten 'Produkt' um ein konkretes (Software-)Produkt oder um einen abstrakten Prozess handelt, ist also nie absolut, sondern immer relativ zu den 'gegebenen Erfordernissen' zu beurteilen. Ein Kriterium, das für die Qualität eines Produkts oder Prozesses maßgeblich ist, kann demnach für ein anderes Produkt oder einen anderen Prozess völlig unerheblich sein. Die Aufgabe der Messung von Qualität ist dementsprechend auch, durch Analyse der gegebenen Erfordernisse eine Vergleichsbasis zu schaffen, anhand derer die Qualität des betrachteten Produkts oder Prozesses beurteilbar gemacht wird.

Darüber hinaus ist zu berücksichtigen, daß es sich bei Qualität um einen abstrakten Begriff handelt; die Sichtweisen unterschiedlicher Personen auf die Qualität des gleichen Produkts oder Prozesses können durchaus stark differieren:

> Im allgemeinen Sprachgebrauch wird der Begriff der Qualität oft sehr undifferenziert verwendet, da er abstrakt und sehr vielschichtig ist. Auch die International Organization for Standardization (ISO) merkt dazu an: '*Like beauty, everyone may have his or her idea of what quality is*'
> *aus* [SSM06]

Neben der Analyse der 'gegebenen Erfordernisse' ist es bei der Messung von Qualität also ebenfalls erforderlich, eine gemeinsame Betrachtungsebene für alle beteiligten Personen zu schaffen, so daß der Begriff 'Qualität' objektiviert werden kann. Im

Rahmen dieser Objektivierung sind mehrere Transformationen notwendig. Zunächst transformiert eine Messung eine abstrakte Eigenschaft des betrachteten Produkts oder Prozesses in eine konkrete, greifbare Zahl. Diese Zahl selbst sagt jedoch noch nichts über die Qualität selbst aus, sondern benötigt eine gültige Interpretation mit Hilfe derer der Brückenschlag zum abstrakten Qualitätsbegriff gelingt.

Der einfachste Ansatz, diese Transformation zu bewerkstelligen, besteht in der Erstellung eines Modells, das von den vielen meßbaren Eigenschaften eines Produkts oder Prozesses auf einen greifbaren und eindeutigen Qualitätsbegriffs abstrahiert. Die folgenden Abschnitte diskutieren sowohl den Aufbau wie auch die konkrete Umsetzung eines Qualitätsmodells, das die angesprochenen Anforderungen erfüllt.

2.1 Modellierung von Qualität

Der Entwurf geeigneter Metriken zur zuverlässigen Bewertung der Qualität sowohl des Produkts wie auch des Prozesses stellt verschiedene Herausforderungen an die Methodik, die zum Herleiten der Meß- und Vergleichsverfahren verwandt wird. In diesem Abschnitt wird die Entwicklung eines flexiblen und anpassungsfähigen Qualitätsmodells beschrieben, das auf unterschiedliche Projektstrukturen anwendbar ist und sich auf verschiedene Anforderungen anpassen läßt.

Der einfachste Ansatz zur Auswertung eines Satzes von gegebenen Metriken wäre der direkte Vergleich der Meßwerte. Dieses Verfahren birgt jedoch einige erhebliche Nachteile die im Folgenden noch näher erläutert werden, da die Zusammenhänge zwischen den Meßergebnissen und den einzelnen Aspekten des Gesamtprozesses vernachlässigt werden. Die Beschreibung jedes Prozesses läßt sich in unterschiedlich stark gewichtete Teilaspekte aufteilen, die sich wiederum in Unteraspekte gliedern (zum Beispiel betont ein Prozess die Zusammenarbeit zwischen den Projektbeteiligten stärker als ein anderer, der ein besonders starkes Gewicht auf die Einhaltung des geplanten Funktionsumfangs für den nächsten Releasezyklus legt). Diese interne Prozesstruktur würde beim direkten Vergleich der einzelnen Meßwerte nicht berücksichtigt, woraus ein ungenaues und wenig aussagerkräftiges Ergebnis resultieren würde. Darüber hinaus wäre mit dem soeben beschriebenen Vorgehen kein Vergleich auf der Ebene der verschiedenen, von der Prozessbeschreibung betonten Aspekte der Entwicklung, die möglicherweise genaueren Aufschluss über die Ursachen einer Prozessverbesserung oder -verschlechterung geben könnte, möglich.

Außerdem muß bei der Entwicklung von Metriken immer eine Zuordnung zu diesen Prozessaspekten stattfinden, um feststellen zu können, ob die Anwendung einer Metrik auf den Prozess überhaupt sinnvoll ist bzw. wie aussagekräftig diese Metrik tatsächlich ist. Beispielsweise wäre eine Metrik, die die Prioritätsverteilung der verschiedenen Fehlerberichte analysiert nicht aussagekräftig für einen Prozess, in dem keine Verwendung der Priorisierungsoptionen des Bug Tracking-Systems vorgesehen ist.

Bei der Entwicklung eines Qualitätsmodells ist also ein zweistufiges Vorgehen erforderlich. Zunächst müssen die Teilaspekte des Prozesses identifiziert und ggf. in feinere Unteraspekte gegliedert werden und anschließend gemäß ihrer Gewichtung in Relation zueinander gesetzt werden. Im zweiten Schritt werden dann zu den einzelnen Teil- bzw. Unteraspekten Metriken definiert, die genau diese Unteraspekte aussagekräftig zu messen in der Lage sind. Zur Ermittlung der Gesamtprozessqualität können dann die Messwerte dieser Teil- bzw. Unteraspekte mit einer ebenfalls zu definierenden Rechenvorschrift zusammengefaßt werden, um Aussagen über die Qualität der übergeordneten Aspekte bzw. des gesamten Prozesses zu gewinnen.

Laut Simon, Seng und Mohaupt [SSM06] lassen sich Qualitätsmodelle für Quellcode-Bewertung in Form einer Baum-ähnlichen Struktur entwickeln. Schackmann und Lichter [SL08] übertragen diese Darstellung eines Qualitätsmodells auch auf die Bewertung von Entwicklungsprozessen, da diese Modelldarstellung die oben aufgeführten Anforderungen an ein aussagekräftiges Qualitätsmodell erfüllt.

Im folgenden wird das Meta-Modell zur Entwicklung individueller Qualitätsmodelle auf Basis des in [SSM06] eingeführten Baum-ähnlichen Qualitätsmodells beschrieben.

2.1.1 Ein Qualitätsmodell in Baumstruktur

Die in [SSM06] und [SL08] eingeführten Qualitätsmodelle stellen sogenannte *bidirektionale Qualitätsmodelle* dar, die aus verschiedenen, im folgenden näher zu bezeichnenden Komponenten bestehen (Abbildung 2.1).

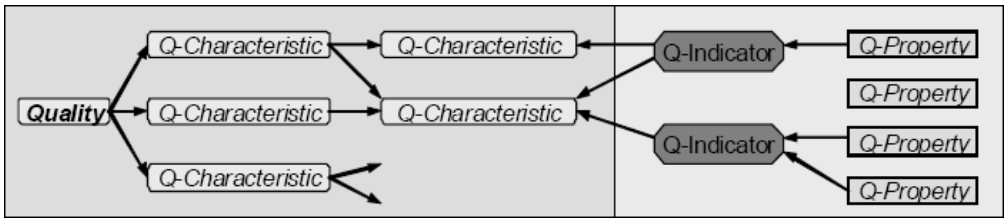

Abbildung 2.1: Struktur eines bidirektionalen Qualitätsmodells, *aus [SL08]*

Bidirektionale Qualitätsmodelle bestehen aus zwei miteinander verbundenen Komponenten, die unabhängig voneinander betrachtet werden können. An dieser Darstellung wird bereits deutlich, daß es sich bei den zwei Komponenten dieser Qualitätsmodelle nicht um echte Bäume handelt, sondern um *azyklische gerichtete Graphen*, da einzelne Knoten der Graphen mehr als eine eingehende Kante besitzen können. Da sich ein solcher Graph jedoch ohne Einfluß auf das Ergebnis der hier genutzten Anwendung in einen äquivalenten Baum umformen läßt, wird diese Graphenstruktur im Verlauf dieser Arbeit dennoch als *Baum* bezeichnet.

Die Knoten der linken Komponente des Modells setzen sich zusammen aus *Qualitätskriterien* (*Quality Characteristics*, die die einzelnen Teilaspekte des Prozesses beschreiben. Die (gewichteten) Kanten des linken Baums geben die Zusammenhänge zwischen diesen Teilaspekten bzw. Qualitätskriterien an. Prinzipiell ist eine beliebige

Tiefe des Baumes möglich, die ganaue Ebenenanzahl hängt von der Granularität der ermittelten Unteraspekte ab.

Die rechte Komponente des Modells besteht aus zwei unterschiedlichen Knotentypen, die grundsätzlich in nur zwei Ebenen angeordnet sind. Die erste Ebene besteht aus den *Qualitätsmerkmalen* (*Quality Indicators*), die eine Zusammenfassung der zweiten Ebene, den *Qualitätseigenschaften* (*Quality Properties*), nach einer noch näher zu bestimmenden Rechenvorschrift darstellen. Diese Ebene dient als Bindeglied zwischen der rein technischen Betrachtungsweise die bei der Ermittlung der Qualitätseigenschaften anwendung findet und der prozessorientierten, abstrakten Sichtweise der Qualitätskriterien. Die Qualitätseigenschaften ergeben sich aus den objektiv gemessenen Informationen, die bei der Auswertung der Metriken gewonnen werden.

2.1.2 Berechnung der Qualität übergeordneter Qualitätskriterien

Im Gegensatz zum zunächst erwähnten einfachen Qualitätsmodell das die Metrikergebnisse (bzw. die Qualitätseigenschaften) direkt miteinander vergleicht bieten sich bei Anwendung des bidirektionalen Qualitätenmodells Möglichkeiten, die einzelnen Qualitätskriterien untereinander zu vergleichen. Dies ermöglicht es beispielsweise auf einfache Weise Schwachstellen in der Prozessqualität aufzuspüren, indem die Werte der einzelnen Qualitätskriterien miteinander verglichen werden und Projekte, die in diesem Vergleich unterdurchschnittlich abschneiden, näher zu untersuchen. Wichtig ist dieses Vorgehen insbesondere dann, wenn man den zeitlichen Verlauf der Prozessqualität betrachten und die Effektivität und Auswirkungen von Maßnahmen zur Verbesserung der Prozessqualität ermitteln möchte.

Um diese Möglichkeit zu nutzen ist es jedoch erforderlich, eine Berechnungsvorschrift zur Ermittlung der Werte eines (Teil-)Kriteriums basierend auf den untergeordneten Kriterien zu definieren. In der graphischen Darstellung des Qualitätsmodells wird die Zuordnung von Teilkriterien zu übergeordneten Kriterien durch die Kanten in der linken Komponente realisiert. Da verschiedene Teilkriterien gemäß der Prozessdefinition unterschiedlich wichtig für den Prozess bzw. das übergeordnete Kriterium sein können, ist es erforderlich, diese Kanten zu gewichten. Wird bei der Identifikation der Prozesskriterien ein Kriterium als besonders wichtig empfunden, so ist die Kante, die dieses Kriterium mit dem übergeordneten Knoten verbindet stärker zu gewichten als die Kante, die den übergeordneten Knoten mit einem als weniger wichtig empfundenen Kriterium verbindet.

Verallgemeinert kann man also die Zuordnung von Teilkriterien zu einem übergeordneten Kriterium wie in Abbildung 2.2 darstellen.

Das übergeordnete Kriterium besitzt n ausgehende Kanten, die es jeweils mit den ihm zugeordneten Teilkriterien verbinden. Die Werte, die für die Teilkriterien ermittelt wurden, werden mit m_i bezeichnet, die Kantengewichte, die die Wichtigkeit jedes Teilkriteriums für as übergeordnete Kriterium definieren, sind mit w_i gekennzeichnet. Um nun eine Qualitätsaussage $f(m)$ für das übergeordnete Kriterium zu ermitteln,

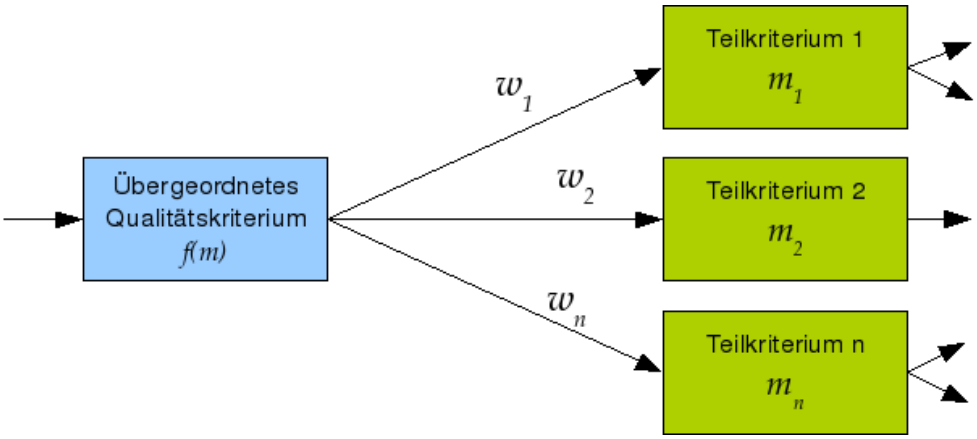

Abbildung 2.2: verallgemeinerte Darstellung der Zuordnung von Teilkriterien zu einem übergeordneten Kriterium

bietet es sich an, die gewichteten Werte der Teilkriterien als lineare Komponenten einer Funktion aufzufassen. Die Aufsummierung und anschließende Normierung durch Division durch die summierten Gewichte liefert dann den Funktionswert $f(m)$ des übergeordneten Teilkriteriums:

$$f(m) = \frac{\sum_{i=1}^{n}(w_i \cdot m_i)}{\sum_{i=1}^{n} w_i}$$

Diese Berechnungsvorschrift ist keinesfalls bindend, da je nach den Anforderungen an die Auswertung ebenso polynomielle oder logarithmische Formeln angewandt werden können.

Wendet man diese Berechnungsvorschrift bei den Qualitätsmerkmalen beginnend rekursiv auf den gesamten Baum an, so erhält man für jedes Qualitätskriterium wie auch für den Gesamtprozess einen individuellen Wert, der eine Aussage über die Qualität dieses Kriteriums liefert.

Über die oben beschriebene Berechnungsvorschrift hinaus lassen sich die in der Evaluierung der Metriken m ermittelten Werte an einer oberen respektive unteren Grenze vergleichen oder ihre Zugehörigkeit zu einem bestimmten Intervall prüfen (siehe hierzu auch den nächsten Abschnitt). Als Ergebnis dieses Vergleiches erhält man einen Wahrheitswert b, der die Werte 0 oder 1 (bzw. *wahr* oder *falsch* annehmen kann. Der Wert 1 bzw. *wahr* wird interpretiert als 'Der Metrikwert erfüllt eine bestimmte Vorgabe', der Wert 0 oder *falsch* hingegen als 'Der Metrikwert erfüllt eine bestimmte Vorgabe nicht'. Eine Gruppe von Metriken, die die Einhaltung bestimmter Vorgaben in den verschiedenen Qualitätseigenschaften überprüft, kann analog zur obigen Vorgehensweise als Baumstruktur ausgedrückt werden. Die Überprüfung, ob alle Vorgaben

einer solchen Gruppe eingehalten werden, kann dann durch logische Verknüpfung der in den Teilkriteien ermittelten Wahrheitswerte vorgenommen werden:

$$f(b) = \bigwedge_{i=1}^{n} b_i$$

für die Aussage 'Alle Vorgaben der Teilkriterien müssen eingehalten werden' und

$$f(b) = \bigvee_{i=1}^{n} b_i$$

für die Aussage 'Die Vorgabe mindestens eines Teilkriteriums muß eingehalten werden'.

Auf diese Weise ist es möglich, für die Auswertung verschiedene Qualitätsstufen (*Quality Benchmark Levels, QBLs*) zu definieren [SSM06], in denen jeweils unterschiedliche, aufeinander aufbauende Vorgaben einzuhalten sind. Jeder dieser Qualitätsstufen ist dabei ein separater Qualitätenbaum zugeordnet, der nach den hier eingeführten Regeln ausgewertet werden kann. Erreicht nun der Wurzelknoten des Baums nach der Berechnung den Wert *wahr*, so kann dies interpretiert werden als 'das betrachtete Projekt erfüllt die Vorgaben des zugehörigen *QBL*'. Derartige Vorgehensweisen finden sich beispielsweise auch bei der *Capability Maturity Model Integration* [Car], die ebenfalls der Bewertung der Qualität von Entwicklungsprozessen dient.

2.1.3 Vergleichbarkeit von Metriken

Bei der Berechnung von Qualitätseigenschaften treten naturgemäß sehr unterschiedlich dimensionierte Werte auf, die sich einer Zusammenfassung nach der im vorherigen Abschnitt eingeführten Berechnungsvorschrift entziehen. So ist beispielsweise die Zusammenfassung einer Prozent- oder Bruchangabe (die z.B. aus der Anwendung einer Metrik zur Ermittlung des Anteils als dringend eingestufter Bug resultiert) mit einer absoluten Angabe (die u.a. aus einer Metrik zur Ermittlung aller noch nicht behobenen Fälle resultiert) zunächst nicht sinnvoll möglich.

Selbst beim Vergleich unterschiedlicher Projekte anfallende gleich dimensionierte Werte (wie sie beispielsweise die Ermittlung der Anzahl aller noch nicht behobenen Fälle hervorgehen) können nicht problemlos miteinander verglichen werden. Besitzt Projekt A insgesamt 10000 Einträge in der Datenbank des Bug Tracking-Systems von denen 100 noch nicht behoben wurden, Projekt B hingegen 1000 Einträge, von denen 80 noch nicht behoben wurden, so würde bei einem Vergleich dieser beiden Werte das Projekt B besser abschneiden. Tatsächlich beträgt der Anteil an unbehobenen Einträgen bei Projekt A lediglich 1%, während Projekt B immerhin 8% unbehobene Einträge besitzt und somit objektiv als schlechter einzustufen ist. Aus diesem Grund sind derartige Metriken immer dahingehend zu entwickeln, daß eine Relation zur Projektgröße hergestellt wird (siehe hierzu auch Abschnitt 4.2).

Durch Beachtung dieser Besonderheit wird das Problem der Vergleichbarkeit unterschiedlich dimensionierter Metriken jedoch nicht vollständig gelöst; weiterhin existie-

ren Metriken, deren Wert beispielsweise eine zeitliche Dimension besitzt sowie andere Metriken, die ein dimensionsloses Ergebnis liefern. Um die Ergebnisse dieser Metriken über die zugeordneten Qualitätsmerkmale zu Qualitätskriterien zusammenfassen zu können ist es erforderlich, daß die Werte aller Qualitätsmerkmale ebenso wie die Werte der Qualitätskriterien dimensionslos sind. Benötigt wird also eine Methodik, wie man sowohl dimensionslose wie auch unterschiedlich dimensionierte Werte zu einem dimensionslosen Wert umrechnen kann.

Hierzu kann man sich die Charakteristika des zu untersuchenden Prozesses und dessen Gütekriterien selbst zu Nutze machen. Um Metriken sinnvoll anwenden zu können, wird grundsätzlich ein Vergleichswert benötigt. Dies kann einerseits eine vorgegebene obere oder untere Schranke für die Werte einer jeden Qualitätseigenschaft sein (zum Beispiel in Form der Aussage 'Jeder Bug soll innerhalb von 7 Tagen behoben werden'), andererseits können, wenn solche Aussagen nicht zur Verfügung stehen, die Werte eines als ideal empfundenen Projekts oder, sofern die zeitliche Entwicklung der Prozessgüte von Interesse ist, die Werte des gleichen Projekts zu einem früheren Zeitpunkt als Schranken herangezogen werden. Die Idee hinter diesem Ansatz besteht darin, den jeweiligen Qualitätsmerkmalen einen skalaren Wert zuzuweisen, der eine Aussage über die Einhaltung der vorgegebenen Schranken beinhaltet. Hierbei sind mehrere, je nach Anwendungsbereich unterschiedliche Vorgehensweisen zu betrachten:

Vergleich an vorgegebener Schranke: Der einfachste und aussagekräftigste Fall für die Ermittlung skalarer Werte für Qualitätsmerkmale anhand der berechneten Qualitätseigenschaften besteht aus einem Vergleich an einer fest vorgegebenen oberen oder unteren Schranke. Die Voraussetzung zur Anwendung dieses Verfahrens ist jedoch die Existenz fest definierter Minimal- bzw. Maximalwerte für jede Qualitätseigenschaft. Solche Richtwerte können beispielsweise aus den Anforderungen der ausführenden Organisation an den Prozess und die Güte seiner Einhaltung ermittelt werden.

Die Berechnung des skalaren Werts $f(m)$ eines Qualitätsmerkmals anhand des bei der Ermittlung des Wertes m für die zugeordnete Qualitätseigenschaft durch Anwendung der Metrik erfolgt nach dem Schema

$$f(m) = \begin{cases} 0 & \text{falls } m \leq S_o \\ 1 & \text{falls } m > S_o \end{cases}$$

für den Vergleich an der oberen Schranke S_o beziehungsweise

$$f(m) = \begin{cases} 0 & \text{falls } m \geq S_u \\ 1 & \text{falls } m > S_u \end{cases}$$

für den Vergleich an der unteren Schranke S_u. Mit diesem Vorgehen können direkt aus den Werten für Qualitätsmerkmale und Qualitätskriterien Aussagen der Form 'Die Güte des Merkmals bzw. Kriteriums x erfüllt die Vorgaben vollständig' getroffen werden, wenn der ermittelte Wert für das Merkmal oder Kriterium 0 beträgt. Werte im Intervall zwischen 0 und 1, die erst nach der Zusammenfassung von Qualitätsmerkmalen zu Qualitätskriterien auftreten können zeigen an, daß mindestens ein Teilaspekt

des betrachteten Kriteriums die Vorgaben nicht erfüllt. Beträgt der ermittelte Wert 1, so bedeutet dies daß die Güteziele dieses Kriteriums durchgängig verfehlt werden. Beginnend an der Wurzel des Baums können somit durch nähere Betrachtung der Teilaspekte der Qualitätskriterien die einen Wert größer 0 aufweisen sukzessive alle Qualitätseigenschaften ermittelt werden, die die Vorgaben des Prozesses nicht erfüllen.

Werden zu den zwei unterschiedlichen Zeitpunkten $t1$ und $t2$ jeweils Messungen vorgenommen die zu den Qualitätseigenschaften m_{t1} und m_{t2} führen, so ist mit Hilfe dieses Verfahrens auch eine Aussage zur Entwicklung der Prozessqualität im Verlauf der Zeit möglich. Gilt für ein gewähltes Qualitätskriterium x die Beziehung $f_x(m_{t1}) < f_x(m_{t2})$ so deutet dies auf eine Verschlechterung der Güte dieses Kriteriums hin während der umgekehrte Fall $f_x(m_{t1}) > f_x(m_{t2})$ auf eine Verbesserung der Güte zwischen den beiden Messungen schließen läßt. Dieses Verfahren läßt sich noch leichter anwenden, indem die Metrikwerte der Auswertung zum Zeitpunkt m_{t1} als (obere oder untere) Schranken für die Auswertung zum Zeitpunkt $t2$ herangezogen werden:

Vergleich anhand einer früheren Auswertung: Existieren zu einem gegebenen Prozess keine Vorgaben, die eine Auswertung nach der zuletzt genannten Methode ermöglichen, so ist es dennoch möglich, Aussagen über die zeitliche Entwicklung der Güte zu gewinnen. Hierzu sind mindestens zwei Messungen m_{t1} und m_{t2} zu den auseinanderliegenden Zeitpunkten $t1$ und $t2$ erforderlich. Möchte man nun ermitteln ob sich die Prozessgüte oder die Güte einzelner Kriterien zum Zeitpunkt $t2$ entgegen dem Zeitpunkt $t1$ verbessert oder verschlechtert hat, so kann man obiges Verfahren dahingehend modifizieren, daß als obere respektive untere Schranke der zweiten Messung die Werte der ersten Messung verwendet werden. Die Berechnungsvorschriften zur Ermittlung des skalaren Werts für ein Qualitätsmerkmal stellen sich dementsprechend folgendermaßen dar:

$$f(m_{t2}) = \begin{cases} 0 & \text{falls } m_{t2} \leq m_{t1} \\ 1 & \text{falls } m_{t2} > m_{t1} \end{cases}$$

sowie

$$f(m_{t2}) = \begin{cases} 0 & \text{falls } m_{t2} \geq m_{t1} \\ 1 & \text{falls } m_{t2} < m_{t1} \end{cases}$$

je nach dem ob es sich bei dem zum Zeitpunkt $t1$ ermittelten Wert um einen Maximal- oder Minimalwert handelt.

Die Interpretation der gewonnenen Ergebnisse weicht jedoch von der Interpretation der Ergebnisse beim Vergleich an festgelegten Schranken ab. Ein Wert $f_x(m_{t2}) = 0$ bedeutet in diesem Fall, daß das betrachtete Kriterium bei der zweiten Messung *mindestens so gut* abgeschnitten hat wie zum Zeitpunkt der ersten Messung, wohingegen ein Wert $f_x(m_{t2}) > 0$ darauf hindeutet, daß sich mindestens ein Teilkriterium im Vergleich zur ersten Messung verschlechtert hat. Aussagen sind bei dieser Methode also nur im Rahmen von Güteverschlechterungen möglich, da eine Güteverbesserung durch dieses Verfahren identisch wie eine unveränderte Güte behandelt wird. Um mit

diesem Verfahren auch Verbesserungen der Qualität erfassen zu können, muß das Berechnungsverfahren um eine weitere Option erweitert werden:

$$f(m_{t2}) = \begin{cases} -1 & \text{falls } m_{t2} < m_{t1} \\ 0 & \text{falls } m_{t2} \approx m_{t1} \\ 1 & \text{falls } m_{t2} > m_{t1} \end{cases}$$

beziehungsweise

$$f(m_{t2}) = \begin{cases} -1 & \text{falls } m_{t2} > m_{t1} \\ 0 & \text{falls } m_{t2} \approx m_{t1} \\ 1 & \text{falls } m_{t2} < m_{t1} \end{cases}$$

sofern es sich bei dem zuvor ermittelten Wert nicht um einen Maximal- sondern einen Minimalwert handelt.

Bei dieser Erweiterung wird dem Qualitätsmerkmal bei einer effektiven Verbesserung der Qualitätseigenschaft der Wert -1 zugewiesen, für vernachlässigbar kleine Abweichungen des Wertes für den zweiten Meßpunkt vom ersten Meßpunkt wird der Wert 0 vergeben und für effektive Verschlechterungen wird weiterhin der Wert 1 zugeordnet.

Die Interpretation der ermittelten Werte muß nun dahingehend ergänzt werden, daß der Wertebereich für die berechneten Werte eines Qualitätskriteriums bzw. -merkmals vom Intervall $[0, 1]$ auf das Intervall $[-1, 1]$ ausgeweitet wurde. Ein Wert nahe 0 besagt nun, daß sich das Qualitätskriterium im Vergleich mit der vorherigen Messung kaum verändert hat. Ein deutlich positiver Wert deutet auf eine Gesamtverschlechterung des Kriteriums hin während ein deutlich negativer Wert eine Verbesserung der Güte des Kriteriums darstellt.

Vergleich an einem Referenzprojekt: Der Vergleich an einem Referenzprojekts bietet sich dann an, wenn die Vorgabe des Prozesses lautet 'Die Prozessqualität eines jeden Projekts soll besser sein als die des Vorangegangenen' oder wenn bei einem für die Auswertung zur Verfügung stehenden, separaten Projekt festgestellt wurde, daß dessen Prozess qualitativ besonders hochwertig einzustufen war.

Die Auswertung wird in diesem Fall wie beim Vergleich mit einer früheren Auswertung des gleichen Projekts vorgenommen; lediglich wird als Referenzwert keine frühere Messung des beobachteten Projekts festgelegt, sondern eine beliebige Messung des als Referenz eingestuften Projekts. Stellt man beim Vergleich der aktuellen Messung mit dem Referenzprojekt erhebliche Diskrepanzen fest, so kann eine Analyse der Prozesse beider Projekte dazu beitragen, die Unterschiede beider Prozesse und deren Auswirkungen auf die Qualität des aktuellen Projekts zu ermitteln und daraus Schlüsse zur Verbesserung des aktuellen Prozesses zulassen.

Vergleich mehrerer gleichzeitig bearbeiteter Projekte: Wenn keine definierten Minimal- oder Maximalwerte, die zur Bewertung eines Prozesses herangezogen werden können zur Verfügung stehen und auch kein zum Vergleich geeignetes Referenzprojekt vorliegt, so kann unter bestimmten Bedingungen dennoch ein vergleichsba-

siertes Bewertungsverfahren herangezogen werden. Sofern zum gleichen Zeitpunkt mehrere ähnliche Projekte bearbeitet werden (dies können auch von unterschiedlichen Teams bearbeitete Subprojekte eines übergeordneten Projekts sein), so ist ein Vergleich der Werte zwischen diesen Projekten durchaus sinnvoll.

Voraussetzung hierfür ist jedoch, daß alle verglichenen Projekte dem gleichen Prozessmodell folgen (und somit auch das gleiche Qualitätsmodell für jeden dieser Prozesse anwendbar ist) und daß die verglichenen Projekte separat geleitet werden; an sonsten besteht die Gefahr, daß nicht vergleichbare Prozessaspekte verglichen werden oder daß durch ein gemeinsames Projektmanagement, das für alle betreuten Projekte den gleichen Führungsstil anwendet, die Unterschiede zwischen den einzelnen Projekten nicht mehr auf Unterschiede bei der Abarbeitung des Prozesses, sondern durch statistische Effekte zurückzuführen sind - aussagekräftige Feststellungen zu den Qualitätsunterschieden der verglichenen Projekte sind dann nicht mehr möglich.

Eine absolute Aussage zu den Prozessqualitäten ist, ebenso wie beim Vergleich eines Projektes mit sich selbst zu einem früheren Zeitpunkt, nicht möglich; eine sorgfältige Überprüfung der Vorgehensweisen in den einzelnen, zum Vergleich herangezogenen Projekten, kann jedoch wertvolle Aufschlüsse über den Zusammenhang zwischen den unterschiedlichen Vorgehensweisen und der daraus resultierenden Prozessqualität geben. Kombiniert man diese Vergleichsmethodik durch die im folgenden Abschnitt beschriebenen Kombination mehrerer Auswertungsverfahren mit einer zeitlichen Komponente, so lassen sich ebenfalls die Auswirkungen von Änderungen beim Management der Prozesse erfassen und auswerten.

Um die Vergleichswerte einer Zusammenfassung in einem Qualitätsmodell zugänglich zu machen, empfiehlt sich, ebenso wie bei den in den vorangegangenen Abschnitten beschriebenen Auswertungsverfahren eine Einteilung in Qualitätsklassen vorzunehmen. Bei wenigen ($n < 5$) Projekten, die zu vergleichen sind, bietet sich hier eine Einteilung in drei Qualitätsklassen analog dem Vergleich zu früheren Zeitpunkten an: Das Projekt, dessen Messung nahe am Durchschnitt aller Projekte liegt, wird als Referenzprojekt eingestuft und enthält den Wert $f(m) = 0$, Projekte die unterdurchschnittlich abschneiden erhalten den Wert $f(m) = 1$ und Projekte, deren Messwert besser als der Durchschnitt ausfällt erhalten den Wert $f(m) = -1$. Zu beachten ist hier jedoch, daß ein Vergleich sehr weniger Projekte durch statistische Effekte so stark verfälscht sein kann, daß die Aussagekraft der Auswertung nicht mehr gewährleistet ist. Vergleicht man mehr als 5 Projekte miteinander, kann eine feinere Unterteilung der Qualitätsklassen sinnvoll sein. Der Gedanke hinter diesem Vorgehen ist die Einteilung der Qualitätsklassen nach den verbalen Kriterien ‘durchschnittlich’, ‘gerade noch akzeptabel’ und ‘nicht mehr akzeptabel’ , je nach der Abweichung des Messwerts vom Durchschnitt. Um diese Einteilung vorzunehmen bieten sich verschiedene Berechnungsverfahren an; beispielhaft sei hier die Einteilung der Messwerte in Quartile erläutert, die auch bei der Auswertung des Eclipse-Projekts Anwendung finden wird.

Die Idee hinter diesem Verfahren beruht auf der Aufteilung der Menge M aller Messwerte m_i in annähernd gleichmächtige Teilmengen Q_1, Q_2, Q_3 und Q_4, sogenann-

ter *Qualitätsklassen*, anhand der Messwerte m_i. Bei einer Anzahl von 8 Messwerten ($|M| = 8$) erhält mal also 4 Teilmengen $Q_{1...4}$ mit jeweils 2 Elementen. Die Zugehörigkeit zu einer der Teilmengen ergibt sich aus der Güte des Messwerts. Die Teilmenge Q_1 enthält demnach die beiden besten Messwerte, die Teilmenge Q_4 die beiden schlechtesten. Je nach Zugehörigkeit des Messwertes zu einer dieser Teilmengen wird nun ein Qualitätswert zwischen -2 und 2 vergeben:

$$f(m) = \begin{cases} -2 & \text{falls } m \in Q_1 \\ -1 & \text{falls } m \in Q_2 \\ 1 & \text{falls } m \in Q_3 \\ 2 & \text{falls } m \in Q_4 \end{cases}$$

Bei dieser Methode fällt auf, daß sie zunächst nur anwendbar ist, wenn eine gerade Anzahl von Messwerten vorliegt ($|M| = 2n$). Sie ist aber ebenso anwendbar auf eine Messwertmenge mit ungerader Mächtigkeit ($|M| = 2n + 1$), indem für den Mittelwert (*Median*), der in diesem Auswertungsverfahren die Referenz darstellt, der Wert 0 vergeben wird:

$$f(m) = \begin{cases} -2 & \text{falls } m \in Q_1 \\ -1 & \text{falls } m \in Q_2 \\ 0 & \text{falls } m = \text{median}(M) \\ 1 & \text{falls } m \in Q_3 \\ 2 & \text{falls } m \in Q_4 \end{cases}$$

Die Interpretation dieses Auswertungsverfahren ergibt sich aus der Überlegung, daß sich die Qualität eines Projekts beim Vergleich mit anderen Projekten verbal mit Aussagen wie ‘weicht nur wenig vom Durchschnitt ab’ und ‘weicht stark vom Durchschnitt ab’ beurteilen läßt. Geht man davon aus, daß der Mittel- oder Durchschnittswert aller Projekte eine akzeptable Qualität darstellt, so führt die Anwendung dieser Auswertungsmethode auf Aussagen der Form ‘weicht nur leicht vom Mittelwert ab, ist also noch akzeptabel’ oder ‘weicht so stark vom Mittelwert ab, daß die Qualität des Projekts als nicht mehr akzeptabel angesehen werden kann’. Ein Beispiel für eine solche Auswertung liefert Abbildung 2.1.

Projekt	A	B	C	D	E	F	G	H	I
Messwert m	1	3	6	7,5	10	11	11,5	13	13
Quartil	Q_1		Q_2		-	Q_3		Q_4	
$f(m)$	-2	-2	-1	-1	0	1	1	2	2

Tabelle 2.1: Auswertungsbeispiel anhand von 9 Messwerten

Werden sehr viele Projekte miteinander verglichen so ist, um eine genauere Einstufung zu erhalten, auch eine Erhöhung der Anzahl der Qualitätsklassen Q_i möglich, wobei jedoch die in Abschnitt 2.1.4 beschriebenen Effekte berücksichtigt werden müssen.

Kombination mehrerer Methoden Selbstverständlich können die in den vorangegangenen Abschnitten beschriebenen Methodiken nicht nur isoliert angewandt, sondern auch miteinander kombiniert werden. Kombiniert man beispielsweise den Ver-

gleich früherer Messungen und den Vergleich an einem Referenzprojekt in zwei separaten Auswertungen, so kann man auch die zeitliche Veränderung beider Projekte in die Auswertung einbeziehen. Für das folgende Beispiel soll davon ausgegangen werden, daß für die Entwicklung des Projekts zwei Meilensteine M_1 und M_2 definiert wurden, zu denen in jedem Projekt eine Messung vorgeschrieben wird. Das Projekt P_1 soll als Referenz für das aktuell bearbeitete Projekt P_2 herangezogen werden.

Auswertung	M_1	M_2
Vergleich mit Referenz	-0.2	0.5
Zeitliche Änderung	$-$	-0.2

Tabelle 2.2: Beispielhafte Messwerte für das Projekt P_2 anhand der Referenz P_1

Vergleicht man nun den Wert für den gesamten Prozess (Tabelle 2.2) der beiden Projekte zu den Zeitpunkten M_1 und M_2 miteinander, so könnte man zu der Beobachtung kommen, die Prozessqualität des Projekts habe sich erheblich verschlechtert. Nimmt man nun aber zusätzlich den Vergleich der Werte des Projekts P_2 zum Zeitpunkt der Messung M_2 mit denen der Messung bei M_1 hinzu so fällt auf, daß sich das Projekt tatsächlich verbessert hat. Der zunächst beobachtete, bei alleiniger Betrachtung irreführende Effekt tritt dadurch auf, daß sich das Projekt P_1 im gleichen Zeitraum stärker verbessert und somit das Projekt P_2 in einem direkten Vergleich überholt hat. Derartige mögliche Fehlinterpretationen müssen bei der Wahl des Auswertungsverfahrens unbedingt berücksichtigt werden; ggf. ist in solchen Fällen eine Auswertung mit statischen Vergleichsdaten einer Auswertung mit dynamisch aktualisierten Vergleichswerten vorzuziehen.

Alle Auswertungsverfahren, die einen Vergleich unterschiedlicher Projekte über einen längeren Zeitraum bedingen sind demnach anfällig für mögliche Fehlinterpretationen, da sich der Betrachter stets darüber im Klaren sein muß, daß es sich um relative und nicht um absolute Angaben handelt. Besitzt man Kenntnis über die absolute Bewertung eines dieser verglichenen Projekte ist es möglich, daraus auch Schlüsse über die absolute Einstufung der verglichenen Projekte zu ziehen; fehlt ein solches Projekt mit absoluten Informationen jedoch zum Vergleich, so ist eine ausschließlich vergleichende Analyse möglich die über die absolute Prozessqualität keine Aussage treffen kann. Dennoch eignen sich solche Situationen zur qualitativen Analyse, da man unterschiedliche Prozessauslegungen der verglichenen Projekte analysieren und deren Unterschiede mit den unterschiedlichen Meßergebnissen in Bezug setzen kann um die Prozesse anzupassen und somit die Prozessqualität insgesamt zu steigern.

2.1.4 Aussagekraft der ermittelten Qualitätswerte

Nach der Auswertung einzelner Messwerte mit Hilfe eines der in den vorangegangenen Abschnitten beschriebenen Vorgehensweisen bieten sich nun mehrere Möglichkeiten an, die Qualität der Projekte einzustufen. Zunächst wäre es möglich, die mit Hilfe der Anwendung eines Auswertungsverfahrens aus den Messwerten m_i ermittelten Qualitätswerte $f(m_i)$ auf der Ebene der Qualitätsmerkmale miteinander zu vergleichen. Da

jedoch nicht davon auszugehen ist, daß einzelne Projekte auf der Ebene aller Qualitätsmerkmale ähnliche Vergleichswerte aufweisen, sind Aussagen, die zusammengefaßte Qualitätsmerkmale auf der Ebene der Qualitätskriterien betreffen aufgrund der Streuung der Qualitätswerte kaum sinnvoll möglich. Aus diesem Grund sollten zunächst die Werte der für den Prozess ermittelten übergeordneten Qualitätsmerkmale entsprechend dem Abschnitt 2.1.1 berechnet werden.

Als Ergebnis der Berechnung des Qualitätsmodells ergibt sich für jedes Qualitätskriterium ein Qualitätswert $f(m)$, der im gleichen Intervall liegt wie die zu den Qualitätsmerkmalen zugeordneten Qualitätswerte. Wurde beispielsweise eine Auswertung nach dem Verfahren des Vergleichs an einer früheren Auswertung angewandt, so liegen die Qualitätswerte der Qualitätskriterien im Intervall zwischen -1 und 1: $-1 \leq f(m) \leq 1$.

Bei einer unvorsichtigen Interpretation dieser Werte kann man hier leicht zu falschen Aussagen gelangen. So drängt sich im oben beschriebenen Fall die Interpretation auf, daß ein Wert $f(m) = 0$ bedeutet, daß das Projekt in allen dem betrachteten Kriterium zugeordneten Teilkriterien durchschnittlich abschneidet. Diese Interpretation ist jedoch nicht vollkommen korrekt, da eine Aussage hierzu lediglich für das betrachtete Kriterium Gültigkeit besitzt, nicht jedoch für die Teilkriterien. So würde die Berechnung eines Qualitätskriteriums mit zwei gleichermaßen gewichteten Teilkriterien, für die jeweils die Qualitätswerte $m_1 = 1$ und $m_2 = -1$ ermittelt wurden ebenfalls der Qualitätswert $f(m) = 0$ berechnet werden. Die Aussage, daß das Projekt im betrachteten Qualitätskriterium durchschnittlich abschneidet basiert lediglich darauf, daß das unterdurchschnittliche Abschneiden in einem der Teilkriterien durch ein überdurchschnittliches Abschneiden in einem anderen Teilkriterium kompensiert wird.

Auch durch die mögliche unterschiedliche Gewichtung von Teilkriterien ergeben sich mögliche Fehlinterpretationen, wenn die unterschiedliche Gewichtung nicht in die Überlegungen zur Interpretation einbezogen wird. So könnte man bei einem Qualitätswert $f(m) = 0,79$, der für ein Qualitätskriterium nach der Auswertungsmethode des Vergleichs an einer definierten oberen Schranke ermittelt wurde zu der Aussage ‘das Projekt erfüllt im betrachteten Kriterium die Vorgaben von 79% der Teilkriterien’. Diese Aussage ist jedoch nur dann korrekt, wenn alle Teilkriterien bis hin zu den Qualitätsmerkmalen gleichermaßen gewichtet wurden. Setzt sich das betrachtete Kriterium A aus den Teilkriterien A_1, A_2 und A_3 zusammen, wobei das Teilkriterium A_1 einfach, das Teilkriterium A_2 doppelt und das Kriterium A_3 dreifach gewichtet wird, so würde die Erfüllung der Vorgaben der Kriterien A_1 und A_2 bei gleichzeitiger Nichterfüllung des Kriteriums A_3 zu der Aussage ‘Das Projekt erfüllt das Kriterium A zu 66%’ führen. Der Messwert hingegen berechnet sich durch die Formel

$$f(m_A) = \frac{f(m_1) + 2 \cdot f(m_2) + 3 \cdot f(m_3)}{6}$$

zu

$$f(m_A)\frac{0 + 2 \cdot 0 + 3 \cdot 1}{6} = 0,5$$

Im Gegensatz dazu sind vergleichende Aussagen zwischen verschiedenen Projekten oder verschiedenen Qualitätskriterien des gleichen Projekts durch direkten Vergleich

der Qualitätswerte der Kriterien zulässig; Die Qualitätswerte für das Kriterium A mit $f(m_A) = 0,5$ und das Kriterium B mit $f(m_B) = 0,79$ führen völlig korrekt zu der Aussage 'Das Projekt erfüllt die Vorgaben für das Kriterium B besser als die Vorgaben für das Kriterium A'. In Anbetracht dessen, daß die Anwendung von Metriken darauf abzielt, Ansätze für Verbesserungen zu finden und Schwächen zu identifizieren, ist diese Art der Aussage vollkommen ausreichend.

Insbesondere bei der Anwendung vergleichsbasierter Auswertungsverfahren kann ein weiterer Effekt auftreten, der bei ungenauer Betrachtung zu Irritationen führen kann. Intuitiv würde man erwarten, daß die Summe der Qualitätswerte aller verglichenen Projekte an jeder beliebigen Stelle des Qualitätenbaums Null beträgt, da alle Projekte zusammen betrachtet exakt durchschnittlich abschneiden müßten; bei einer zusammenfassenden Auswertung aller Projekte wäre dies auch tatsächlich der Fall. Bei einer Einzelbetrachtung der Projekte ist dies jedoch in der Regel nicht so.

Die Verwendung von Quantilen zur Einteilung der Messwerte in Qualitätsklassen wird zusätzlich vermieden, daß einzelne Ausreißer das Gesamtergebnis durch statistische Effekte verfälschen. Liegen die Messwerte jedoch sehr dicht beieinander, so können die beschriebenen Effekte jedoch dennoch auftreten; dies trifft insbesondere dann zu, wenn mehrere Messwerte identisch sind. Ein Beispiel hierzu bietet Tabelle 2.3. Hier werden fünf Projekte miteinander verglichen, wobei die Messwerte von zwei Projekten identisch sind. Der Median berechnet sich zu 3, weshalb die Projekte B und C jeweils den Wert 0 für ihre Übereinstimmung mit dem Median erhalten. Hieraus ergibt sich eine Summe aller Qualitätswerte von 1 entgegen der Erwartung, daß die Summe über alle Qualitätswerte stets 0 betragen sollte.

Projekt	A	B	C	D	E
Messwert m	2	3	3	4	5
$f(m)$	-1	0	0	1	1

Tabelle 2.3: Bewertung bei identischen Messwerten

Dieser Effekt läßt sich in keinem Fall vollständig eliminieren, da er regelmäßig dann auftritt, wenn Messungen zu identischen Messwerten führen. Bei der Auswertung realer Prozesse kommt es häufig vor, daß die verwendete Metrik für mehr als einen Prozess den Wert 0 liefert, da das auszuwertende Kriterium auf diese Prozesse nicht zutrifft (Ein Beispiel hierzu wäre die Messung des Anteils an Bugs mit der höchsten Prioritätsstufe, wenn Projekte keinen Gebrauch von der Möglichkeit der Priorisierung einzelner Bugs machen). Abhilfe schafft hier lediglich die Korrektur des Qualitätsmodells auf die realen Prozessgegebenheiten oder die Anpassung der Auswertungsmethode für dieses Qualitätsmerkmal. Da der beschriebene Effekt jedoch nicht die Aussagekraft sondern lediglich die intuitive Bedeutung der Qualitätswerte beeinflußt, ist es in aller Regel nicht erforderlich, Maßnahmen gegen das Auftreten dieses Effekts zu ergreifen.

Bei der Einteilung der Messwertmenge M in mehrere Qualitätsklassen Q_i kann jedoch über den beschriebenen, eher als kosmetisch einzustufenden Effekt hinaus ein

erhebliches Problem auftauchen, wenn die Anzahl i der Qualitätsklassen Q_i größer gewählt wird als die Menge der (paarweise verschiedenen) Messwerte m_j. Sollen die 4 paarweise verschiedenen Messwerte aus Tabelle 2.3 in 5 Qualitätsklassen, die auf den Quartilen und dem Median der Messwertmenge M basieren eingeteilt werden, so müßten die Projekte B und C aufgrund ihrer Übereinstimmung mit dem Median den Qualitätswert 0 erhalten. Zur Bestimmung der vier Quartile Q_1 bis Q_4 stehen nun aber nur noch drei Messwerte zur Verfügung, wodurch sich die Frage stellt, in welche der Quartil-basierenden Qualitätsklassen die drei verbleibenden Messwerte einzuordnen sind. Eine Möglichkeit wäre die Einteilung des Projekts D in die Qualitätsklasse Q_3 und des Projekts E in die Klasse Q_4. Dennoch verbleibt für das Projekt A die Frage, ob es in die Qualitätsklasse Q_1 oder Q_2 eingeordnet werden soll, die sich nur beantworten läßt sofern man vorab eine Auflösungsvorschrift für solche Fälle definiert. Man könnte für derartige Messwerte beispielsweise festlegen, daß sie grundsätzlich in die dem Median am nächsten gelegene, in Frage kommende Qualitätsklasse (in diesem Fall also Q_2) eingestuft werden.

Um derartige Schwierigkeiten von vorne herein zu vermeiden sollte bei der Wahl der Anzahl der Qualitätsklassen darauf geachtet werden, daß deren Anzahl deutlich kleiner ist als die Anzahl der zu ermittelnden Messwerte ($i \ll j$). Als allgemeine Regel kann an dieser Stelle die Empfehlung ausgesprochen werden, höchstens halb so viele Qualitätsklassen einzuführen wie Meßwerte vorhanden sind ($2i \leq j$). Dadurch wird sichergestellt, daß jeder Qualitätsklasse (mit Ausnahme der Median-Klasse) im Normalfall 2 Messwerte zugeordnet werden; der oben beschriebene Effekt, daß Qualitätsklassen vollkommen leer bleiben, tritt somit erst auf, wenn die Messwerte von 4 Projekten identisch sind. Darüber hinaus übt die Wahl der Zahl der Qualitätsklassen nach dieser Empfehlung einen 'glättenden' Effekt auf die Auswertung aus: Zwei Projekte, die annähernd gleiche Messwerte besitzen, werden bei sukzessiven Auswertungen über die Zeit auch bei einer durch statistische Effekte begründeten Rangfolgenänderungen mit hoher Wahrscheinlichkeit in die gleiche Qualitätsklasse eingestuft, was bei einer sehr feinen Abstufung der Qualitätsklassen nicht der Fall wäre.

2.2 Identifikation von Qualitätsmodellen

Um aussagekräftige Metriken zu entwickeln und diese mit Hilfe eines Qualitätsmodells in einen sinnvollen Zusammenhang zu bringen ist es erforderlich, die Prinzipien und Ziele des zu vermessenden Prozesses zu identifizieren. Erst in diesem Zusammenhang läßt sich beurteilen, ob eine Metrik oder ein Aspekt des Qualitätsmodells tatsächlich zielführend ist, d.h. ob die Metrik oder der Aspekt für den betrachteten Prozess tatsächlich relevant ist. Basis für die Entwicklung eines Qualitätsmodells und der zugehörigen Metriken ist also immer eine explizite oder implizite Beschreibung des betrachteten Prozesses, wobei sich die Vorgehensweisen bei der Identifikation des Qualitätsmodells und der Ableitung entsprechender Metriken je nach Art des Prozesses und seiner Beschreibung unterscheiden können.

Laut [SL08] und [ED] entsprechen die (verfeinerten) Qualitätsaspekte den Zielen, die eine Organsiation für ihren Entwicklungsprozess vorgibt:

> The process quality characteristics of interest correspond to information needs that are in general derived from the objectives of the organization [SL08]

Diese Ziele werden, je nach Art des Prozesses, in einer Prozessbeschreibung erläutert und können direkt aus dieser abgeleitet werden. Bei agilen Prozessen, wie sie in der Open Source-Entwicklung häufig eingesetzt werden, enthält die Prozessbeschreibung oftmals jedoch nur eine sehr grobe Sicht auf die einzelnen Aspekte des Prozesses, die als solche nicht zur Identifikation feingranularer Qualitätskriterien oder gar Qualitätsmerkmalen geeignet ist. In einem solchen Fall bietet sich eine *Top-Down-Vorgehensweise* an, die basierend auf den grundlegenden Qualitätsmerkmalen und einer genauen Beobachtung der praktischen Anwendung des Prozesses so lange rekursiv feinere Qualitätskriterien identifiziert bis schließlich die Ebene der Qualitätsmerkmalen erreicht ist zu denen dann unmittelbar die zugehörigen Qualitätseigenschaften in Form von Metriken entwickelt werden können. Hierbei helfen auch spezifische Angaben zur Vorgehensweise in bestimmten Situationen, die vom Prozessinhaber bereit gestellt werden können. Ein Beispiel hierfür ist die Anleitung zur Vorgehensweise bei der Bearbeitung von Einträgen in der Bugzilla-Datenbank des Eclipse-Prozesses [EBU].

Andererseits ist es jedoch auch möglich, daß der Prozesseigentümer eine Reihe von Qualitätsmerkmalen vorgibt, die nicht zu Qualitätskriterien zusammengefaßt sind. In diesem Falle bietet sich die umgekehrte Vorgehensweise, ein *Bottom-Up-Ansatz* an. Basierend auf den vorgegebenen Qualitätsmerkmalen können diese sukzessive zu Qualitätskriterien zusammengefaßt werden um aus dieser Zusammenfassung eine grobere Sicht auf den Prozess zu ermöglichen. Hierbei spielt die Überlegung eine Rolle, welche zusätzlichen Aussagen aus der reinen Auswertung der Qualitätsmerkmale anhand der zugeordneten Qualitätseigenschaften zu gewinnen sind. Zwei Qualitätsmerkmale mit den Aussagen *Zeit bis zur ersten Bearbeitung eines Bugs* und *Gesamte Lebensdauer eines Bugs bis zur Behebung* können beispielsweise zu einem Qualitätskriterium mit der Aussage *Bearbeitungsgeschwindigkeit von Bugs* zusammengefaßt werden.

Zu beachten ist in beiden Fällen, daß sich die identifizierten Qualitätsmerkmale und -kriterien mit Hilfe der vorliegenden Informationen, im Falle dieser Arbeit mit Hilfe der in der Bug-Datenbank erfassten Daten, aussagekräftig messen lassen. Beim Vergleich zwischen verschiedenen Projekten oder Prozessen ist darüberhinaus sicherzustellen, daß die Qualitätsmerkmale auf einen Großteil der beobachteten Projekten bzw. Prozessen anwendbar sind, um einen sinnvollen Vergleich zu ermöglichen. Sollte dies nur auf einen Teil der betrachteten Projekte zutreffen müssen darüberhinaus Vorkehrungen getroffen werden um die durch Nichtanwendbarkeit einzelner Metriken auf bestimmte Projekte entstehenden Probleme aus den vorangegangenen Abschnitten zu umgehen - beispielsweise durch Festlegung einer Auflösungsregel wie in Abschnitt 2.1.4 beschrieben.

Bei Bedarf können beide Ansätze auch kombiniert werden. Die Qualitätsmerkmale eines Qualitätsmodells, das mit Hilfe des *Top-Down-Ansatzes* entwickelt wurde können mit Hilfe des *Bottom-Up-Ansatzes* zu einem neuen Qualitätsmodell zusammengefaßt werden, das dann eine differenzierte Sicht auf den betrachteten Prozess ermöglicht. Hierdurch können beispielsweise die Bedürfnisse verschiedener Interessengruppen berücksichtigt werden ohne befürchten zu müssen daß durch wahlloses Entwickeln von Qualitätsmerkmalen die Aussagekraft des Qualitätsmodells verloren geht.

Sobald das (noch ungewichtete) Qualitätsmodell vollständig identifiziert wurde, müssen die Kantengewichte des Qualitätenbaums festgelegt werden. Wie bereits bei der Identifikation des Modells selbst dient hierbei in erster Linie die Prozessbeschreibung als Anhaltspunkt für die Gewichtung der einzelnen Qualitätsmerkmale und Qualitätskriterien zueinander. Wenn die Prozessbeschreibung bestimmte Kriterien stärker hervorhebt als andere so sind diese im Qualitätsmodell entsprechend stärker zu gewichten. Darüber hinaus stellt auch die subjektive Aussagekraft einzelner Qualitätseigenschaften einen Anhaltspunkt zur Gewichtung bereit; eine Metrik, die vielen Störfaktoren unterworfen ist oder die von mehreren Prozessaspekten beeinflußt wird ist geringer zu gewichten als eine, die eine sehr eindeutige Aussage in Hinsicht auf das bei der Entwicklung der Metrik definierte Ziel zu treffen in der Lage ist. Bei dieser Einstufung hilft auch der Vergleich der Ergebnisse die ein Qualitätsmodell liefert mit den bislang gemachten Erfahrungen bei der Ausführung des Prozesses, sofern solche vorliegen. Gegebenenfalls müssen die Gewichtungen der einzelnen Qualitätskriterien und -merkmale iterativ angepaßt werden bis sich eine Übereinstimmung zwischen den Ergebnissen des Qualitätsmodells und den Erfahrungswerten einstellt. Bei dieser Vorgehensweise muß jedoch berücksichtigt werden, daß subjektive Qualitätseindrücke durchaus erheblich von der objektiv meßbaren Qualität eines Produkts abweichen können; in solchen Fällen ist es keineswegs zulässig, die in der Prozessbeschreibung geforderten Qualitätsmerkmale dahingehend abzuwandeln, daß sie mit der subjektiven Wahrnehmung der Entwickler übereinstimmen; dieses Vorgehen käme einem 'Hineinmessen' von Qualität in ein Produkt, das diese Qualität objektiv nicht besitzt.

2.3 Quality Benchmark Levels

In vielen Szenarien, in denen eine Auswertung und Vermessung von Qualitätskriterien erwünscht ist, existiert eine eindeutige Priorisierung bestimmter Qualitätsmerkmale; so könnte ein Prozess beispielsweise fordern, daß zunächst eine definierte Menge grundlegender Qualitätskriterien zu optimieren ist, bevor die nächste Gruppe von Qualitätskriterien erfüllt werden soll. Dieser Ansatz entspricht einer Art 'Roadmap' mit Hilfe derer das Ziel möglichst hoher Qualität schrittweise erfüllt werden soll. Einen vergleichbaren Ansatz verfolgt die *Capability Maturity Model Integration* [Car].

Sofern derartige Forderungen vorliegen, empfiehlt sich zur Einstufung von Projekten ebenfalls ein stufenbasierter Ansatz beispielsweise in Form sogenannter *Quality Benchmark Levels* (*QBLs*), die in [SSM06] beschrieben werden. Der Ansatz basiert auf dem Gedanken, daß zunächst Mindestanforderungen in einer begrenzten Gruppe von

Qualitätskriterien zu erfüllen sind; während diese Anforderungen pro *QBL* schrittweise verschärft werden, kommen jeweils neue Anforderungen hinzu. Ein *Quality Benchmark Level* gilt jeweils genau dann als erfüllt, wenn die Anforderungen sämtlicher niedrigerer *QBLs* und die Anforderungen des zu erfüllenden *QBL* eingehalten werden.

Auch wenn exakte Vorgaben zur Priorisierung von Qualitätsmerkmalen fehlen, kann durch die Betrachtung der Auswertung erkannt werden, ob ein Prozess bzw. dessen Anwendung einer Klassifizierung mit Hilfe von *Quality Benchmark Levels* zugänglich ist; dies ist dann der Fall, wenn die ermittelten Erfüllungsgrade mehrerer miteinander verglichener Projekte bzw. Zeiträume echte Teilmengen voneinander darstellen:

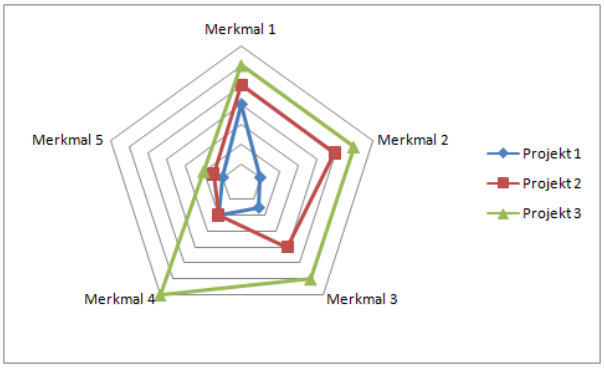

Abbildung 2.3: Gruppe von Projekten, die einer Auswertung mit Hilfe von *QBLs* zugänglich sind

Abbildung 2.3 zeigt eine Gruppe von Projekten, die einer Auswertung mit Hilfe von *QBLs* zugänglich sind; die Erfüllungsgrade der einzelnen Projekte sind ineinander verschachtelt, so daß steigende Qualität eindeutig durch eine Vergrößerung der Fläche innerhalb des Diagramms erkennbar ist.

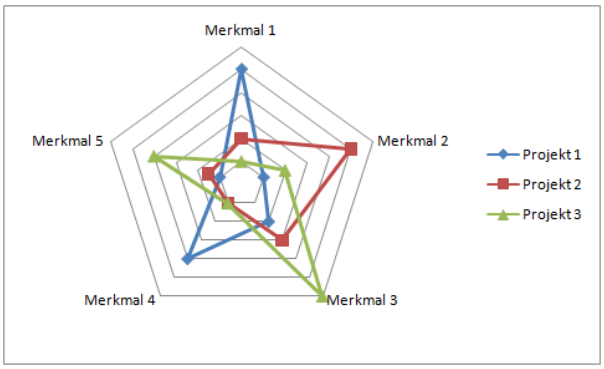

Abbildung 2.4: Gruppe von Projekten, die sich einer Auswertung mit Hilfe von *QBLs* entziehen

Die Projekte in Abbildung 2.4 hingegen entziehen sich einer sinnvollen Auswertung durch *Quality Benchmark Levels*, da die drei im Diagramm vorhandenen Projekte of-

fenbar unterschiedlichen Wert auf die Erfüllung bestimmter Qualitätsmerkmale legen. Würde man hier ohne in der Prozessbeschreibung klar definierte Prioritätsverteilung versuchen, eine Priorisierung einzuführen, so würde dies dazu führen, daß alle Projekte lediglich ein vergleichsweise geringes *QBL* erfüllen, wohingegen die subjektive Auffassung der Projektbeteiligten ein differenzierteres Bild ergeben könnte.

Kapitel 3

Merkmale der Open Source-Entwicklung

Zwischen den Vorgehensweisen bei der Entwicklung klassischer Softwareprojekte und den bei der Bearbeitung von Open Source-Projekten angewandten Methoden existieren teils erhebliche Unterschiede, die bei der Bewertung der jeweiligen Prozesse unbedingt berücksichtigt werden müssen. Dieses Kapitel stellt einige wesentliche Unterschiede heraus, die Einfluß auf die Vorgehensweise bei der Beurteilung eines Prozesses aus dem Open Source-Umfeld haben.

3.1 Strukturelle Unterschiede

In der allgemeinen Wahrnehmung haben Open Source-Projekte im gegensatz zu klassischen kommerziellen Projekten oft den Ruf, chaotisch und unorganisiert zu sein, da sie hauptsächlich von 'Freizeit-Hackern' betreut und vorangebracht würden:

> It is a common misconception to think of "open source" as a purely chaotic process which results in a — more or less — usable software product, created by spare time hackers who do not share the same maintainability concerns as their "professional" or academic counterparts. [BP02]

Wie dieses Zitat verdeutlicht, handelt es sich bei dieser einseitigen Wahrnehmung jedoch um eine Fehleinschätzung. Sicherlich existiert eine unzählige Menge von Open Source-Projekten die diesem Klischee gerecht werden; insbesondere größere Projekte, die aufgrund ihres Nutzens in der Industrie besondere Beachtung erlangen, werden jedoch in den meisten Fällen von großen Unternehmen sowohl finanziell als auch mit Arbeitskraft unterstützt. Die Erfahrungen dieser Unternehmen fließen somit nicht nur in die technischen Aspekte der Entwicklung, sondern auch in die Entwicklungsprozes-

se ein, da es durchaus im Interesse der unterstützenden Firmen liegt, die Entwicklung eines qualitativ hochwertigen und wartbaren Produkts zu unterstützen:

> Even “free speech” projects like GCC, or GNU/Linux are driven in large parts by the financial involvement of major companies (e. g. IBM, Red Hat) who assign skilled developers and other monetary resources to software products where everyone can read and modify the source code. The outcome, of course, must be a maintainable product in one way or another, because maintainability is a major reason why “dinosaur” projects like GCC, *BSD, Emacs, GNU/Linux, etc. still play an important role today, each in their individual fields. [BP02]

Daß Entwicklungen im Open Source-Umfeld grundsätzlich chaotisch, unstrukturiert und nicht zielgerichtet ablaufen trifft also grundsätzlich, insbesondere im Rahmen der großen Open Source-Projekte, keinesfalls zu. Dennoch existieren teils erhebliche strukturelle Unterschiede zu klassischen Entwicklungsprozessen, die größtenteils eine direkte Folge des offenen Charakters der Open Source-Projekte sind.

In einem klassischen Software-Projekt wird üblicherweise zwischen den Rollen ‘Entwickler’ und ‘Kunde’ unterschieden wobei die Kunden nicht unmittelbar am Entwicklungsprozess beteiligt sind sondern ihre Wünsche und Anregungen einer Vertriebsabteilung mitteilen, die dann über die Einbeziehung der entsprechenden Funktion in das endgültige Produkt entscheidet. Im Umfeld der quelloffenen Software-Entwicklung verschmelzen diese Rollen hingegen stark. Üblicherweise wird in solchen Projekten zwischen der Anwender-Gemeinde (oft *User Community* oder einfach nur *Community* genannt) und der Entwickler-Gemeinde (je nach Grad der Beteiligung am Projekt meist *Contributors* oder *Committers* bezeichnet) unterschieden, zwischen denen jedoch keine klaren Grenzen existieren. So fordert beispielsweise die Beschreibung des Entwicklungsprozesses des Eclipse-Projekts [GW05], daß die Entwickler des Projekts gleichzeitig Anwender ihres eigenen Produkts sind (‘*Consume your own Output*’). Umgekehrt steht es jedem Anwender frei, seine eigenen Beiträge in die Entwicklung des Projekts einzubringen. Auf die Charakteristika dieser beiden Gruppen wird in den folgenden Abschnitten näher eingegangen.

3.2 Die Entwickler-Gemeinde

Die Entwickler-Gemeinde eines Open Source-Projekts besteht in der Regel aus denjenigen Personen, die direkte Beiträge zur Entwicklung des Projekts und seiner Artefakte (z.B. der vom Projekt hergestellten Software-Produkte) leisten. Diese Gruppe umfaßt das Projektmanagement, die Entwickler sowie die Autoren verschiedener Dokumente, die im direkten Zusammenhang mit dem Entwicklungsprojekt stehen (beispielhaft zu nennen wären hier die Autoren von Benutzerhandbüchern oder Webdokumenten). Da aufgrund der offenen Gestaltung solcher Projekte prinzipiell jede beliebige Person als Mitglied der Entwickler-Gemeinde tätig werden kann, ist die Struktur der Ent-

wicklergemeinde ständigen Änderungen unterworfen; da der fachliche und persönliche Hintergrund der Mitglieder oft stark differiert oder nicht bekannt ist, zeichnet sich die Entwickler-Gemeinde darüber hinaus durch eine starke Heterogenität aus. Die Mitglieder der Entwickler-Gemeinde eines großen Open Source-Projekts stammen aus vielen unterschiedlichen Unternehmen und geographischen Regionen, so daß einerseits der fachliche Hintergrund stark variiert (eine Tatsache, die sowohl Vor- als auch Nachteile bietet), andererseits unter Umständen aber auch erhebliche Sprachbarrieren zu überwinden sind. Das Eclipse-Projekt zählt zum momentanen Zeitpunkt (November 2008) 588 aktive sogenannte *Committer*, die insgesamt 61 unterschiedlichen Organisationen angehören; 146 der Committer hingegen gehören keinerlei Organisation an sondern tragen individuell und auf eigene Initiative zum Projekt bei [Das]. Zusätzlich zu diesen 588 aktiven Committern zählt das Eclipse-Projekt 419 Mitglieder der Entwickler-Gemeinde, die sich aus der aktiven Unterstützung des Projekts zurückgezogen haben. Neben diesen *Committern* existiert jedoch noch eine nicht erfaßte Anzahl sogenannter *Contributors*, die zwar Beiträge zum Projekt geleistet haben, jedoch nur lose in die Abläufe des Projekts eingebunden sind.

Aus diesen Beobachtungen ergeben sich einige wichtige Erkenntnisse zur Struktur eines Open Source-Entwicklungsteams und zu den besonderen Anfordernissen, die an die Entwicklung eines Open Source-Projekts gestellt werden müssen.

Während in einem klassischen Entwicklungsprojekt eine überschaubare Anzahl von Entwicklern arbeitet, die weitestgehend den selben fachlichen Hintergrund besitzen und sich in den meisten Fällen in unmittelbarer geographischer Nachbarschaft befinden, besitzt das Entwicklungsteam eines Open Source-Projekts eine breit gestreute fachliche Kompetenz, einen großen personellen Umfang, sehr unterschiedliche Hintergründe und ist mitunter geographisch weit gestreut. Hieraus ergeben sich gegenüber einem eingespielten Entwickler-Team eines klasssischen Entwicklungsprojekts einige Chancen, jedoch sind auch gewisse Probleme abzusehen, die bei der Planung des Projekts unbedingt berücksichtigt werden müssen. Die auch durch die Zugehörigkeit zu verschiedenen Organisationen bedingten unterschiedlichen Gewohnheiten der einzelnen Mitglieder bei der Bearbeitung eines Projekts müssen mit Hilfe einer sinnvollen Projektbeschreibung vereinheitlicht werden; dies kann unter anderem beispielsweise durch die Bereitstellung entsprechender Werkzeuge erreicht werden, die eine Koordination der Entwicklungsaktivitäten fördern. Hierzu zählen insbesondere entsprechende Dokumentations-Systeme (z.B. Wikis), jedoch tragen auch gut funktionierende *Change Management*- oder *Change Request Management*-Systeme zu einer Abstimmung und Koordination der Entwicklergemeinde bei.

Durch den offenen Charakter eines jeden Open Source-Projekts ist es darüberhinaus jeder Person unabhängig von ihrem eigenen fachlichen Kenntnisstand gestattet, Beiträge zum Projekt zu leisten. Zusätzlich zu den Auswirkungen, die dieses Charakteristikum auf die Struktur des Entwickler-Teams hat (Mitglieder kommen ständig hinzu während andere ständig wegfallen) ist kaum absehbar, wie sich der Beitrag eines unerfahrenen Projekt-Mitglieds auf die Struktur des Gesamtprojekts auswirkt. Um eventuelle negative Auswirkungen die hieraus resultieren zu vermeiden, befolgt das Eclipse-Projekt einen strengen mehrstufigen Prozess [EDP08] mit Hilfe dessen Beiträge neuer *Contri-*

butors geprüft und bewertet werden, bevor diese Beiträge tatsächlich in die Projekte integriert werden. Dieser Mechanismus hat sich offenbar bewährt; in einer Umfrage [Law07] gaben lediglich 4% derjenigen befragten Personen, die bereits Beiträge zum Eclipse-Projekt geleistet haben, an, daß ihre Beiträge ohne Angabe eines sinnvollen Grundes abgelehnt worden seien; 73% hingegen stellten fest, daß ihre Beiträge akzeptiert und anerkannt wurden.

Ein weiteres Charakteristikum, das bei der Planung eines Open Source-Projekts unbedingt berücksichtigt werden muß ist die Tatsache, daß niemand streng verpflichtet ist, sich an der Entwicklung zu beteiligen. Jeder Entwickler (bzw. jede Organisation, die sich finanziell oder durch Bereitstellung von Arbeitskräften am Projekt beteiligt) kann seine Unterstützung zu jedem beliebigen Zeitpunkt zurückziehen. Entscheidet nun eine größere Gruppe von Entwicklern das Projekt nicht mehr zu unterstützen kann dies zum vollständigen Scheitern des Projekts führen, sofern nicht Maßnahmen getroffen wurden, die ein solches Ereignis kompensieren können. Der Eclipse-Prozess fordert daher für alle beteiligten Projekte einen ständigen Ausbau seiner Entwicklergemeinde [EDP08]. Besonders schwere Probleme in diesem Zusammenhang können dann auftreten, wenn sich besonders aktive Organisationen unerwartet aus der Entwicklung zurückziehen. So stammen beispielsweise mehr als die Hälfte aller im Jahr 2008 zu Eclipse beigetragenen Codezeilen von den beiden aktivsten am Projekt beteiligten Organisationen *IBM* und *STARorganisation* (nicht zu einer Organisation gehörende Entwickler wurden hierbei nicht berücksichtigt), während die 57 übrigen Organisationen, die Beiträge zum Code des Projekts leisten, lediglich auf 43% der beigetragenen Codezeilen kommen.

Jeder einer Open Source-Entwicklung zugrunde liegende Prozess muß diese speziellen Gegebenheiten unbedingt berücksichtigen um eine erfolgreiche und produktive Entwicklung des Projekts sicher zu stellen.

3.3 Die Anwender-Gemeinde

Die Anwender-Gemeinde eines Open Source-Projekts besteht hauptsächlich aus den Benutzern des im Rahmen des Projekts entwickelten Produkts. Im Gegensatz zu klassischen Software-Entwicklungsprozessen sind die Anwender hier jedoch unmittelbar an der Entwicklung des Projekts beteiligt – sie stehen als ‘Kunden’ des Projekts im Mittelpunkt der Entwicklung der Open Source-Produkte. Was im Rahmen des Eclipse-Prozesses als ‘*Always have a Client*’ bezeichnet wird läßt sich allgemein so formulieren, daß innerhalb eines Open Source-Projekts die Anforderungen der Anwender inkrementell und sehr feingranular umgesetzt werden sollen. Ähnlich wie bei kommerziellen, nicht offenen Entwicklungsprojekten steht und fällt der Erfolg eines Open Source-Projekts mit der Zufriedenheit seiner Anwender; liefert ein Produkt nicht die gewünschte oder benötigte Funktionalität so sinkt die Akzeptanz der Anwender und das Projekt erfüllt seinen Zweck nicht mehr. Aus diesem Grund wird unter dem Stichwort ‘Offenheit’ im Rahmen der Open Source-Entwicklung insbesondere auch eine starke Einbeziehung der Anwendergemeinde in die Entwicklung gefordert; bei Eclip-

se wird diese Forderung durch den Begriff *Community Involvement* [GW05] konkretisiert.

Da bei derartigen Projekten eine direkte Kommunikation zwischen Entwicklern und Anwender-Gemeinde erforderlich ist (im Gegensatz zu vielen kommerziellen Projekten existiert hier keine zwischengeschaltete Instanz, die vermittelnd zwischen Anwendern und Entwicklern wirken könnte), müssen die Modalitäten zur Kommunikation zwischen Entwicklern und Anwendern genau definiert werden.

3.4 Unterschiede bei der Prozess-Gestaltung

> Usually, the only constant in a free software project is constant change. [BP02]

Wie die Beschreibungen in den vorangegangenen Abschnitten und dieses Zitat zeigen, ist stetige Änderung ein Hauptcharakteristikum der Open Source-Entwicklung: Das Entwickler-Team ist ständigen Änderungen unterworfen und die Zielsetzung und der Umfang des Projekts kann sich, basierend auf den Rückmeldungen der Anwender-Gemeinde, ebenfalls kontinuierlich ändern. Dieser Charakter eines ständig im Umbruch befindlichen Projekts erfordert neben flexiblen Projektbeteiligten auch einen flexiblen Prozess, der jederzeit auf veränderte Randbedingungen anpassbar ist.

Aus diesem Grund setzen die meisten Open Source-Projekte agile Software-Entwicklungsprozesse ein, die sehr stark auf die individuelle Verantwortung der einzelnen Projektbeteiligten setzen. Anstatt genaue Vorgaben an die Vorgehensweisen zur Erreichung eines bestimmten Ziels zu machen spezifizieren diese Prozesse lediglich diese Ziele, während die Methodik zum Erreichen eines jeden Ziels den Mitarbeitern des Projekts selbst überlassen wird:

> This document imposes requirements and constraints on the operation of the Projects, and it does so on behalf of the larger Eclipse community. It is an explicit goal of the Development Process to provide as much freedom and autonomy to the Projects as possible while ensuring the collective qualities benefit the entire Eclipse community. [...] Part of the strength of this document is in what it does not say, and thus opens for community definition through convention, guidelines, and public consultation. A document with too much structure becomes too rigid and prevents the kind of innovation and change we desire for Eclipse. In areas where this document is vague, we expect the Projects and Members to engage the community-at-large to clarify the current norms and expectations. [EDP08]

Nur durch diese Offenheit des Entwicklungsprozesses ist die Möglichkeit gegeben, angemessen auf unvorhergesehene Veränderungen zu reagieren und ein optimales Umfeld für ein jedes (Teil-)Projekt zu schaffen. Nachteilig an dieser Vorgehensweise ist

jedoch, daß die konkreten Vereinbarungen zum Erreichen dieses Ziels, die für jedes (Teil-)Projekt individuell gestaltet werden können, mitunter zwischen den Projekten stark variieren können, was unter anderem eine Auswertung des Prozesses mit definierten Metriken oder die Identifikation von Qualitätskriterien zur Vermessung der Prozesse erschwert. Auch der Wechsel eines Entwicklers von einem Projekt zu einem Schwesterprojekt ist für ihn möglicherweise mit einer erheblichen Umstellung in seiner Arbeitsweise verbunden, die geeignet ist, seine Produktivität zumindest zeitweise in erheblichem Maße zu verringern.

Kapitel 4

Metriken auf Bug-Tracking-Datenbanken

Die in dieser Arbeit betrachteten ‘Software-Prozessdaten’ beschränken sich ausschließlich auf Informationen, die aus den in einer Bug-Datenbank gespeicherten Daten ermittelt werden können. Hinsichtlich dieser Daten existieren jedoch einige Einschränkungen und Besonderheiten, die in diesem Kapitel diskutiert werden. Darüber hinaus wird die Vorgehensweise bei der Entwicklung von Metriken mit Hilfe des Werkzeugs *BugzillaMetrics* beschrieben.

4.1 Wesen der Information in einer Bug-Tracking-Datenbank

Eine Bug-Tracking-Datenbank enthält, einige wenige Anwendungsbereiche ausgenommen, in erster Linie Daten zu im Rahmen des Tests und des Betriebs einer Anwendung gefundener Fehler sowie Erweiterungswünsche, die entweder von den Benutzern des Produkts oder von den Entwicklern als Gedankenstütze eingestellt werden. Nicht selten wird das Bug-Tracking-Werkzeug auch zur Kommunikation zwischen den Projektbeteiligten und zur Diskussion beispielsweise über den Sinn gewünschter Erweiterungen verwandt.

Aus diesen Eigenschaften ergeben sich vielfältige Analysemöglichkeiten, jedoch bestehen selbstverständlich auch Grenzen hinsichtlich der Aussagekraft und dem Umfang der in der Bug-Datenbank vorgehaltenen Information.

Ein Eintrag in einer Bugzilla-Datenbank beinhaltet neben den Daten zur Projektzugehörigkeit Informationen zu Ersteller und Bearbeiter eines Fehlerberichts, zum Zustand, in dem sich der Fehler aktuell befindet sowie verschiedene weitere Daten, die beispielsweise die Priorität oder Schwere des Fehlers speichern. Im Falle einer Bugzilla-Datenbank wird zu jedem Eintrag darüber hinaus noch ein Änderungs-Logbuch

geführt, in dem die Entwicklung eines Fehlerberichts seit seiner Erstellung genau zurückverfolgt werden kann.

Trivialerweise läßt sich anhand dieser Eigeschaften schließen, daß sich durch Anwendung von Metriken auf Bug-Datenbanken das Verhalten des Entwicklungsteams bei der Abarbeitung von Fehlerberichten und Erweiterungswünschen auswerten läßt. Diese Aussage trifft jedoch, in dieser vereinfachten Variante, nicht zu. Sinnvoll und aussagekräftig messbar sind ausschließlich diejenigen Aspekte eines Entwicklungsprozesses, die direkte Auswirkungen auf die Struktur der Bug-Datenbank haben. Insbesondere zählen hierzu natürlich Vorgaben bezüglich der korrekten Verwendung des Bug-Tracking-Werkzeugs. Nicht meßbar durch ausschließliche Betrachtung einer Bug-Datenbank sind Aspekte, die sich nicht oder nur unvollständig in der Bug-Datenbank wiederspiegeln, hier wären beispielsweise Vorgaben zur korrekten Verwendung eines Versionskontrollsystems oder Anforderungen an den Programmierstil zu nennen.

Selbst Aspekte, die sich unter bestimmten Umständen durch ausschließliche Betrachtung von Einträgen einer Bug-Tracking-Datenbank vollständig messen ließen, besitzen häufig nicht die Aussagekraft, die man bei oberflächlicher Betrachtung des Meßverfahrens erwarten könnte; so wäre beispielsweise eine Metrik, die die Kommunikationsbereitschaft der Projektentwickler vermessen soll indem die Anzahl der Kommentare bei den Bug-Einträgen gemessen werden, spätestens dann nicht voll aussagekräftig, wenn die Entwickler ebenfalls Diskussionen über E-Mail, Telefon oder andere Kommunikationsmittel und nicht ausschließlich über das Bug-Tracking-Werkzeug führen.

Die isolierte Betrachtung von ausschließlich auf Basis einer einzelnen Sammlung von Prozessdaten gewonnenen Erkenntnissen ist somit nie aussagekräftig genug, um ein absolutes und vollständiges Urteil, das alle Aspekte und Kriterien des Prozesses abdeckt, über den zu vermessenden Prozess zu fällen; hingegen kann die Kombination einer solchen Messung mit weiteren Auswertungen ein durchaus angemessenes und vollständiges Bild der Prozessqualität liefern, sofern alle angewandten Meßverfahren die für ihren Bereich relevanten Randbedingungen und Möglichkeiten berücksichtigen. Derartige Kombinationsmöglichkeiten werden in Abschnitt 6.1 noch erläutert.

4.2 Randbedingungen

Um Metriken auf Daten, die in einem Bug-Tracking-Werkzeug erfaßt wurden, sinnvoll anwenden zu können, müssen einige Randbedingungen berücksichtigt werden. Diese Randbedingungen betreffen sowohl die Anwendbarkeit bestimmter Metriken auf einen speziellen Prozess, die Referenzdaten wie auch den Prozess selbst.

Möchte man aussagekräftige Messergebnisse haben, existieren zunächst einige Anforderungen an die Nutzung des Bug-Tracking-Werkzeugs selbst. Ein Prozess, der die Nutzung des Werkzeugs nicht explizit vorschreibt und genaue Angaben zur korrekten Verwendung macht, wird sich kaum aussagekräftig mit Hilfe der Bug-Datenbank vermessen lassen. Wenn es jedem Entwickler freigestellt ist, ob und auf welche Weise

er die Bug-Datenbank nutzt, fehlen die Grundlagen, um sinnvolle und aussagekräftige Metriken zur Vermessung zu identifizieren; wendet man auf einen solchen Prozess einen Satz von Standard-Metriken an, so läßt sich im besten Falle das individuelle Verhalten und die Gewohnheiten der Entwickler, nicht jedoch die Qualität der Einhaltung des Prozesses bestimmen. Eine klare, eindeutige und vollständige Vorgabe zum Umgang mit dem im Rahmen des Prozesses eingesetzten Bug-Tracking-Werkzeugs ist also unabdingbar für die sinnvolle Vermessung des Prozesses auf Basis von Bug-Tracking-Daten.

Eine weitere wichtige Voraussetzung für die korrekte Interpretation der Messwerte ist eine solide Vergleichswertbasis. Da die reinen Messwerte, die durch die Anwendung von Metriken auf den zu betrachteten Prozess isoliert betrachtet höchstens geringe Aussagekraft haben, ist es notwendig, zur Interpretation Vergleichswerte heranzuziehen, an die ebenfalls einige Voraussetzungen gestellt werden müssen. Da sich, im Gegensatz zur Qualität eines Produkts, die meisten Prozesse grundlegend in ihrer Zielsetzung und den Methoden, mit denen diese Zielsetzung erreicht werden sollen, unterscheiden, muß die Vergleichswertbasis anhand von Prozessen gebildet werden, die die gleiche Zielsetzung besitzen und die gleichen Vorgaben zur Erreichung des Ziels treffen; sind solche Vergleichsdaten nicht verfügbar, können im Notfall auch Daten eines lediglich ähnlichen Prozesses herangezogen werden, was sich jedoch in jedem Fall negativ auf die Aussagekraft der ermittelten Ergebnisse auswirkt. Ein einfaches Mittel, diese Voraussetzung einer korrekten Vergleichsbasis zu erfüllen ist das Heranziehen früherer Messwerte des gleichen Prozesses als Vergleichsbasis; auch ein Vergleich zwischen mehreren gleichzeitig ablaufenden Prozessen, die der gleichen Prozessbeschreibung folgen, ist möglich. Insbesondere bei agilen Prozessen, die ständigen Änderungen und Anpassungen unterworfen sind, muß jedoch darauf geachtet werden, daß sich die Prozessbeschreibung des aktuell betrachteten Prozesses nicht wesentlich geändert hat seit die Vergleichsdaten ermittelt wurden; insbesondere im Open Source-Umfeld sind derartige Prozesse häufig anzutreffen.

Eine weitere Einschränkung ergibt sich im Zusammenhang mit dem Vergleich zwischen verschiedenen Projekten. Differiert die Projektgröße zwischen dem aktuell betrachteten Projekt und dem zur Bildung der Vergleichsbasis herangezogenen Projekt erheblich, so wirken sich unter Umständen statistische Effekte auf die Aussagekraft der ermittelten Ergebnisse aus. Die Messwerte von im Vergleich sehr kleinen Projekten variieren mitunter im Laufe der Zeit sehr stark, was zu einer unzulässigen Verzerrung der Ergebnisse führt. Aus diesem Grund sollten mit den hier vorgestellten Methoden nur Projekte einer gewissen Mindestgröße, die für jede Messung individuell ermittelt werden muß, auszuwerten. Eine Alternative besteht darin, mehrere kleine Projekte zu einem größeren Gesamtprojekt zusammengefaßt zu betrachten. BugzillaMetrics bietet hierzu die Möglichkeit, verschiedene `groupingParameters` anzugeben, die die betrachteten Projekte auf unterschiedliche Weisen zusammenfassen.

Die Wahl eines der verschiedenen in Kapitel 2.1.3 vorgestellten Auswertungs- und Vergleichsverfahren wirkt sich insbesondere auf das Wesen und die Qualität der ermittelten Ergebnisse aus. Je nach gewünschter Aussage muß eines der vorgestellten Auswertungsverfahren gewählt und überprüft werden, ob das gewählte Auswertungs-

verfahren die zu gewinnenden Aussagen sinnvoll unterstützt. Eine weitere Randbedingung für die Anwendung von Metriken auf Bug-Tracking-Datenbank ist demnach die Existenz eines klaren Ziels, das durch die Auswertung erreicht werden soll. Das reine Messen 'auf Verdacht' in der Hoffnung, ein aussagekräftiges Ergebnis zu erhalten, führt nicht zum Ziel.

Wie zu Beginn dieses Kapitels erläutert, lassen sich mit Hilfe von Auswertungen über Bug-Tracking-Datenbanken nicht alle Aspekte eines Software-Entwicklungsprozesses beurteilen und einstufen. Es ist daher für eine komplette Beurteilung unabdingbar, weitere Informationen zusammenzutragen, die die Beurteilung der übrigen Aspekte eines Entwicklungsprozesses ermöglichen. Denkbar wären hier Metriken auf *Change Management*-Systemen oder auch Gespräche mit bzw. Umfragen unter den Projektbeteiligten, wie sie häufig bereits durchgeführt werden. Auch können solche zusätzlichen Quellen Auskunft über die Aussagekraft der Bug-Metriken liefern; Vergleiche zwischen den im Rahmen eines formalen Auswertungsverfahren gewonnenen Erkenntnissen mit den individuellen Einschätzungen der Projektbeteiligten sollten jedoch grundsätzlich vorsichtig beurteilt werden, da die subjektiven Einschätzungen der Projektmitarbeiter mitunter stark von den objektiv gemessenen Kriterien einer werkzeugunterstützten Auswertung abweichen können.

4.3 Entwicklung von Metriken mit Bugzilla Metrics

Wie bereits in Abschnitt 1.4 angedeutet, wurde das Werkzeug im Hinblick auf die Erstellung benutzerdefinierter Metriken erstellt, die über die Fähigkeiten im Auslieferungszustand von Bugzilla bereits enthaltenen Metrikwerkzeuge hinausgehen. Aus diesem Grund verwendet Bugzilla Metrics eine XML-Beschreibungssprache, mit der unter Zuhilfenahme bestimmter vordefinierter Auswertungsverfahren in gewissem Rahmen eigene Auswertungen erstellbar sind. Bei der Erstellung der Metrikbeschreibungen in XML kann eine webbasierte graphische Benutzeroberfläche zu Hilfe genommen werden, die das Grundgerüst einer neu zu definierenden Metrik automatisch erstellt, bevor im XML-Quelltext entsprechende Anpassungen von Hand vorgenommen werden.

Bugzilla Metrics unterscheidet zwischen vier Basis-Auswertungsverfahren:

`Count Events:` Diese Auswertung zählt in einem frei zu definierenden Zeitraum sogenannte *Events*, zu denen neben Übergängen in verschiedene Zustände das Hinzufügen von Kommentaren oder das Hinzukommen weiterer Bugs zählen. Eine Anwendungsmöglichkeit für dieses Verfahren ist die Ermittlung der Anzahl aller Bugs, die im gesamten Zeitraum ein bestimmtes Filterkriterium, das in BugzillaMetrics angegeben werden kann, um die betrachteten Bugs anhand ihrer Eigenschaften auszuwählen, erfüllten.

`Count Until:` Dieses Auswertungsverfahren entspricht weitgehend dem `Count Events` mit dem Unterschied, daß hier nicht die *Events* im gesamten Auswertungszeitraum erfasst werden, sondern nur bis zum Eintritt eines weiteren, vom Benutzer festzulegenden *Events*. Beispielsweise können mit diesem Verfahren die Anzahl der zu einem Bug hinzugefügten Kommentare gezählt werden, bis der Bug in den Zustand `CLOSED` übergeht.

`Interval Length:` Dieses Verfahren misst die Zeit in Tagen im Lebenszyklus eines Bugs, die zwischen zwei vom Benutzer anzugebenden *Events* vergeht. Mit diesem Auswertungsverfahren ist es zum Beispiel möglich, die Lebensdauer eines Bugs im unbearbeiteten Zustand zu messen.

`Residence Time:` Dieses Verfahren misst den Zeitraum, den ein Bug in einem bestimmten Zustand verbracht hat, bis ein vom Benutzer zu bestimmender *Event* auftritt. Mit diesem Verfahren läßt sich beispielsweise ermitteln, wie lange es nach der Behebung eines Bugs dauert, bis ein Tester die korrekte Behebung des Bugs verifiziert.

Die mit Hilfe dieser vier Auswertungsverfahren gewonnenen Zahlen werden als sogenannten *Case Value Calculators* gespeichert; ein *Case Value Calculator* entspricht somit einer Menge von Variablen, die für jeden ausgewerteten Eintrag den jeweils bei der Durchführung der Auswertung ermittelten Wert beinhaltet. Jede Metrikdefinition kann eine beliebige Anzahl dieser *Case Value Calculator*s besitzen, die unter Anwendung verschiedener Rechenoperationen wie Addition, Subtraktion, Multiplikation und Division miteinander verrechnet werden können. Somit ergibt sich beispielsweise die Möglichkeit, auch prozentuale Werte zu erfassen, indem ein *Case Value Calculator* zur Ermittlung der Gesamtzahl aller Bugs erstellt wird und mit einem weiteren *Case Value Calculator*, der nur die Anzahl einer bestimmten Teilmenge aller Bugs ermittelt verrechnet wird.

Darüber hinaus lassen sich auf die *Case Value Calculator*s statistische Funktionen wie Summe, Maximum, Minimum oder Median anwenden. Mit Hilfe eines sogenannten `Mapping`s ist es außerdem möglich, nicht-numerischen Feldern, die in einem *Case Value Calculator* verarbeitet werden sollen, frei definierbare numerische Werte zuzuweisen. Dies macht es zum Beispiel möglich, die Anzahl offener Fälle gemäß ihrer Priorisierung unterschiedlich zu gewichten.

Die Ausgabe der von einer Metrik ermittelten Werte kann auf verschiedene Weisen erfolgen. Zunächst kann festgelegt werden, zu welchen Zeitpunkten eine Werteausgabe stattfinden soll. Hierbei wird der gesamte Auswertungszeitraum in einzelne Zeitabschnitte unterteilt, die entweder äquidistant (Wochen, Monate, Jahre) oder individuell festgelegt werden können. Zu jedem der definierten Zeitpunkte wird bei der Ausführung der Metrik eine eigene Auswertung erstellt und die ermittelten Werte auszugeben. Hierdurch ist es leicht möglich, den zeitlichen Verlauf einer der Werte einer Metrik in nur einem Durchlauf des Werkzeugs zu ermitteln.

Die Ausgabe der Metrikwerte erfolgt in verschiedenen Formaten. Zunächst ist eine graphische Darstellung der Werte in einem Linien- oder Flächendiagramm möglich, die eine einfache Beurteilung des zeitlichen Verlaufs der Metrikwerte und den direkten, intuitiven Vergleich zwischen verschiedenen ausgewerteten Projekten oder Komponenten ermöglicht. Zur näheren Analyse der ermittelten Werte und der von der Metrik erfassten Bugs steht darüberhinaus ein Export in verschiedene Formate, darunter XML, CSV und Microsoft Excel zur Verfügung. Zur genauen Auswertung und Validierung einer Metrik ohne zusätzliche Hilfsmittel steht außerdem eine tabellarische Ansicht der ermittelten Werte im HTML-Format zur Verfügung wobei jeder ermittelte Wert einen Hyperlink zur Suchfunktion der zugrundeliegenden Bugzilla-Installation besitzt, die einen direkten Zugriff auf diese von der Metrik erfassten Bugs erlaubt. Hierdurch ist es leicht möglich, diese Bugs näher zu analysieren und in Folge dessen Fehler in der Metrikdefinition oder Ursachen für einen schlechten Metrikwert zu ermitteln.

Bei der Entwicklung von Metriken mit BugzillaMetrics ist ein dreistufiges, iteratives Vorgehen angebracht. Zunächst sollte in der Entwurfsphase ermittelt werden, welche Daten erfasst werden sollen, welche Aussagekraft die ermittelten Werte besitzen und ob die gewünschte Auswertung mit BugzillaMetrics realisierbar ist. Im Anschluß daran findet die Implementierung der Metrik in BugzillaMetrics statt. Während der Implementierung ist es möglich, daß festgestellt wird, daß sich eine Auswertung nicht im gewünschten Maße umsetzen läßt, woraufhin die in der Entwurfsphase erarbeitete Spezifikation anzupassen ist. Sobald die Implementierung lauffähig ist sollte die erstellte Metrik ausgeführt werden um mit Hilfe der oben erwähnten tabellarischen Ergebnisse zu überprüfen, ob die entwickelte Metrik tatsächlich das tut, was der Ersteller beabsichtigt oder ob unerwünschte Seiteneffekte auftreten. Dieses dreistufige Vorgehen ist in Abbildung 4.1 dargestellt.

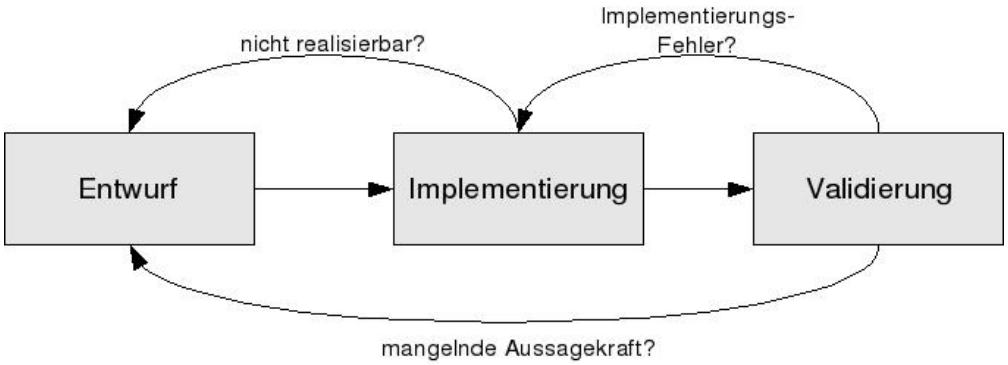

Abbildung 4.1: Dreistufiges Vorgehen bei der Entwicklung von Metriken

4.3.1 Entwurf von Metriken

In der Entwurfsphase jeder Metrik müssen zunächst einige grundlegende Fragen geklärt werden. Idealerweise liegt in dieser Phase bereits eine aus den Prozessvorgaben

entwickelte Spezifikation der Metrik vor, aus der hervorgeht, welche Aussage aus den Metrik-Ergebnissen abgeleitet werden soll. Aus dieser Spezifikation muß nun ermittelt werden, ob die gewünschte Aussage mit Hilfe der in BugzillaMetrics unterstützten Auswertungsverfahren überhaupt gewonnen werden kann. Hierbei spielen sowohl Überlegungen hinsichtlich der Aussagekraft der in Bugzilla erfaßten Daten zu dieser Fragestellung eine Rolle wie auch die technischen Möglichkeiten, die BugzillaMetrics bietet. So ist es beispielsweise nicht sinnvoll, durch Auswertung der Bug-Datenbank eine verlässliche Aussage zur strukturellen Qualität des dem Projekts zugrunde liegenden Quelltexts zu erhalten. Gleichermaßen muß überprüft werden, ob die technischen Möglichkeiten und insbesondere die zur Verfügung stehenden Auswertungsverfahren die gewünschte Aussage überhaupt zulassen. Sollte letzteres nicht der Fall sein, so besteht die Möglichkeit, die Spezifikation dahingehend anzupassen daß eine der ursprünglich gewünschten Aussage sehr ähnliche Aussage getroffen wird, die im Rahmen der technischen Möglichkeiten von BugzillaMetrics ermittelbar ist. Sind diese Fragen geklärt, so muß im nächsten Schritt ermittelt werden, welches der vier zur Verfügung stehenden Auswertungsverfahren tatsächlich zur gewünschten Aussage führt. Hierbei müssen selbstverständlich auch Kombinationen verschiedener Verfahren berücksichtigt werden.

Neben diesen technischen Überlegungen müssen in der Entwurfsphase auch inhaltliche Aspekte berücksichtigt werden. Bei der Anwendung der unterschiedlichen Auswertungsverfahren kann es, je nach gewählter Berechnungsmethode, zu Seiteneffekten kommen, die die Aussagekraft einer Metrik in hohem Maße in Frage stellen.

So ist beispielsweise bei der Anwendung von Auswertungsverfahren mit integrierendem Charakter (d.h. Zählverfahren, bei denen Werte über die Zeit aufsummiert werden) zu beachten, daß es leicht zu Fehlinterpretationen kommen kann, wenn der integrierende Charakter dieser Verfahren nicht in angemessenem Maße berücksichtigt wird. Dadurch, daß bei diesen Auswertungsverfahren auch sämtliche Werte aus der Vergangenheit berücksichtigt werden, wirken sich schlechte Metrikwerte aus der Vergangenheit immer auch auf die Auswertungen zu einem späteren Zeitpunkt aus.

Ähnliche Effekte treten bei Anwendung der verschiedenen Statistik-Funktionen auf einen *Case Value Calculator* auf. So ist es in bestimmten Fällen angebracht, das arithmetische Mittel aller in einer Metrik erfassten Bugs zu ermitteln, in anderen Fällen ist jedoch die Verwendung des Median unabdingbar. Diese Wahl hängt in erster Linie davon ab, ob statistische Ausreißer gezielt erfasst oder unterdrückt werden sollen. Die Anwendung des arithmetischen Mittels auf eine Menge von Bugs, in denen einige Ausreißer enthalten sind, kann zu starken Schwankungen in den ermittelten Metrikwerten zu den verschiedenen Auswertungszeiträumen führen, die einen aussagekräftigen Vergleich unmöglich machen. Umgekehrt kann eine gezielte Ermittlung von Ausreißern mit Hilfe des Medians nicht und durch das arithmetische Mittel nur sehr ungenau stattfinden. Für diesen Zweck definiert BugzillaMetrics die Funktionen `countBelowThreshold` und `countAboveThreshold`, mit denen Ausreißer gezielt unterdrückt oder hervorgehoben werden können.

Ein weiterer in diesem Zusammenhang wichtiger Effekt ist die Tatsache, daß BugzillaMetrics bei der Berechnung der Metriken nach den Verfahren `Count until`, `Interval Length` und `Residence Time` den Metrikwert erst zu dem Zeitpunkt auswertet, zu dem der als 'Abbruchbedingung' definierte *Event* eintritt. Die Ursache dafür, daß ein bestimmter Bug als Ausreißer klassifiziert werden kann, liegt demnach unter Umständen zeitlich lange bevor sich die Wirkung in einer rapiden Verschlechterung des Metrikwerts äußert. Beispielhaft sei hier die Auswertung der durchschnittlichen Bearbeitungszeit aller Bugs genannt. Wird einer der betrachteten Bugs zu einem Zeitpunkt t_1 versehentlich nicht geschlossen, so hat dies zunächst keine Auswirkung auf die Metrikwerte zu einem Zeitpunkt $t_2 > t_1$. Wird nun die Bug-Datenbank zum Zeitpunkt $t_3 > t_2$ zur Verbesserung der Qualität der in der Bug-Datenbank gespeicherten Informationen überprüft und dabei der irrtümlich nicht geschlossene Bug im Nachhinein auf `CLOSED` gesetzt, so würde eine Metrikauswertung des Intervalls $(t_2 \ldots t_4)$ zum Zeitpunkt $t_4 > t_3$ ein deutlich schlechteres Ergebnis anzeigen, obwohl sich die Qualität der in der Bug-Datenbank enthaltenen Informationen gegenüber dem Intervall $(t_1 \ldots t_2)$ tatsächlich verbessert hat.

Weiterhin ist zu beachten, daß sich bei der Auswertung von Metriken ermittelte Absolutwerte (zum Beispiel die Anzahl aller offenen Fälle in einem bestimmten Zeitraum) selten zum Vergleich zwischen unterschiedlichen Projekten oder auch unterschiedlichen Zeiträumen eignen. Durch die in den meisten Fällen vorhandenen Unterschiede in den Projektgrößen, entweder zwischen zwei unterschiedlichen Projekten oder auch zwischen zwei verschiedenen Zeitpunkten im Laufe der Entwicklung des gleichen Projekts, sind Absolutwerte zum Vergleich ungeeignet. Beim Entwurf von Metriken, die Vergleiche implizieren, sollte beim Zählen von Bugs grundsätzlich auf prozentuale oder anteilsmäßige Werte zurückgegriffen werden. So ist beispielsweise eine Auswertung, die die Anzahl aller Bugs mit Priorität `P1` zählt, nicht mit anderen Projekten vergleichbar, hingegen läßt sich eine Aussage zum Verhältnis aller Summe aller Bugs mit Priorität `P1` zur Summe aller in diesem Zeitraum erfassten Bugs ohne weiteres mit anderen Projekten oder dem gleichen Projekt zu einem früheren Zeitraum vergleichen.

Derartige Seiteneffekte lassen sich häufig erst im Validierungsschritt tatsächlich identifizieren, weshalb es unabdingbar ist, beim Auftreten derartiger Seiteneffekte in die Entwurfsphase zurückzukehren und die Spezifikation für die zu entwickelnde Metrik dahingehend anzupassen, daß die erkannten Seiteneffekte nicht mehr auftreten.

4.3.2 Implementierung von Metriken

Im Anschluß an den vorangegangenen Schritt muß die soeben entworfene Metrik in ein für BugzillaMetrics handhabbares Format gebracht werden. Das Werkzeug benötigt für eine Metrik zwei Eingabedateien im XML-Format; die erste Eingabedatei beinhaltet die Spezifikation der Metrik während die zweite die Darstellung der Ausgabe (z.B. die Art des Diagramms und die anzuzeigenden `Case Value Calculators`) steuert.

Zum Zwecke der vereinfachten Implementierung von Metriken stellt BugzillaMetrics eine web-basierte Oberfläche zur Verfügung, die die Erzeugung eines Grundgerüsts für XML-Metrikbeschreibungen, die Bearbeitung des XML-Quelltexts der Metrik und die direkte Auswertung der Metrik unterstützt. Darüberhinaus können einmal entwickelte Metriken zur späteren Wiederverwendung gespeichert werden. Weniger komplexe Metriken können somit direkt ohne manuellen Eingriff durch Zusammenstellung der entworfenen Kriterien über die Web-Oberfläche mit Hilfe des integrierten Code-Generators implementiert werden; für umfangreichere Metriken und Funktionen, die von der Oberfläche nicht unterstützt werden kann zunächst mit dem integrierten Code-Generator das Grundgerüst erzeugt werden (wobei hier auch die Ableitung einer komplexeren Metrik aus verschiedenen, in der Standardinstallation von BugzillaMetrics vordefinierten Basismetriken oder die Erweiterung und Anpassung einer zuvor abgespeicherten Metrik möglich ist) um das generierte Grundgerüst in der weiteren Folge von Hand zu verfeinern und anzupassen. Dies ist insbesondere dann erforderlich, wenn umfangreiche Verrechnungen verschiedener `Case Value Calculators` stattfinden müssen oder der Zugriff auf bestimmte erweiterte Funktionen des Werkzeugs benötigt wird. Hierbei ist jedoch zu beachten, daß sich Metriken, deren Quellcode manuell bearbeitet wurde, im Anschluß einer Veränderung über den Code-Generator entziehen und fortan eine rein manuelle Anpassung des Codes erforderlich ist.

Zum Abschluß der Implementierung einer Metrik muß überprüft werden, daß der erstellte XML-Quelltext der Metrik syntaktisch korrekt ist. Bei der Ausführung einer Metrik antwortet BugzillaMetrics bei syntaktischen Fehlern mit einer Fehlermeldung, die auf das gefundene Problem hinweist.

4.3.3 Validierung von Metriken

Zum Zwecke der Überprüfung ob die implementierte Metrik die geforderte Aussage tatsächlich unterstützt, stellt BugzillaMetrics eine tabellarische Übersicht der ermittelten Werte zusammen mit einem Link auf die Suchfunktion der zugrunde liegenden Bugzilla-Installation bereit. Zunächst sollten die ermittelten Ergebnisse intuitiv und auf Basis vorliegender Erfahrungswerte auf Plausibilität überprüft werden. Hierbei sind insbesondere starke Schwankungen in den Metrik-Werten zu verschiedenen Auswertungszeiträumen sowie Nullwerte von Interesse.

Eine Reihe von Nullwerten kann darauf hindeuten, daß die entwickelte Metrik auf das Projekt (zumindest in diesem Zeitraum) nicht anwendbar ist, da die Voraussetzungen für die Anwendung der Metrik nicht vorliegen. Dies kann zum Beispiel auf die Nichtverwendung einer oder mehrerer der von Bugzilla bereitgestellten Möglichkeiten hindeuten. In einem solchen Fall ist zu überlegen, ob die entwickelte Metrik sinnvoll auf das zu betrachtende Projekt anwendbar ist, insbesondere auch im Hinblick auf die Probleme, die mit Nullwerten beim Vergleich mehrerer Projekte untereinander auftreten können (vgl. hierzu Abschnitt 2.1.4).

Starke Schwankungen und einzelne Ausreißer in den ermittelten Metrikwerten deuten ebenfalls auf eine ungenügende Umsetzung der entwickelten Metrik hin. Eine Möglichkeit zu ermitteln, ob in solchen Fällen tatsächlich ein Fehler im Entwurf der Metrik vorliegt, bietet sich an in dem man die Details der dem Meßwert zugrunde liegenden Bugs betrachtet und nach möglichen Fehlerquellen fahndet. Wie oben erwähnt bietet BugzillaMetrics eine entsprechende Funktion, mit der dies ohne übermäßigen Aufwand möglich ist. Das Ziel bei der Analyse der einzelnen Bugs ist, Merkmale in diesen Bugs zu finden die dazu geführt haben könnten, daß ein Bug irrtümlich von der Metrik erfasst wurde, obwohl er für die Auswertung nicht relevant ist. Außerdem sollte bei der Analyse der Bugs nach Ursachen gefahndet werden die dazu geführt haben, daß einer der weiter oben beschriebenen Seiteneffekte aufgetreten ist. Stellt der Bug beispielsweise einen Ausreißer hinsichtlich der Länge seines Lebenszyklus dar, so deuten lange Intervalle ohne Aktivität mit abschließendem Statusübergang mit hoher Wahrscheinlichkei darauf hin, daß dieser Bug keine tatsächlich lange Bearbeitungszeit hatte sondern daß es lediglich versäumt wurde, diesen Bug nach seiner Behebung sofort zu schließen.

Wurden in diesem Schritt mögliche Probleme und Seiteneffekte identifiziert, so ist zur Entwurfsphase zurückzukehren, um die gefundenen Probleme zu eliminieren. Sollten bei der Betrachtung der Auswertungsergebnisse keine Auffälligkeiten (mehr) erkennbar sein, so muß dennoch überprüft werden, ob die ermittelten Ergebnisse korrekt sind. Da es aufgrund der hohen Anzahl von Bugs in den betrachteten Projekten in den meisten Fällen nicht praktikabel ist, eine vollständige Validierung der entwickelten Metrik durchzuführen indem jeder zugrunde liegende Bug analysiert wird, ist ein stichprobenartiges Vorgehen gerechtfertigt. In diesem Zusammenhang sollte eine ausreichende Zahl zufällig ausgewählter Bugs aus der Menge der der Auswertung zugrunde liegenden Bugs analysiert werden, wobei festzustellen ist, ob die in der Auflistung erscheinenden untersuchten Bugs den Kriterien der entwickelten Metrik entsprechen.

Kapitel 5

Evaluierung des Eclipse-Prozesses

Im folgenden sollen nun die in den vorangegangenen Abschnitten erarbeiteten Prinzipien beispielhaft auf einen realen Prozess aus dem Open Source-Umfeld angewandt werden. Gegenstand der Betrachtungen in diesem Kapitel ist das Open Source-Entwicklungsprojekt *Eclipse*, einer Sammlung von Werkzeugen und Frameworks zur Softwareentwicklung mit einem Schwerpunkt auf `Java` - Entwicklung. Nach eigenen Angaben ist *Eclipse* inzwischen zur größten nicht von *Microsoft* entwickelten Software-Entwicklungsplattform geworden [Cer05].

Das Projekt selbst ist in eine Anzahl von Subprojekten gegliedert, die jeweils separat verwaltet werden. An den einzelnen Subprojekten sind derzeit 563 Entwickler, sogenannte *Committer* beteiligt, die zwischen Januar und Oktober 2008 etwa 34 Millionen neue und geänderte Codezeilen in ungefähr einer Million *Commits* beigetragen haben [Das]. Unter den aktiven Committern stellen Entwickler von `IBM` mit 169 Mitarbeitern die größte Gruppe dar. Die übrigen Entwickler stammen aus verschiedenen Unternehmen, es besteht jedoch auch eine nicht näher organisierte Gruppe von 123 Einzelpersonen, die unabhängig von ihrem Engagement in einer Firma, die die Entwicklung von Eclipse unterstützt, zu Eclipse beitragen.

Die große Anzahl von Subprojekten und Entwicklern macht das Eclipse-Projekt besonders geeignet für eine Analyse mit Hilfe der im Vorangegangenen entwickelten Techniken. Durch die Tatsache, daß es sich bei `Eclipse` nicht um ein Projekt handelt, das von einer geringen Anzahl von Entwicklern aus einem einzelnen Unternehmen vorangetrieben wird, sondern von einer großen Anzahl von Individuen mit sehr unterschiedlichem Hintergrund entwickelt wird, stellt das Projekt besondere Anforderungen an die Qualität des verwendeten Entwicklungsprozesses.

In den folgenden Abschnitten erfolgt nun die Bewertung der Prozessqualität der einzelnen Subprojekte mit Hilfe eines vergleichsbasierten Verfahrens. Hierbei wird in der Reihenfolge vorgegangen, die in den vorangegangenen Abschnitten nahegelegt wurde. Zunächst erfolgen einige Betrachtungen zur Beschreibung des Entwicklungsprozesses im Rahmen des Eclipse-Projekts, im Anschluß daran wird ein Qualitätsmodell entwickelt und die dazugehörigen Metriken beschrieben um im letzten Schritt eine Aus-

wertung der Ergebnisse vorzunehmen. Zuletzt werden die ermittelten Ergebnisse mit anderen Untersuchungen zur Qualität des Eclipse-Prozesses verglichen.

Als Datenbasis für die Auswertung des Prozesses dient die auf *Bugzilla* basierende Bug-Tracking-Datenbank, die sämtliche erfassten Fehler-Reporte und Änderungswünsche seit Oktober 20001 beinhaltet. Betrachtet werden jedoch aus verschiedenen, im folgenden erläuterten Gründen ausschließlich die Daten der Releases der Jahre 2006 (*Callisto*), 2007 (*Europa*) und 2008 (*Ganymede*).

5.1 Der Eclipse-Entwicklungsprozess

Die Beschreibung des für *Eclipse* vorgegebenen Entwicklungsprozesses [EDP08] gliedert sich in zwei Teile. Der erste Teil befaßt sich mit organisatorischen Vorgaben und Empfehlungen, während der zweite Teil den Lebenszyklus eines jeden Eclipse-Projekts festlegt. Dieser zweite Teil ist für die Untersuchungen in dieser Arbeit irrelevant, da hier lediglich beschrieben wird, wie bei der Etablierung eines weiteren Projekts und dessen Integration in das umfassende Eclipse-Projekt vorzugehen ist. Die in der Bug-Datenbank vorhandenen Informationen beziehen sich lediglich auf die fortwährende Entwicklung der bereits bestehenden Projekte, weshalb ausschließlich dieser Teil der Prozessbeschreibung für die folgenden Betrachtungen herangezogen wird (Siehe hierzu auch Abschnitt 1.2).

In der Prozessbeschreibung des Entwicklungsprozesses für Eclipse [EDP08] selbst finden sich verschiedene Vorgaben und Vorschläge zur generellen Vorgehensweise bei der Entwicklung eines Eclipse-Projekts während aus anderen Quellen (insbesondere [GW05]) weitere Informationen zu den Prioritäten und Vorgaben des Prozesses bereitgestellt werden.

Da die Auswertung des Prozesses anhand einer Bug-Tracking-Datenbank stattfindet, übt die korrekte und vorgabenkonforme Verwendung des Bugtracking-Systems einen nicht unerheblichen Einfluß auf die Ergebnisse aus; dies legt nahe, auch das korrekte Vorgehen bei der Erstellung und Bearbeitung von Einträgen in diese Datenbank in die Auswertung einfließen zu lassen. Der Leitfaden *Eclipse Bugzilla Use* [EBU] gibt eine genaue Beschreibung des Arbeitsablaufes bei der Erstellung und Abarbeitung der Einträge, aus der direkt die entsprechenden Qualitätsmerkmale abgeleitet werden können.

5.1.1 Struktur des Eclipse-Projekts

Wie eingangs erwähnt besteht das Eclipse-Projekt aus einer Anzahl von Subprojekten, die im sogenannten *Ecosystem* zusammengefaßt werden. Das *Ecosystem* selbst besitzt drei Gruppen (*Communities*) von Personen, die eine direkte Verbindung zu einzelnen Projekten des *Ecosystems* besitzen:

***Contributors* und *Committers*:** Diese Personengruppe besteht aus den Entwicklern der Projekte. Während *Committers* Zugriff auf die Quellcode-Repositories des Projekts haben, tragen *Contributors* ebenfalls Code und andere Resourcen zum Projekt bei. Der Eclipse-Prozess legt großen Wert darauf, daß diese Gruppe möglichst breit und mit vielen unterschiedlichen Kenntnissen angelegt ist während die Projektleitung dazu angehalten wird, dauerhafte Bemühungen zu unternehmen, neue *Contributors* und *Committers* anzuwerben. Die Vielfältigkeit der Projektbeteiligten wird im sogenannten *Continuation Review* regelmäßig überprüft:

> The purpose of a Continuation Review is to verify that a Proposal or Project continues to be a viable effort and a credit to Eclipse. The Project team will be expected to explain the recent technical progress and to demonstrate sufficient adopter, developer, and user support for the Project. The goal of the Continuation Review is to avoid having inactive projects looking promising but never actually delivering extensible frameworks and exemplary tools to the ecosystem. [EDP08]

Anwender (*Users*): Die Gruppe der Anwender legt die Grundlage für die Entwicklung und Wartung eines Eclipse-Projekts. Der Eclipse-Prozess macht deutlich, daß eine aktive und engagierte Anwendergemeinschaft ein positives Zeichen für den Nutzen und die Notwendigkeit des Projekts darstellt. Durch eine große Anwendergemeinde soll darüberhinaus sicher gestellt werden, daß sich weitere Organisationen, sowohl aus dem kommerziellen wie auch dem Open Source-Umfeld, für das Projekt interessieren und an seiner Entwicklung teilnehmen.

***Adopters*:** Als *Adopter* werden Gruppen und Personen bezeichnet, die basierend auf den innerhalb des Eclipse-Projekts entwickelten Technologien und Werkzeuge eigene Produkte entwickeln, selbst jedoch nicht Mitglied des Eclipse-Projekts sind. Der Eclipse-Prozess fordert, daß der Open Source-Charakter der Entwicklung durch eine angemessene *Adopter*-Gemeinde aufrecht erhalten wird.

Das Fehlen angemessener *Contributor*-, Anwender- oder *Adopter*-Gemeinschaften wertet der Eclipse-Prozess als mangelhafte Offenheit, Transparenz und Attraktivität.

Die Projekte selbst werden von einem Projektverantwortlichen (*Project Leader*) geleitet. Handelt es sich bei einem Projekt um ein sogenanntes *Top Level Project*, so übernimmt ein *Project Management Committee* diese Aufgabe. Die Projektleiter werden, ebenso wie zusätzliche *Committer* von der Entwicklergemeinschaft des Projekts gewählt. Zu den Aufgaben der Projektleitung gehören:

- die Sicherstellung einer effektiven Fortentwicklung des Projekts
- das Ausräumen von Hindernissen, die Beseitigung von Problemen im Projektumfeld sowie die Lösung von Konflikten

- die Einhaltung des Open Source-Charakters der Entwicklung
- die Sicherstellung, daß das Projekt mit den Richtlinien des Eclipse-Prozesses in Einklang steht

5.1.2 Vorgaben an die Subprojekte

Die Beschreibung des Eclipse-Prozesses stellt verschiedene Anforderungen an die im *Ecosystem* zusammengefassten Projekte. Insbesondere werden die folgenden Prinzipien besonders hervorgehoben:

Offenheit: Das Eclipse-Projekt (und all seine Subprojekte) soll niemanden ausschließen; für alle Beteiligten gelten die gleichen Regeln, selbst wenn es sich bei den Projektbeteiligten um Angehörige der direkten Konkurrenz handelt. Darüberhinaus soll niemand, insbesondere potentielle *Contributors* von der Beteiligung am Projekt ausgeschlossen werden.

Transparenz: Alle Diskussionen und Informationen zu einem Projekt wie beispielsweise Informationen zu geplanten Funktionen oder Projektpläne sollen offen gegenüber Dritten sowie öffentlich und leicht zugänglich sein.

Meritokratie: Die Verantwortung, die ein Projektmitglied übernimmt oder übernehmen kann steht im direkten Zusammenhang mit dem Verdienst, den das Mitglied im Zusammenhang mit diesem Projekt erworben hat. Insbesondere werden Projektverantwortliche anhand ihres Verdienstes durch die übrigen Projektmitglieder gewählt.

Das Ziel der Eclipse-Foundation ist es, höchste Qualität erweiterbarer Frameworks, Beispielanwendungen, transparente Prozesse und offene Projekte sicherzustellen. Um diese Ziele zu erreichen soll sichergestellt werden, daß sämtliche Projekte zum Wohl der Mitglieder des *Ecosystems* wie auch dem der Open Source-Gemeinschaft im Umfeld des Projekts geleitet werden. Um dies sicherzustellen sind alle Projekte angehalten, die folgenden Kriterien einzuhalten:

- Alle Projektpläne und die Menge der geplanten Funktionen sollen rechtzeitig, offen und transparent veröffentlicht werden
- Die Qualität der entwickelten Produkte und Frameworks soll hinreichend sein, um die Entwicklung qualitativ hochwertiger (*commercial grade*) Anwendungen auf deren Basis zu ermöglichen
- Zu jedem Projekt sollen ausführliche Beispielanwendungen erstellt werden, die ermöglichen, daß das Projekt einer breiten Anwendergemeinde zugänglich wird

- Jedes Projekt soll am jährlichen Roadmap-Prozess teilnehmen, der höchste Transparenz und gegenseitige Kommunikation mit den übrigen Projekten des *Ecosystems* ermöglicht

Der Eclipse-Prozess betont ebenfalls, daß den Projekten durch die Prozessvorgaben keine unnötigen Einschränkungen auferlegt werden; jedes Projekt soll möglichst viele Freiheiten genießen um die Flexibilität in der Entwicklung sicherzustellen, wobei die vorgegebenen Ziele und Einschränkungen sicherstellen sollen, daß die Zusammenarbeit im globalen Kontext in einem geregelten Umfeld stattfindet:

> [The Eclipse Process] imposes requirements and constraints on the operation of the Projects, and it does so on behalf of the larger Eclipse community. It is an explicit goal of the Development Process to provide as much freedom and autonomy to the Projects as possible while ensuring the collective qualities benefit the entire Eclipse community. [EDP08]

Die Anforderungen an die Subprojekte ergeben sich somit aus den direkten Anforderungen eines jeden Projekts, wobei auch evolutionäre Kriterien und Vorgaben, die sich im Laufe der Zeit aus der aktiven Entwicklung heraus ergeben haben, eine Rolle spielen:

> Part of the strength of [the Eclipse Process] is in what it does not say, and thus opens for community definition through convention, guidelines, and public consultation. A [process] with too much structure becomes too rigid and prevents the kind of innovation and change we desire for Eclipse. In areas where [the Eclipse Process] is vague, we expect the Projects and Members to engage the community-at-large to clarify the current norms and expectations.[EDP08]

Derartige Anforderungen wurden im Jahr 2005 von Gamma und Wiegand [GW05] vorgestellt. Die Abbildung 5.1 visualisiert den Zusammenhang zwischen den verschiedenen Kriterien, die von Eclipse-Projekten eingehalten werden sollen. Im Abschnitt 5.2 wird auf diese Kriterien näher eingegangen.

5.1.3 Releases

Die Veröffentlichung neuer Versionen der Projekte des *Ecosystems* findet in Form von *Milestones* statt, die im Abstand von etwa einem Monat veröffentlicht werden. Hierbei werden die ersten 6 Milestones mit `M1` bis `M6` bezeichnet, während die 5 folgenden Milestones als *Release Candidates* mit `RC1` bis `RC5` betitelt werden. Am Ende des Entwicklungszyklus einer Hauptversion steht dann die *Final*-Version, die idealerweise alle geplanten Änderungen für diesen Release-Zyklus beinhaltet und im Rahmen

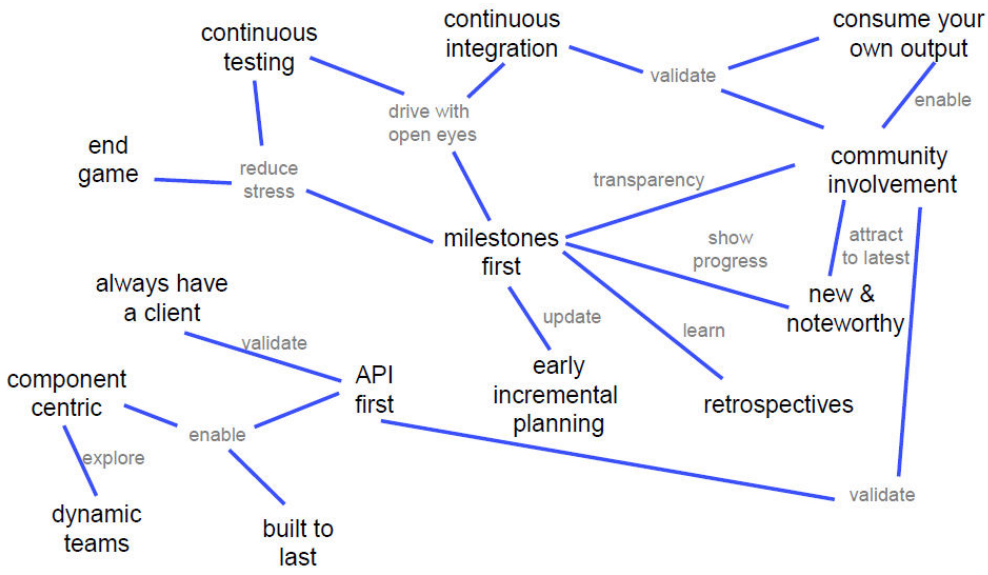

Abbildung 5.1: Zusammenhang zwischen den evolutionären Kriterien des Eclipse-Prozesses (aus [GW05])

der vorangegangenen Milestone-Releases ausführlich durch die *Community* getestet wurde.

Um eine größtmögliche Konsistenz zwischen den wichtigsten Projekten des *Ecosystems* zu erreichen werden die `Final`-Releases der größten Projekte zeitlich abgestimmt und finden innerhalb einer kurzen Zeitperiode regelmäßig jährlich im Juni statt. Diese als *Simultaneous Release* bezeichnete Praxis soll sicherstellen, daß die Integration der verschiedenen Projekte problemlos abläuft und daß *Adopters* bereits im Vorfeld planen können, zu welchem Zeitpunkt welche Versionen bestimmter Subprojekte verfügbar werden [Gan]. Bei diesem *Simultaneous Release* handelt es sich jedoch nicht um eine Zusammenführung der beteiligten Projekte, sondern lediglich um die Festlegung eines gemeinsamen Veröffentlichungszeitpunkts.

Am Release des Jahres 2008 das unter dem Codenamen *Ganymede* bekannt gemacht wurde, waren insgesamt 23 Projekte beteiligt, die in einem vierstufigen Verfahren veröffentlicht wurden. Zunächst wurde am 13. Juni das Kernprojekt `The Eclipse Project`, bestehend aus den Produkten `Platform`, `JDT`, `PDE` und `Equinox` veröffentlicht. Im Verlauf von 5 Tagen wurden im Anschluß die von diesen Grundlagen abhängigen Projekte hinsichtlich ihrer Abhängigkeiten untereinander gegen die bereits fertiggestellten Projekte getestet und ebenfalls veröffentlicht. Am 24. Juni wurden alle in diesem Release enthaltenen Projekte der Öffentlichkeit zugänglich gemacht.

Die Beschreibung des Prozesses für das *Simultaneous Release*, die in jedem Release-Jahr überarbeitet wird, beinhaltet zusätzliche Vorgaben für die beteiligten Projekte; so definiert die Beschreibung des *Simultaneous Release* für das *Ganymede*-Release von

2008 [Gan] unter anderem eine starke Zusammenarbeit zwischen den Projekten die an diesem *Simultaneous Release* teilnehmen.

5.1.4 Einschränkungen bei der Bewertbarkeit des Prozesses

Neben den Einschränkungen, die die ausschließliche Betrachtung einer Bug-Datenbank zur Bewertung und Einstufung eines Entwicklungsprozesses stellt (siehe hierzu Abschnitt 4.2) bestehen durch die Beschreibung des Eclipse-Prozesses selbst einige Einschränkungen hinsichtlich der Bewertbarkeit.

Die Beschreibung des Eclipse-Prozesses betont wie in den vorangegangenen Abschnitten dargestellt die Flexibilität und Eigenverantwortlichkeit eines jeden Teilprojekts sowie den agilen Charakter des Prozesses selbst. Während Metriken mit großer Zuverlässigkeit auf statische Prozesse, d.h. solchen, deren Prozessbeschreibung keiner dauerhaften Änderung über die Zeit unterworfen ist, angewandt werden können, gestaltet sich dies bei agilen Prozesse, insbesondere in Hinblick auf die zeitliche Komponente schwieriger. Ein zu einem früheren Zeitpunkt identifiziertes Qualitätsmerkmal kann zu einem späteren Zeitpunkt bereits obsolet sein; ebenso kann ein Kriterium, das auf ein Projekt anwendbar ist, auf ein anderes Projekt angewandt wesentlich weniger aussagekräftig sein. Aus diesem Grund ist es erforderlich, bei der Identifikation von Qualitätsmerkmalen auf einer allgemeinen Ebene zu operieren und die Details der unterschiedlichen Vorgehensweisen durch eine möglichst allgemein gehaltene Metrik zu übergehen, sofern keine genauen Vorgaben an die Erfordernisse des Prozesses vorhanden sind. Dieser Unterschied wird deutlich wenn man die Beschreibung der Anforderungen an die Planung des Projekts (aus [GW05]) mit den Vorgaben an die Verwendung der Bugzilla-Datenbank [EBU] vergleicht; während bei der Projektplanung lediglich das Ziel (z.B. Einbeziehung der Anwendergemeinde) klar vorgegeben ist während die Methoden zur Erreichung dieser Ziele nicht detailliert beschrieben sind und vom jeweiligen Projekt entsprechend den eigenen Anforderungen interpretiert werden sollen, stellt die Beschreibung zur Verwendung der Bug-Datenbank auch exakte Vorgaben zur Erreichung des jeweiligen Ziels bereit, die jeweils durch eine eigene Metrik überprüft werden können. Während im einen Fall lediglich die Erreichung des Ziels (bzw. daraus abgeleiteter Teilziele) gemessen werden kann, ist im anderen Fall eine genaue Überprüfung der Vorgehensweise durch eine Vielzahl von Metriken möglich. Dies ist bei der Einstufung der Aussagekraft der entsprechenden Metriken zu berücksichtigen.

Neben dieser Einschränkung verzichtet der Eclipse-Prozess ausdrücklich auf die Vorgabe festgelegter Metriken oder Qualitätskriterien:

> We cannot foresee all circumstances and as such should be cautious of being overly prescriptive and/or requiring certain fixed metrics.
>
> The frameworks, tools, projects, processes, community, and even the definition of quality continues to, and will continue to, evolve. Creating rules or processes that force a static snapshot of any of these is detrimental to the health, growth, and ecosystem impact of Eclipse. [EDP08]

Aus diesem Grund kommt ein Vergleich der Projekte an vorgegebenen Schranken zur Bewertung der Qualität nicht in Frage; stattdessen müssen die Projekte untereinander sowie mit sich selbst zu unterschiedlichen Zeitpunkten verglichen werden. Hinsichtlich der Aussagekraft der Auswertung stellt dies jedoch nur eine geringfügige Einschränkung dar, da die gewonnenen Erkenntnisse dennoch zur Verbesserung der Prozesse der Subprojekte herangezogen werden können; ein Projekt, das bei der vergleichenden Auswertung unterdurchschnittlich abschneidet kann seinen Prozess mit dem eines besser bewerteten Projekts vergleichen und andhand der gefundenen Unterschiede Rückschlüsse auf mögliche Prozessverbesserungen ziehen.

5.2 Ein Qualitätsmodell für Eclipse

Zur Entwicklung eines Qualitätsmodells für die Auswertung der Bug-Datenbank des Eclipse-Prozesses stehen insbesondere drei Dokumente zur Verfügung: Die Vorgaben zur Verwendung des Bugzilla-Bugtrackers [EBU], die Beschreibung des Eclipse-Prozesses [EDP08] sowie die Unterlagen zum Vortrag *The Eclipse Way* von Gamma und Wiegand [GW05]. Darüber hinaus lassen sich auch allgemeine, zum verbreiteten Qualitätsverständnis zählende Kriterien anwenden, die in den genannten Dokumenten nicht ausdrücklich erwähnt wurden – hierzu zählt beispielsweise die Entwicklungsgeschwindigkeit oder die Prioritätsverteilung in der Abarbeitung unterschiedlich schwer eingestufter Bugs.

Der betrachtete Entwicklungsprozess läßt sich anhand der Quelle und der Vorgaben und Kriterien, auf die im Folgenden bezug genommen wird in drei erste Qualitätsmerkmale einteilen:

- Die Einhaltung der Vorgaben zur Verwendung von Bugzilla (im weiteren Verlauf als *Bugzilla-Verwendung* bezeichnet) als Zusammenfassung der Frage ‘Wie gut hält das betrachtete Projekt die Vorgaben zur Verwendung von Bugzilla ein?’

- Die Einhaltung von Prozessvorgaben wie sie beispielsweise im Abschnitt 5.1 vorgestellt werden (im weiteren Verlauf als *Prozesseinhaltung* bezeichnet) als Zusammenfassung der Frage ‘Wie gut werden die Prozessvorgaben hinsichtlich der wichtigsten Prozesseigenschaften eingehalten?’

- Die Einhaltung allgemeiner Qualitätskriterien, die im Eclipse-Prozess nicht gesondert erwähnt werden, jedoch den allgemeinen Anforderungen an die Güte eines Entwicklungsprozesses entsprechen (im weiteren Verlauf als *Sonstiges* bezeichnet) als Zusammenfassung der Frage ‘Wie gut hält das betrachtete Projekt allgemeine Anforderungen an die Qualität des Entwicklungsprozesses ein?’

Diese drei Hauptmerkmale können nun mit Hilfe der Kriterien aus den entsprechenden Dokumenten weiter verfeinert werden.

5.2.1 Bugzilla-Verwendung

Die Kriterien zur Verwendung des Werkzeugs *Bugzilla* im Rahmen des Eclipse-Projekts werden durch das Dokument *Eclipse Bugzilla Use* [EBU] festgelegt.

In diesem Dokument findet zunächst eine Beschreibung des Lebenszyklus eines Bugs statt. Im Laufe der Bearbeitung eines Fehlerberichts wechselt der Eintrag je nach durchgeführter Aktion mehrfach seinen Zustand (siehe Abbildung 5.2), was einer Auswertung mit Bugzilla Metrics zugänglich ist. Das daraus abgeleitete Qualitätsmerkmal *Bug-Lebenszyklus*, das der Frage 'Wie gut und vollständig wird der Lebenszyklus eines Eintrags eingehalten?' entspricht beinhaltet insbesondere Messungen zu übersprungenen Zuständen (z.B. bei Überspringen des Zustands `VERIFIED`).

Im Anschluß an die Beschreibung des Bug-Lebenszyklus findet eine Zuordnung bestimmter Felder eines Bug-Eintrags an unterschiedliche Personengruppen statt. Die *Benutzer* (*Users*) der Bug-Datenbank sind für das Einstellen neuer Fehlerberichte oder Erweiterungswünsche zuständig während die *Committer* für die Umsetzung der gewünschten Funktionalität oder die Behebung des gemeldeten Fehlers zuständig sind. Die Benutzer dürfen jedoch keine Einträge in denjenigen Feldern (siehe hierzu die Beschreibung der entsprechenden), die ausschließlich von Committern auszufüllen sind, bei der Erstellung eines Fehlerberichts vornehmen. Diese Vorgabe wird jedoch bereits seitens Bugzilla durch die Eingabemasken für neue Felder überwacht, so daß hier keine Messungen erforderlich sind.

Ein neu eingestellter Fehlerbericht soll baldmöglichst von einem Committer analysiert und der entsprechenden Einstufung zugeführt werden. Dies wird im Dokument [EBU] durch zwei Teilprozesse beschrieben: Zunächst soll der Committer feststellen, ob es sich bei dem eingetragenen Fehlerbericht tatsächlich um einen Fehler handelt und ob alle notwendigen Informationen vorliegen. Dieses Vorgehen wird in [EBU] als *Validierung* (*Validation*) bezeichnet; ein detailliert beschriebener, gültiger und damit qualitativ hochwertiger Fehlerbericht sollte sich ohne längere Diskussion über die Kommentar-Funktion von *Bugzilla* von einem Entwickler validieren lassen. Im Anschluß an diesen Validierungsvorgang soll eine Einstufung des Fehlerberichts anhand seiner Schwere in verschiedene Prioritätsklassen erfolgen; dieser Schritt wird als *Priorisierung* (*Priorization*) bezeichnet. Eine hohe Qualität der Priorisierung zeichnet sich dadurch aus, daß eine enge Korrelation zwischen der subjektiv vom Reporter empfundenen Schwere eines Fehlers und der in *Bugzilla* festgelegten Priorität des Bugs besteht. Aus dieser Vorgehensweise ergeben sich die zwei Qualitätskriterien *Validierung* und *Priorisierung*.

Im nächsten Schritt der Fehlerbehebung wird nun an der Software selbst gearbeitet und der Fehler behoben. Sobald die gewünschte Funktionalität umgesetzt oder der Fehler behoben ist, erfolgen wieder einige Aktionen in Bugzilla hinsichtlich des nun behobenen Eintrags. Hierzu zählt beispielsweise die Überprüfung, ob der Fehler tatsächlich behoben wurde. Die Einhaltung dieses Vorgehens wird im Qualitätskriterium *Bearbeitung* zusammengefaßt.

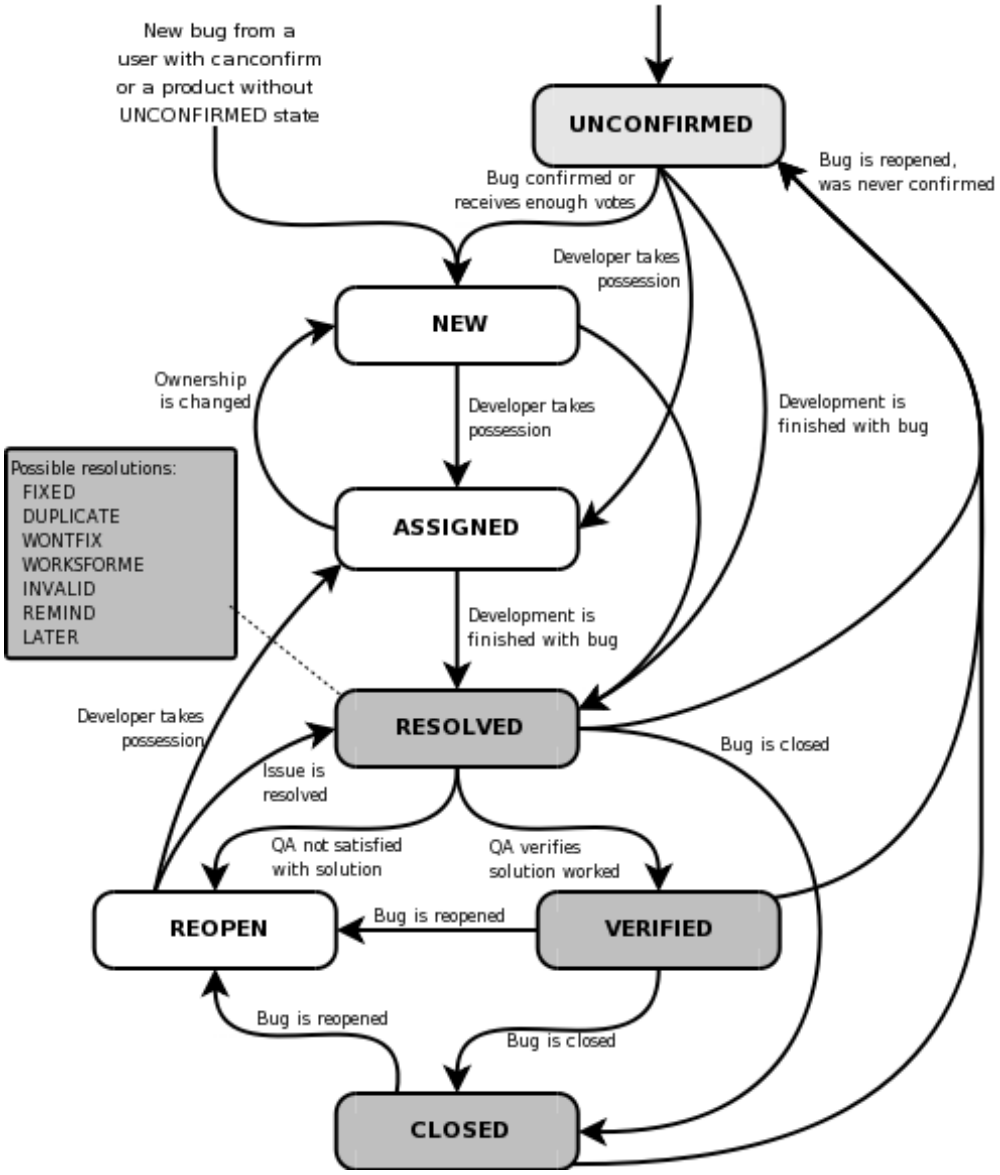

Abbildung 5.2: Lebenszyklus eines Eintrags in der Bugzilla-Datenbank, aus [EBU]

Erreicht ein Projekt nun Release-Reife und möchte eine neue Version veröffentlichen, so sollen wiederum bestimmte Aktionen die Behebung des Fehlers in der veröffentlichten Version verdeutlichen. Die Vorgabe lautet, daß vor einem Release sämtliche in diesem Release enthaltenen Fehlerbehebungen und Erweiterungen auf den Zustand `Closed` gesetzt werden. Diese Forderung ergibt das Qualitätskriterium *Release*.

Der letzte Abschnitt der Richtlinie zur Verwendung von Bugzilla nennt einige Merkmale, die qualitativ hochwertige und somit zügig zu bearbeitende Fehlermeldungen besitzen sollen. Diese Merkmale ergeben das Qualitätskriterium *Qualität der Einträge*.

5.2.2 Prozesseinhaltung

Die Richtlinien des Entwicklungsprozesses selbst sind in [EDP08] und [GW05] aufgeführt. Wie bei den Kriterien zur Verwendung des Bugzilla-Werkzeugs können die aufgeführten Kriterien in einzelne Teilbereiche aufgeteilt werden, die jeweils unterschiedliche Anforderungen stellen.

Der Vortrag 'The Eclipse Way' von Gamma und Wiegand [GW05] nennt als erstes wichtiges Kriterium den Begriff *Milestones First*. Unter diesem Begriff, der stark mit der Prozessvorgabe *Transparenz* in Zusammenhang steht, werden bestimmte Vorgaben getroffen, die insbesondere fordern, daß das Ziel jedes Entwicklungszyklus die Definition und Einhaltung eines *Milestones*, einem definierten Entwicklungsstand zu einem vorgegebenen Datum, ist. Aus der daraus resultierenden Frage 'Wie gut werden die Milestones eingehalten und wie transparent ist der Funktionsumfang eines Milestone' ergibt sich das Qualitätskriterium *Milestones First*.

Das nächste von Gamma und Wiegand postulierte Kriterium lautet *Always Beta*; dieses Kriterium fordert eine kontinuierliche Weiterentwicklung und einen kontinuierlichen Test jedes Entwicklungsabschnittes, auch wenn sich das Software-System im auslieferungsbereiten Zustand befindet. Es sollen also, abgesehen von dem Zeitraum, in dem ein Projekt bereit zum Release gemacht wird (*End Game*), zu jedem Zeitpunkt neue Funktionalitäten in das System aufgenommen werden. Die kontinuierliche Fortentwicklung und Erweiterung des Systems läßt sich anhand der Bug-Datenbank beispielsweise durch das Verhältnis von Fehlerberichten zu Erweiterungswünschen messen. Aus der Frage 'Wird das System kontinuierlich weiterentwickelt?' ergibt sich das Qualitätskriterium *Always Beta*.

Das nächste in [GW05] definierte Kriterium hat die Bezeichnung *Community Involvement* und korreliert stark mit den Forderungen nach *Offenheit* und *Transparenz* aus [EDP08]. Gefordert wird eine ständige Einbeziehung der Entwickler- und Anwendergemeinschaft durch klare Informationen und direkte Beteiligung der Anwendergemeinde an der Fortentwicklung des Systems. Neben einigen nicht mit Hilfe der Bug-Datenbank messbaren Kriterien zur Informationsverteilung sind auch zu diesem Kriterium einige, mit Hilfe von BugzillaMetrics auswertbare Forderungen ableitbar, beispielsweise die Aktualisierung der Fehlerberichte, wenn sich der Bearbeitungszustand des Bugs geändert hat: Sobald die Bearbeitung eines Fehlers begonnen wurde, sollte sich dies im Fehlerbericht durch Setzen des `ASSIGNED`-Status wiederspiegeln; sobald der Fehler abgearbeitet wurde, sollte der Fehlerbericht auf den Status `RESOLVED` gesetzt werden. Aus der Frage 'Wie gut werden Anwender- und Entwicklergemeinde in den Entwicklungsprozess eingebunden?' ergibt sich das Qualitätskriterium *Community Involvement*.

Zum Abschluß eines jeden Release-Zyklus' tritt die Entwicklung in einen kurzen Abschnitt mit der Bezeichnung *End Game* ein. In diesem Zeitraum sollen möglicherweise im Vorfeld liegen gebliebene Fehler endgültig behoben und die im betreffenden Release eingebundenen Funktionalitäten durch Veröffentlichen eines *Release Notes*-Dokuments angekündigt werden; dies kann beispielsweise durch Schließen der zugehörigen

Bug-Einträge erfolgen. In keinem Fall sollen zu diesem Zeitpunkt noch Erweiterungen des Systems vorgenommen werden, die kurz vor dem eigentlichen Release noch zu Schwierigkeiten und neuen Fehlerberichten führen könnten. Das Ziel dieses Entwicklungsabschnittes ist also das Herbeiführen einer ausreichenden Konsistenz zwischen Dokumentationen und dem Softwaresystem. Aus der Frage wie gut dies gelingt ergibt sich das Qualitätskriterium *End Game*.

Das letzte wichtige, anhand von Messungen der Bug-Datenbank zumindest teilweise auswertbare Kriterium aus [GW05] wird als *Built To Last* bezeichnet. Hiermit gemeint ist eine nachhaltige Entwicklung, die das Software-System so stabil, zuverlässig und nachhaltig nutzbar wie möglich macht. Merkmale dieses Kriteriums ist beispielsweise die Zahl der wiedereröffneten Fehler-Einträge, d.h. die Menge der Fehler, die nur unzureichend und nicht nachhaltig gelöst wurden. Aus der Frage 'Wie nachhaltig ist die Entwicklung?' ergibt sich somit das letzte Qualitätskriterium aus dem Bereich *Prozesseinhaltung* mit der Bezeichnung *Built To Last*.

5.2.3 Sonstiges

Neben den in den Prozessdokumentationen veröffentlichten Kriterien sind selbstverständlich auch weitere, allgemein anerkannte Qualitätskriterien auf den Eclipse-Prozess anwendbar. Diese werden in dem separaten übergeordeten Kriterium *Sonstiges* zusammengefaßt.

Zunächst spielt die Auslastung des Entwicklungsteams eine wichtige Rolle. Insbesondere aufgrund der für Open-Source-Prozesse charakteristische heterogene Struktur des Entwicklungsteams ist die Kapazität und damit zusammenhängend die Auslastung des Teams ein wichtiger Anhaltspunkt für die Qualität des Prozesses. Eine Überlastung des Entwicklerteams führt nicht nur zu einer verlangsamten Abarbeitung von Fehlerberichten und Erweiterungswünschen sondern sorgt meist auch dafür, daß die Einhaltung anderer Prozesskriterien zugunsten einer möglichst schnellen Entwicklung in den Hintergrund rückt. Aus der Frage 'besitzt das Entwicklerteam eine ausreichende Kapazität?' ergibt sich somit das Qualitätskriterium *Team-Auslastung*.

Direkt mit der Auslastung des Teams in Zusammenhang steht auch die Entwicklungsgeschwindigkeit. Während ein überlastetes Team auch bei maximaler Ausreizung der erreichbaren Entwicklungsgeschwindigkeit hier wahrscheinlich kein befriedigendes Ergebnis erreichen wird, ist der umgekehrte Fall bei einem nicht ausgelasteten Team nicht zwangsläufig automatisch wahr. Eine mangelhafte Aufgabenverteilung kann beispielsweise auch in einem nicht ausgelasteten Team zu einer Reduktion der Entwicklungsgeschwindigkeit führen. Aus der sich daraus ergebenen Frage 'Wie effektiv nutzt das Entwicklungsteam die zur Verfügung stehende Zeit?' entsteht das Qualitätskriterium *Entwicklungsgeschwindigkeit*.

Ebenfalls mit der Entwicklungsgeschwindigkeit und der Organisation des Teams in Zusammenhang steht die Forderung nach einer optimalen Einteilung der Entwicklungszeit auf unterschiedlich aufwändige Entwicklungsschritte. Als hochprior einge-

stufte Entwicklungsschritte (dies sind insbesondere Bug-Einträge mit hoher Priorität oder großer Schwere) sollten bevorzugt abgearbeitet werden; auch existieren in Bugzilla mehrere Methoden, einen Eintrag als besonders wichtig einzustufen: Einerseits die *Schwere* (*Severity*) des Bugs, andererseits die Prioritätsklasse (*Priority*). Beide sollten korrelieren, damit unterschiedlichen Betrachtungsweisen der Entwickler (ein Entwickler schaut zunächst auf das Feld der *Severity*, ein anderer bevorzugt das Feld *Priority*) Rechnung getragen wird. Während weniger schwere Fälle durchaus eine hohe Priorität besitzen können (beispielsweise weil ein großer Anteil der übrigen Funktionalität von der Behebung des Falls abhängt), sollte ein Fall mit gravierender *Severity* grundsätzlich mit höchster Priorität abgearbeitet werden. Aus der Frage nach einer sinnvollen Aufwandseinhaltung bei der Entwicklung ergibt sich das Qualitätskriterium *Aufwand*.

Aus allen oben aufgeführten Kriterien ergibt sich das Qualitätsmodell für die betrachteten Aspekte in Form eines Baums, der in Abbildung 5.3 dargestellt ist.

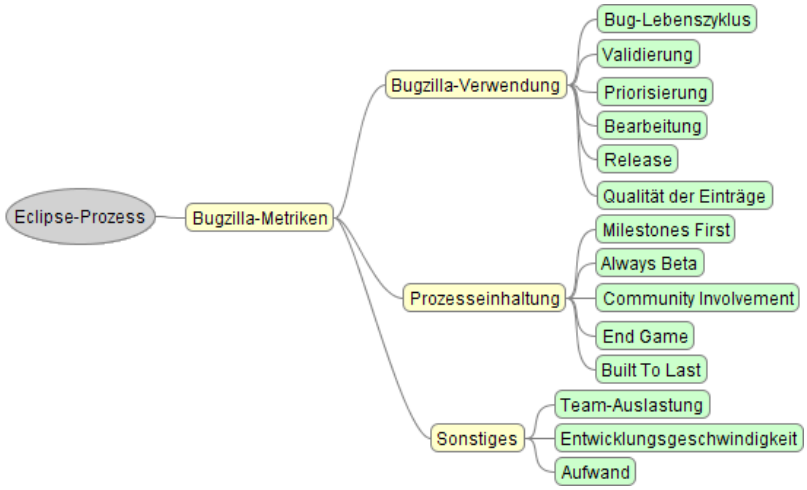

Abbildung 5.3: Qualitätsmodell für den Eclipse-Prozess

5.3 Unterschiedliche Sichtweisen

Betrachtet man das Qualitätsmodell aus Abbildung 5.3, so fällt auf, daß diese Darstellung des Qualitätsmodells ausschließlich diejenige Sicht verkörpert, die aufgrund der Dokumentation entsteht. Die Unterscheidung zwischen den verschiedenen an einem Projekt beteiligten Personengruppen, die für eine Gruppe von Projekten mit derart heterogener Struktur wie die Projekte des *Eclipse-Ecosystems* durchaus sehr relevant ist, fällt hier vollkommen weg. Aus diesem Grund wird das oben angegebene Prozessmodell ausschließlich zur Identifikation und Definition von Qualitätseigenschaften verwandt; die Auswertung selbst erfolgt anhand eines Modells, das die unterschiedlichen Benutzergruppen und deren Aufgaben bei der Entwicklung der Projekte besser berücksichtigt. Dieses Modell entsteht durch eine Transformation des Modells aus Abbildung

5.3, indem mit Hilfe des Top-Down-Ansatzes für diese Sichtweise Qualitätsmerkmale identifiziert werden und in Folge die Qualitätseigenschaften dieses Modells mit den Qualitätsmerkmalen des neu abzuleitenden Qualitätsmodells verbunden werden.

Bei der Identifizierung eines Qualitätsmodells für die Gruppen-bezogene Sicht tritt zunächst die Tatsache in Erscheinung, daß zwei primäre Gruppen an der Entwicklung eines *Eclipse*-Projekts beteiligt sind: die Entwickler (*Committers* bzw. *Developers* und *Contributors*) sowie die Anwender der Projekte (*Users*).

Betrachtet man zunächst die Anwender, so erkennt man, daß die Benutzer sich zunächst durch das Einstellen und Kommentieren von Einträgen an der Pflege der Bugzilla-Datenbank beteiligen. Aus dieser Beobachtung folgt das Qualitätsmerkmal *Bugzilla-Verwendung durch die Anwender*. Über diese direkte Beteiligung hinaus existieren einige Vorgaben an die Transparenz der *Eclipse*-Projekte. Diese Vorgaben regeln die Kommunikation der Entwickler mit den Anwendern, so daß letztere stets über den Fortschritt der Entwicklung sowie neue Funktionen und behobene Fehler im nächsten Release informiert bleiben. Hieraus folgt das Qualitätsmerkmal *Transparenz* als Teil aller die Anwender betreffenden Qualitätsmerkmale.

Ebenso wie die Anwender verändern die Entwickler die Inhalte der Bugzilla-Datenbank. Aus dieser Tatsache folgt das Qualitätsmerkmal *Bugzilla-Verwendung durch die Entwickler* als erstes Teilkriterium der Entwickler-bezogenen Qualitätsmerkmale. Im Gegensatz zu den Anwendern sind die Entwickler direkt an der Abarbeitung der Bug-Reporte beteiligt. Die Qualität dieser Abarbeitung läßt sich durch das Qualitätsmerkmal *Abarbeitung* beschreiben. Ein wichtiges Element hinsichtlich der Entwickler ist auch die sorgfältige Planung der Entwicklung mit Hilfe von Milestones und der Priorisierung bestimmter Bug-Reporte. Aus diesem Element folgt unmittelbar das Qualitätsmerkmal *Planung*.

Neben diesen Anwender- und Entwicklerbezogenen Qualitätsmerkmalen können einige zusätzliche Qualitätsmerkmale identifiziert werden, mit Hilfe derer Aussagen über die Verwendung bestimmter durch Bugzilla bereitgestellten Funktionalitäten getroffen werden können. Hierzu zählt insbesondere die umfassende Verwendung der Zustände `VERIFIED` und `CLOSED` nach der Behebung eines Bugs. Da sich herausgestellt hat, daß sich hieraus keine eindeutigen Qualitätsaussagen ergeben, werden diese Qualitätsmerkmale in einem separaten Qualitätsmerkmal mit der Bezeichnung *Informative Metriken* zusammengefaßt.

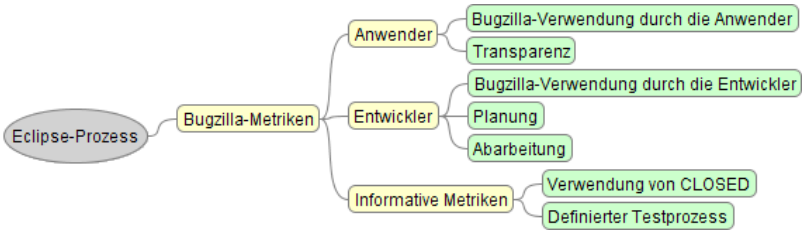

Abbildung 5.4: Qualitätsmodell in Gruppen-Sicht für den Eclipse-Prozess

Als Baum für dieses gruppenbezogene Qualitätsmodell ergibt sich Abbildung 5.4.

5.4 Metriken für das Eclipse-Qualitätsmodell

Im folgenden werden die zum im vorherigen Abschnitt entwickelten Qualitätsmodell gehörigen Metriken beschrieben. Zu jeder Metrik erfolgt jeweils eine **Definition**, die angibt, welche Eigenschaften des betrachteten Projekts durch diese Metrik vermessen werden. Die **Beschreibung** enthält jeweils eine Begründung, weshalb die vorgestellte Metrik relevant und aussagekräftig für die Vermessung der *Eclipse*-Projekte ist und aus welchem Dokument die Metrik abgeleitet wurde. Die anschließende **Klassifizierung** gibt an, aus welchem Qualitätsmerkmal des Baumes der Dokumenten-Sicht die Qualitätseigenschaft abgeleitet wurde und für welches Qualitätsmerkmal des Baumes, der die Gruppen-Sicht reflektiert die Metrik relevant ist. Abschließend erfolgt unter dem Titel **Bedeutung** eine Betrachtung der möglichen Ursachen für ein unbefriedigendes Abschneiden eines Projekts in dieser Metrik.

Die Einteilung der Übersicht erfolgt anhand der drei primären Qualitätsmerkmale des Qualitätenbaumes in Gruppen-Sicht, d.h. *Anwender*, *Entwickler* sowie *Informative Metriken*.

5.4.1 Anwender-bezogene Metriken

Anzahl der gelösten Fälle ohne zugewiesenen `Target Milestone`

Definition: Betrachtet wird der Anteil aller in einem Zeitintervall gelösten Fälle, die zum Zeitpunkt des Statusübergangs nach `RESOLVED` keinen gültigen Eintrag im Feld `Target Milestone` besaßen, sofern das Feld `Resolution` zu diesem Zeitpunkt den Wert `FIXED` besaß. Gültige Einträge sind solche, die einen tatsächlichen Milestone bezeichnen; Einträge wie *Future* gelten nicht als tatsächlicher Milestone. Zu berücksichtigen ist, wie bei allen Metriken die sich mit gelösten Bugs befassen, daß in den Eclipse-Projekten zeitweise die `Resolutions` 'LATER' bzw. 'REMIND' im Zusammenhang mit dem Zustand `RESOLVED` verwendet wurden, um das Aufschieben von Fehlern zu kennzeichnen, ohne daß diese tatsächlich behoben wurden. Zielvorgabe dieser Metrik ist eine Minimierung des Metrikwerts.

Beschreibung: Die Beschreibung des Eclipse-Prozesses [GW05] fordert eine Milestone-ausgerichtete Entwicklung. Darüberhinaus wird eine größtmögliche Transparenz [EDP08] gegenüber den Benutzern und Entwickler-Kollegen gefordert. Das Lösen von Bugs ohne die Information, in welchem Milestone der Fehler behoben ist oder sein wird verletzt beide Vorgaben.

Klassifikation (Dokument-bezogene Sicht): Wie aus der Beschreibung der Metrik hervorgeht, übt diese Metrik Einfluß auf die Qualitätskriterien *Prozesseinhaltung, Milestones First* sowie *Prozesseinhaltung, Communit Involvement* aus.

Klassifikation (Gruppen-bezogene Sicht): Da die Werte dieser Metrik durch die Transparenz gegenüber den Anwendern beeinflußt wird, erfolgt im Gruppen-bezogenen Qualitätsmodell eine Zuweisung an das Qualitätskriterium *Transparenz.*

Bedeutung: Ein ungenügender Metrikwert dieser Metrik kann auf mangelhafte Berücksichtigung der Prozessvorgaben hinsichtlich der Verwaltung von Milestones sowie der Schaffung einer größtmöglichen Transparenz hindeuten.

Anzahl der Kommentare vor Übergang in den Status `ASSIGNED`

Definition: Betrachtet wird die durchschnittliche Anzahl aller Kommentare, die vor dem Übergang der Bugs in den Zustand `ASSIGNED` zu dem jeweiligen Bug erstellt wurden. Zielvorgabe dieser Metrik ist eine Minimierung des Metrikwerts.

Beschreibung: Die Richtlinie zur Verwendung der Bugzilla-Datenbank [EBU] sieht bei einer ungenügenden Beschreibung des Fehlers oder des *Feature Requests*, beispielsweise bei einer unverständlichen Beschreibung oder dem Fehlen von Reproduktionssschritten vor, daß mit Hilfe der Kommentarfunktion weitere Informationen zu diesem Bug eingeholt werden. Bis alle benötigten Informationen vorliegen, soll der Bug im Zustand `NEW` bleiben.

Klassifikation (Dokument-bezogene Sicht): Diese Metrik besitzt lediglich Einfluß auf das Qualitätskriterium *Bugzilla-Verwendung, Validierung*, da es sich um Mängel bei der Erstellung von Bugs handelt, auf die die Projektmitglieder nur geringen Einfluß haben. Aus diesem Grund ist die Aussagekraft dieser Metrik auch zum Qualitätskriterium *Bugzilla-Verwendung, Validierung* als gering einzustufen.

Klassifikation (Gruppen-bezogene Sicht): Da eine hohe Anzahl von Kommentaren zu Bugs in nicht-zugewiesenem Zustand ein Indiz dafür ist, daß Fehlerberichte von Anwendern nicht aussagekräftig formuliert werden, wird diese Metrik im der Gruppen-bezogenen Qualitätsmodell dem Qualitätskriterium *Bugzilla-Verwendung durch die Anwender* zugeordnet.

Bedeutung: Eine hohe durchschnittliche Anzahl von Kommentaren vor der Zuweisung des Bugs an einen *Assignee* deutet auf eine schlechte Qualität der Bug-Beschreibungen hin. Diese kann ihre Ursache in einer nicht ausreichenden Unterweisung der Bug-Reporter haben. Gegenüber den Bug-Reportern müssen die Projektmitglieder eindeutig und klar bekannt machen, welche Informationen sie zur Bearbeitung eines Bugs benötigen.

Doppelt eingestellte Fehlerberichte

Definition: Betrachtet wird der Anteil derjenigen im betrachteten Zeitraum in den Zustand `Resolved` übergegangenen Fehlerberichte, deren Feld `Resolution` den Wert `DUPLICATE` besitzt am Anteil aller im betrachteten Zeitraum auf den Zustand `Resolved` gesetzten Bugs. Zielvorgabe ist eine Minimierung dieses Anteils.

Beschreibung: Findet vor dem Erstellen eines neuen Fehlerberichts durch den Reporter seinerseits keine ausreichende Recherche, ob der Fehler bereits durch andere Personen gemeldet wurde, statt, so entstehen doppelte Fehlereinträge. Das Auffinden und Markieren dieser Duplikate sorgt für zusätzlichen Verwaltungsaufwand seitens der Entwickler; aus diesem Grund sollten doppelte Fehlerberichte vermieden werden, stattdessen sollte der Reporter einen Kommentar zu einem bereits in der Datenbank vorhandenen Eintrag hinzufügen in dem er mitteilt, daß er ebenfalls von diesem Fehler betroffen ist.

Klassifikation (Dokument-bezogene Sicht): Da es sich bei der von dieser Metrik vermessenen Qualitätseigenschaft um eine Eigenschaft aus der Kategorie 'Qualität der Inhalte der Bug-Datenbank' handelt, wird diese Metrik dem Qualitätskriterium *Bugzilla-Verwendung, Qualität der Einträge* zugeordnet.

Klassifikation (Gruppen-bezogene Sicht): Da die Ursache für doppelt eingestellte Fehlerberichte bei den Anwendern zu suchen ist, wird diese Metrik im Gruppenbezogenen Qualitätsmodell dem Qualitätskriterium *Bugzilla-Verwendung durch die Anwender* zugeordnet.

Bedeutung: Ein hoher Anteil an doppelt berichteten Fehlern kann darauf hinweisen, daß die Reporter nicht in ausreichendem Maße dazu angehalten werden, vor dem Einstellen eines Fehlerberichts eine ausführliche Recherche durchzuführen. Werden diese Recherchen durchgeführt während dennoch ein hoher Anteil an Duplikaten vorhanden ist, kann dies aber auch darauf hindeuten, daß die Suchfunktion zur Recherche unzureichend funktional oder bedienbar ist oder daß die Qualität der Einträge dahingehend mangelhaft ist als daß ein Reporter einen bereits vorhandenen Fehlerbericht nicht als Duplikat seiner eigenen Fehlermeldung identifizieren kann.

Als ungültig eingestufte Fehlerberichte

Definition: Betrachtet wird der Anteil derjenigen im betrachteten Zeitraum in den Zustand `Resolved` übergegangenen Fehlerberichte, deren Feld `Resolution` den Wert `INVALID` besitzt am Anteil aller im betrachteten Zeitraum auf den Zustand `Resolved` gesetzten Bugs. Zielvorgabe ist eine Minimierung dieses Anteils.

Beschreibung: Fehlerberichte, die unzureichend oder unklar formuliert sind, sollen gemäß [EBU], sofern nach einem Zeitraum von 7 Tagen keine Klärung herbeigeführt werden konnte, auf den Zustand `Resolved` gesetzt werden, wobei das Feld `Resolution` den Wert `INVALID` erhalten soll. Auf diese Weise wird die Liste der noch zu bearbeitenden Bugs von unklaren, unvollständigen und unsinnigen Fehlerberichten entlastet, was der Übersicht zuträglich ist.

Klassifikation (Dokument-bezogene Sicht): Da es sich bei der von dieser Metrik vermessenen Qualitätseigenschaft um eine Eigenschaft aus der Kategorie 'Qualität der Inhalte der Bug-Datenbank' handelt, wird diese Metrik dem Qualitätskriterium *Bugzilla-Verwendung, Qualität der Einträge* zugeordnet.

Klassifikation (Gruppen-bezogene Sicht): Da die Ursache für unklare und daher als ungültig eingestufte Fehlerberichte bei den Anwendern zu suchen ist, wird diese Metrik im Gruppen-bezogenen Qualitätsmodell dem Qualitätskriterium *Bugzilla-Verwendung durch die Anwender* zugeordnet.

Bedeutung: Ein hoher Anteil an als ungültig eingestuften Fehlerberichten kann auf Probleme mit der Sprachbarriere, die sich durch unterschiedliche Muttersprachen der verschiedenen, weltweit verteilten Beteiligten an einem Eclipse-Projekt ergibt, hindeuten. Ebenso kann aber auch mangelndes technisches Verständnis und die Nichtberücksichtigung der im letzten Abschnitt von [EBU] aufgeführten Kriterien zum Schreiben guter Fehlerberichte seitens der Reporter Ursache für einen hohen Anteil von als ungültig eingestuften Fehlerberichten sein. Bei der stichprobenhaften Betrachtung ungültiger Fehlerberichte hat sich herausgestellt, daß Bedienungsfehler des jeweiligen Produkts durch den Anwender die Hauptursache für als ungültig eingestufte Fehlerberichte sind.

Offene Fälle ohne Version

Definition: Betrachtet wird der Anteil an im betrachteten Zeitraum nicht gelösten Fehlerberichten, bei denen das Feld `Version` nicht gesetzt wurde, am Gesamtvolumen aller im betrachteten Zeitraum nicht gelösten Bugs. Die Zielvorgabe dieser Metrik ist eine Minimierung des Anteils.

Beschreibung: Beim Einstellen eines Fehlerberichts sollen die Reporter angeben, in welcher Produktversion der beschriebene Fehler auftritt, um den Bearbeitern eine einfache Überprüfung und Behebung des Fehlers zu ermöglichen. Das Fehlen dieser Information kann dazu führen, daß der Bearbeiter versucht, den Fehler in einer Produktversion zu reproduzieren, die von diesem Fehler nicht betroffen ist, somit den Fehler mit dem Wert `WORKSFORME` im Feld `Resolution` kennzeichnet und keine Behebung des Fehlers stattfindet.

Klassifikation (Dokument-bezogene Sicht): Da es sich bei der von dieser Metrik vermessenen Qualitätseigenschaft um eine Eigenschaft aus der Kategorie 'Qualität der Inhalte der Bug-Datenbank' handelt, wird diese Metrik dem Qualitätskriterium *Bugzilla-Verwendung, Qualität der Einträge* zugeordnet.

Klassifikation (Gruppen-bezogene Sicht): Die Angabe einer Version, in der ein beschriebener Fehler auftritt, ist Aufgabe der Reporter, die sich hauptsächlich aus den Anwendern eines Produkts zusammensetzen. Aus diesem Grund wird diese Metrik im Gruppen-bezogenen Qualitätsmodell dem Qualitätskriterium *Bugzilla-Verwendung durch die Anwender* zugeordnet.

Bedeutung: Ein hoher Anteil von offenen Fällen, bei denen die Information zur fehlerbehafteten Produktversion fehlt, kann Anzeichen dafür sein, daß die Reporter bei der Erstellung neuer Fehlerberichte nicht mit der erforderlichen Sorgfalt vorgehen; eine weitere Ursache kann aber auch mangelhafte Nachführung der den Reportern zur Auswahl stehenden Produktversionen im Bugtracking-Werkzeug sein.

5.4.2 Entwickler-bezogene Metriken

Nach 7 Tagen unzugewiesene Bugs

Definition: Betrachtet wird der Prozentsatz an im betrachteten Zeitraum neu aufgetretenen Bugs, die am 8. Tag ihrer Erstellung noch keinem Bearbeiter zugewiesen oder auf den Status `RESOLVED` gesetzt wurden. Die Zielvorgabe dieser Metrik ist eine Minimierung des Metrikwerts.

Beschreibung: Das Dokument *Eclipse Bugzilla Use* [EBU] fordert, daß ein neue eingestellter Bug nach spätestens 7 Tagen zugewiesen sein muß, sofern die Angaben des Benutzers, der den Fehlerbericht verfasst hat, ausreichend für eine Behebung des Fehlers sind. Andernfalls muss der Fehlerbericht nach Ablauf des 7. Tages auf den Status `RESOLVED` gesetzt werden.

Klassifikation (Dokument-bezogene Sicht): Diese Metrik übt Einfluß auf das Kriterium *Validierung* des übergeordneten Kriteriums *Bugzilla-Verwendung* aus, da im beschreibenden Dokument explizit gefordert wird, daß jede Fehlermeldung nach 7 Tagen eine entsprechende Statusänderung erfahren muß.

Klassifikation (Gruppen-bezogene Sicht): Hinsichtlich der Gruppen-bezogenen Sicht handelt es sich um eine Qualitätseigenschaft, die die *Bugzilla-Verwendung durch die Entwickler* charakterisiert.

Bedeutung: Ein hoher Anteil an nach 7 Tagen unzugewiesener Fehler kann seine Ursache in einer Überlastung des Projekt-Teams oder einer mangelhaften Beobachtung der neu eingehenden Fehlermeldungen im Bug-Tracking-Werkzeug durch die Projektmitarbeiter haben.

Anteil zugewiesener Fälle ohne *Target Milestone*

Definition: Betrachtet wird in dieser Metrik der Anteil aller Bugs, die im betrachteten Zeitraum einen Übergang in den Zustand `ASSIGNED` erfahren und bei denen das Feld `Target Milestone` des Bug-Eintrags nicht gesetzt ist in Relation zu denjenigen Bugs, die im betrachteten Zeitraum in den Zustand `ASSIGNED` übergehen. Die Zielvorgabe für diese Metrik ist die Minimierung des Metrikwerts.

Beschreibung: Das Dokument [EBU] fordert, daß bei der Zuweisung von Bugs zu einem Bearbeiter (*Assignee*) das Feld `Target Milestone` des Bug-Eintrags auf denjenigen Milestone gesetzt wird, zu dem die vollständige Behebung spätestens geplant ist. Diese Handlung kommt einer gewissen Priorisierung bestimmter Einträge über andere gleich. Darüberhinaus fordert die Eigenschaft *Milestones First* des Eclipse-Prozesses [GW05] eine Transparenz bezüglich des Funktionsumfangs der geplanten Milestones. Diese Transparenz kann durch rechtzeitiges Setzen eines Target Milestone unterstützt werden.

Klassifikation (Dokument-bezogene Sicht): Neben dem Kriterium *Bugzilla-Verwendung, Priorisierung* übt diese Metrik gemäß der Beschreibung auch Einfluß auf das Kriterium *Prozesseinhaltung, Milestones First* aus. Da das Setzen eines Milestones ebenfalls zur allgemeinen Transparenz beiträgt, ist ein Einfluß auf das Kriterium *Prozesseinhaltung, Community Involvement* ebenfalls zu erkennen.

Klassifikation (Gruppen-bezogene Sicht): Da die durch diese Metrik vermessene Qualitätseigenschaft einen erheblichen Einfluß auf die Transparenz der Entwicklung besitzt, wird sie im Gruppen-bezogenen Qualitätsmodell dem Qualitätskriterium *Transparenz* zugewiesen.

Bedeutung: Ein hoher Anteil von zugewiesenen Fehlerberichten ohne gesetzten Target Milestone kann ein Anzeichen für mangelnde Disziplin bei der Verwaltung von Bug-Einträgen sein. Darüberhinaus können ungenügende Werte auch ein Anzeichen für eine nicht zielgerichtete Fortentwicklung des Projekts sein.

Anzahl der Änderungen des `Assignee`

Definition: Betrachtet wird die duchschnittliche Anzahl der Änderungen des Feldes `Assignee` im Gesamtverlauf des Bug-Lebenszyklus (von seiner Erstellung bis zum Übergang in den Zustand `CLOSED`), das den Namen desjenigen Entwicklers beinhaltet, der aktuell mit der Lösung des Bugs betraut ist, gemittelt über die Anzahl aller Bugs. Die Zielvorgabe dieser Metrik ist die Erreichung eines Wertes möglichst nahe bei 1, da jeder Bug im besten Falle lediglich einem `Assignee` zugewiesen weden soll, der den Fehler umgehend behebt.

Beschreibung: Die Zuweisung von Bugs zu einem Bearbeiter sollte möglichst eindeutig sein; jedem Bearbeiter sollte bei der Planung der Implementierung und Wartung ein bestimmter Zuständigkeitsbereich zugewiesen werden. Fällt ein Bug in den Zuständigkeitsbereich eines Bearbeiters, so sollte dieser den Bug nach Möglichkeit vollständig bearbeiten. Sind die Zuständigkeiten nicht eindeutig geklärt oder wird ein Bug, für dessen Behebung der Mitarbeiter nicht zuständig ist, ihm dennoch zugewiesen, so kommt es in der Folge zu 'Weiterreichungen' des Bugs, indem der Bug einem anderen Mitarbeiter zugewiesen wird.

Klassifikation (Dokument-bezogene Sicht): Da die Änderungen des Bearbeiters während der Bearbeitung eines Fehlerberichts auftreten, ist diese Metrik dem Qualitätskriterium *Bugzilla-Verwendung, Bearbeitung* zuzuordnen.

Klassifikation (Gruppen-bezogene Sicht): Da diese Änderungen durch die Entwickler stattfinden, wird diese Metrik im Gruppen-bezogenen Qualitätsmodell dem Qualitätskriterium *Bugzilla-Verwendung durch die Entwickler* zugeordnet.

Bedeutung: Ein unterdurchschnittlicher Metrikwert bei der Auswertung dieser Metrik kann darauf hindeuten, daß die Zuständigkeiten der Mitarbeiter nicht eindeutig geklärt sind oder daß Mitarbeiter in ihrem Zuständigkeitsbereich überlastet sind und aus diesem Grund Tätigkeiten ihres Bereichs an Mitarbeiter anderer Bereiche delegieren müssen. Ebenso ist es denkbar, daß unklare Fehlerberichte zunächst durch einen Mitarbeiter analysiert werden müssen, um den korrekten Bearbeiter zu ermitteln.

Anzahl der Änderungen des `Target Milestone`

Definition: Betrachtet wird die durchschnittliche Anzahl von Änderungen des Feldes `Target Milestone` über diejenigen in der Datenbank enthaltenen Bugs, die sich zum betrachteten Zeitraum in einem 'offenen' Zustand (d.h. nicht `RESOLVED`, `VERIFIED` oder `CLOSED` befanden. Zielvorgabe der Metrik ist eine Minimierung des Metrikwerts.

Klassifikation (Dokument-bezogene Sicht): Das Aufschieben der Bearbeitung von Fehlern, für die bereits eine Zielvorgabe hinsichtlich der Behebung getroffen wurde verletzt in erheblichem Maße die Milestone-orientierte Entwicklung, die in [GW05] gefordert wird. Ein Benutzer, der sich auf die Behebung eines Falls zu einem definierten Zeitpunkt verläßt (und möglicherweise eigene Entwicklungen davon abhängig macht), gerät mitunter in erhebliche Schwierigkeiten, wenn die Behebung des Fehlers nun doch nicht zum festgelegten Zeitpunkt stattfindet.

Klassifikation: Die Ergebnisse dieser Metrik bewerten die Qualität der Milestone-orientierte Entwicklung. Aus diesem Grund ist diese Metrik dem Qualitätskriterium *Prozesseinhaltung, Milestones First* zuzuordnen.

Klassifikation (Gruppen-bezogene Sicht): Diese Metrik vermißt gemeinsam mit der nächsten Metrik (*Anzahl der Fälle mit Änderungen des `Target Milestone`*) ein Qualitätskriterium mit der Bezeichnung *Milestone-Änderungen*. Bei diesem Qualitätsmerkmal handelt es sich um eines, das durch sorgfältige Planung beeinflußt wird. Im Gruppen-bezogenen Qualitätsmodell wird es daher dem Qualitätskriterium *Planung* zugeordnet.

Bedeutung: Ein mangelhaftes Ergebnis in dieser Metrik kann darauf hindeuten, daß die Projektmitarbeiter nicht in der Lage sind, den entstehenden Aufwand bei der Behebung eines Fehlers korrekt einzuschätzen oder die festgelegte Bearbeitungsreihenfolge der unterschiedlichen Bugs korrekt einzuhalten.

Anzahl der Fälle mit Änderungen des `Target Milestone`

Definition: Betrachtet wird der Anteil derjeniger Bugs, die im betracheteten Zeitraum gelöst wurden und bei denen mehr als eine Änderung des Feldes `Target Milestone` stattgefunden hat. Zielvorgabe der Metrik ist eine Minimierung des Metrikwerts.

Beschreibung: Analog der zuvor beschriebenen Metrik *Anzahl der Änderungen des Target Milestone* wird hier nicht die durchschnittliche Anzahl der Milestone-Änderungen erfaßt, sondern die Anzahl der Fälle, bei denen es zu Änderungen des Target Milestone kommt. Zulässig ist hierbei genau eine Änderung (diejenige, die beim erstmaligen Setzen des Milestone auftritt), alle weiteren Änderung dieses Feldes gelten als unerwünschte Verschiebung des geplanten Entwicklungsziels. Im Gegensatz zur Metrik *Anzahl der Änderungen des Target Milestone* wird durch diese Metrik jedoch erfaßt, wie verbreitet Milestone-Änderungen im Projekt sind.

Klassifikation (Dokument-bezogene Sicht): Wie die Metrik *Anzahl der Änderungen des Target Milestone* ist diese Metrik dem Qualitätskriterium *Prozesseinhaltung, Milestones First* zuzuordnen.

Klassifikation (Gruppen-bezogene Sicht): In der Gruppen-bezogenen Sicht wird diese Metrik gemeinsam mit der Metrik *Anzahl der Änderungen des Target Milestone* dem Qualitätskriterium *Milestone-Änderungen* zugeordnet.

Bedeutung: Ein mangelhafter Wert dieser Metrik kann darauf hindeuten, daß die Entwickler des Projekts nicht in ausreichendem Maße Meilenstein-orientiert arbeiten.

Bugs ohne Reaktion innerhalb von 2 Tagen

Definition: Betrachtet wird der Anteil derjeniger Bugs, zu denen innerhalb von 2 Tagen nach Erstellung keine Reaktion (Statusänderung oder Hinzufügen eines Kommentars) stattfand, am Gesamtvolumen aller zu diesem Zeitpunkt in der Datenbank vorhandenen Bugs. Zielvorgabe dieser Metrik ist eine Minimierung des Metrikwerts.

Beschreibung: Die Richtlinien zur Verwendung von Bugzilla [EBU] geben vor, daß innerhalb von 2 Tagen nach dem Einstellen eines Bugs in die Datenbank eine Reaktion seitens der Entwickler erfolgen muß. Dies kann entweder durch eine Statusänderung (Zuweisung zu einem *Assignee*, Setzen des Bugs auf den Zustand `RESOLVED`) oder durch Hinzufügen eines Kommentars (sofern die Beschreibung des Fehlers nicht aufschlußreich ist) erfolgen.

Klassifikation (Dokument-bezogene Sicht): Diese Metrik übt Einfluß auf zwei Qualitätskriterien aus. Zum einen gibt die Richtlinie zur Verwendung von Bugzilla vor, daß auf einen neu eingestellten Fehler innerhalb von 2 Tagen reagiert werden muß (Zuordnung: *Bugzilla-Verwendung, Validierung*), zum anderen dient die Reaktion zur Information der Community-Mitglieder. Somit ist diese Metrik ebenfalls aussagekräftig für das Qualitätskriterium *Prozesseinhaltung, Community Involvement*.

Klassifikation (Gruppen-bezogene Sicht): Hinsichtlich der Gruppen-bezogenen Sicht ist diese Metrik analog zur Metrik *Unbearbeitete Bugs nach 7 Tagen* dem Qualitätskriterium *Bugzilla-Verwendung durch die Entwickler* zuzuordnen.

Bedeutung: Ein hoher Anteil an Bugs, die die Zeitgrenze von 2 Tagen bei der ersten Reaktion übeschreiten kann seine Ursache in einer mangelhaften Beobachtung der Bug-Datenbank durch die Projektmitglieder oder in einer Überlastung der Projektmitglieder haben.

In der Abschlußphase bearbeitete Erweiterungen

Definition: Betrachtet wird der Anteil an Bugs vom Typ `ENHANCEMENT`, die innerhalb eines Monats vor dem Final-Release einer Eclipse-Hauptversion in den Status `RESOLVED` übergehen im Verhältnis zu allen Bugs, die in diesem Zeitraum in den Zustand `RESOLVED` übergehen. Die Zielvorgabe dieser Metrik ist eine Minimierung des ermittelten Werts.

Beschreibung: Gemäß der Beschreibung der zeitlichen Abfolge der Arbeitsschritte bei der Entwicklung einer neuen Eclipse-Hauptversion [GW05] wird zwischen dem letzten `Release-Candidate`-Milestone und der Veröffentlichung der `Final`-Version eine Art Projektabschlußphase (*End Game*) durchgeführt, in der nur noch Fehler einer bestimmten Kategorie und Priorität behoben werden sollen. Keinesfalls sollen in dieser Zeit noch Erweiterungen der Funktionalität implementiert werden, da diese dann nur ungenügend getestet werden und eine Nichteinhaltung des festgelegten Veröffentlichungs-Datums zur Folge haben können.

Klassifikation (Dokument-bezogene Sicht): Diese Metrik ist ausschließlich dem Qualitätskriterium *Prozesseinhaltung, End Game* zuzuordnen.

Klassifikation (Gruppen-bezogene Sicht): Hinsichtlich des Gruppen-bezogenen Qualitätsmodells erfolgt eine Zuordnung zum Qualitätskriterium *Planung*, da eine sorgfältige Planung der Releases und der in ein Release zu integrierenden neuen Funktionen dazu führt, daß kurz vor Abschluß des Release-Zyklus keine neuen Funktionalitäten und Erweiterungen mehr umzusetzen sind.

Bedeutung: Eine hohe Anzahl von Erweiterungen, die in der *End Game*-Entwicklungsphase implementiert werden kann, selbst wenn in dieser Phase keine Fehler mehr zu beheben sind, zu späteren Schwierigkeiten mit der Final-Version führen. Darüber hinaus kann ein schlechtes Abschneiden bei dieser Metrik darauf hindeuten, daß die Entwickler die verschiedenen Projektphasen und ihre Bedeutung nicht verstanden haben oder die Vorgaben zu den Tätigkeiten im Rahmen dieser Projektphasen mißachten.

Rate der wiedereröffneten Fälle

Definition: Betrachtet wird der Anteil an Fällen, deren Felder `RESOLUTION` nicht die Werte `LATER` oder `REMIND` besaßen und die im betrachteten Zeitraum wiedereröffnet wurden im Verhältnis zu allen Bugs, die in diesem Zeitraum geschlossen wurden. Die Zielvorgabe dieser Metrik ist eine Minimierung des Anteils an wiedereröffneten Fällen.

Beschreibung: Ein Fall wird dann wiedereröffnet, wenn ein Benutzer feststellt, daß der beschriebene Fehler nicht, nicht vollständig oder nur unzureichend behoben wurde. Im Idealfall werden sämtliche Fehler bereits beim ersten Behebungsversuch vollständig behoben, so daß eine Wiedereröffnung der Fälle nicht notwendig ist.

Klassifikation (Dokument-bezogene Sicht): Da die Rate der wiedereröffneten Fälle ein Merkmal der Nachhaltigkeit der Entwicklung ist, wird diese Metrik dem Qualitätskriterium *Prozesseinhaltung, Built to Last* zugeordnet.

Klassifikation (Gruppen-bezogene Sicht): Die durch diese Metrik vermessene Qualitätseigenschaft wird durch die korrekte Abarbeitung der Bugs beeinflußt. Aus diesem Grund wird sie im Gruppen-bezogenen Qualitätsmodell der Qualitätseigenschaft *Abarbeitung* zugeordnet.

Bedeutung: Ein hoher Anteil an wiedereröffneten Fällen kann auf eine ungenügende Sorgfalt bei der Behebung von Fehlern hindeuten. Darüber hinaus kann eine hohe Rate an Fällen, die wiedereröffnet werden, auf ein nicht ausreichend sorgfältiges Vorgehen beim Test des Produkts auf behobene Fehler hindeuten.

Dauer bis zur Wiedereröffnung behobener Bugs

Definition: Gemessen wird der Anteil der Bugs im betrachteten Zeitraum, die innerhalb einer Woche nach dem Übergang in den Zustand `RESOLVED` in den Zustand `REOPENED` übergehen an der Anzahl aller Bugs, die innerhalb des letzten Jahres in den Zustand `REOPENED` übergingen. Die Zielvorgabe dieser Metrik ist eine Minimierung des Anteils.

Beschreibung: Ein Fehlerbericht, der innerhalb einer kurzen Zeitspanne wiedereröffnet wird, deutet auf einen mangelhaft behobenen Fehler bzw. einen Fehler, dessen Behebung nicht sorgfältig getestet wurde, hin. Hingegen deutet eine lange Zeitspanne bis zur Wiedereröffnung darauf hin, daß die wichtigsten Fehlerursachen behoben und lediglich einige nicht häufig benötigte Spezialfälle bei der Behebung nicht berücksichtigt wurden.

Klassifikation (Dokument-bezogene Sicht): Die Metrik, die den Anteil derjenigen Bugs mit kurzer Zeitspanne bis zur Wiedereröffnung behobener Fehler vermißt wird von der Nachhaltigkeit der Entwicklung des Produkts beeinflußt, so daß diese Metrik dem Qualitätskriterium *Prozesseinhaltung, Built to Last* zugeordnet wird.

Klassifikation (Gruppen-bezogene Sicht): Da ein hoher Anteil von Bugs, die kurz nach ihrer Behebung wiedereröffnet werden, durch eine sorgfältige Abarbeitung der Fehlerberichte vermieden werden kann, wird diese Metrik im Rahmen der Gruppenbezogenen Sicht dem Qualitätsmerkmal *Abarbeitung* zugewiesen.

Bedeutung: Wie in der Beschreibung der Metrik angedeutet kann ein kurzes mittleres Intervall bis zur Wiedereröffnung darauf hindeuten, daß Fehlerbehebungen nicht sorgfältig durchgeführt und getestet werden.

Mittlere Bearbeitungszeit wiedereröffneter Fälle

Definition: Betrachtet wird die mittlere Bearbeitungszeit wiedereröffneter Fälle, d.h. die Zeitspanne zwischen dem Übergang in den Zustand `REOPENED` in den Zustand `RESOLVED`. Die Zielvorgabe dieser Metrik ist eine Minimierung dieser Zeitspanne.

Beschreibung: Wenn ein Fall wiedereröffnet wird, ist dies üblicherweise ein Anzeichen dafür, daß seit dem ersten Auftreten des Fehlers bereits eine im Verhältnis zu nicht wiedereröffneten Fällen lange Zeit vergangen ist, in der die betroffene Funktion nicht oder nur eingeschränkt zur Verfügung stand, da sich die Zeit zwischen der Behebung und der Wiedereröffnung zur Wartezeit auf die Behebung des Fehlers addiert. Aus diesem Grund ist eine besonders zeitnahe Behebung wiedereröffneter Fehler meist notwendig.

Klassifikation (Dokument-bezogene Sicht): Diese Metrik wird von der Nachhaltigkeit der Entwicklung beeinflußt weshalb sie dem Qualitätskriterium *Prozesseinhaltung, Built to Last* zugeordnet wird.

Klassifikation (Gruppen-bezogene Sicht): Zusammen mit den beiden folgenden Metriken (*Verweildauer offener Fälle* und *Verweildauer von Fällen mit hoher Priorität*) wird diese Metrik zu einem Qualitätsmerkmal *Bearbeitungszeit* zusammengefaßt. Hinsichtlich des Gruppen-bezogenen Qualitätsmodells erfolgt eine Zuordnung dieses Qualitätsmerkmals zum Qualitätsmerkmal *Planung*, da die Bearbeitungszeit durch eine sorgfältige Planung der Entwicklung beeinflußt wird.

Bedeutung: Eine lange mittlere Zeitspanne zur Bearbeitung wiedereröffneter Fälle kann darauf hindeuten, daß das Entwicklungsteam nicht berücksichtigt, daß wiedereröffnete Fälle grundsätzlich eine höhere Priorität besitzen sollten als neu hinzugekommene Fehlerberichte.

Verweildauer offener Fälle

Definition: Diese Metrik vermißt die durchschnittliche Zeitspanne zwischen der Erstellung eines Fehlerberichts und seinem Übergang in einen der Zustände `RESOLVED`, `VERIFIED` oder `CLOSED`. Zielvorgabe der Metrik ist eine Minimierung dieser Bearbeitungs-Zeitspanne.

Beschreibung: Die Zeitspanne zwischen der Erstellung eines Fehlerberichts und der Behebung des Fehlers ist ein grundlegender Anhaltspunkt für die Entwicklungsgeschwindigkeit eines Projekts.

Klassifikation (Dokument-bezogene Sicht): Da diese Metrik die Entwicklungsgeschwindigkeit vermißt, wird sie dem Qualitätskriterium *Sonstiges, Entwicklungsgeschwindigkeit* zugeordnet.

Klassifikation (Gruppen-bezogene Sicht): Gemeinsam mit der vorangegangenen (*Mittlere Bearbeitungszeit wiedereröffneter Fälle*) und der folgenen Metrik (*Verweildauer von Fällen mit hoher Priorität*) wird diese Metrik zu einem Qualitätsmerkmal *Bearbeitungszeit* zusammengefaßt, das seinerseits dem Qualitätsmerkmal *Planung* der Gruppen-bezogenen Sicht zugeordnet ist.

Bedeutung: Eine lange durchschnittliche Zeitspanne zwischen der Erstellung und der Behebung eines Bugs kann auf eine Überlastung des Entwicklungsteams hindeuten. Wenn auf andere Weise festgestellt wird, daß die Entwicklungsgeschwindigkeit deutlich höher ist als von dieser Metrik ermittelt kann dies ein Anzeichen dafür sein, daß behobene Fehler nicht zeitnah in einen Zustand, der die Behebung des Fehlers kennzeichnet, überführt werden.

Verweildauer von Fällen mit hoher Priorität

Definition: Gemessen wird das Verhältnis zwischen den durchschnittlichen Verweildauern offener Fälle (siehe hierzu die Beschreibung der vorangegangen Metrik), die der Prioritätsklasse 1 zugeordnet wurden, zur durchschnittlichen Verweildauer der übrigen Fälle mit unterschiedlichen Prioritätsklassen. Die Zielvorgabe dieser Metrik ist eine Minimierung des Faktors.

Beschreibung: Fälle, die als besonders wichtig eingestuft werden, sollten vordringlich und zügig bearbeitet werden. Das Verhältnis der Verweildauern zwischen solchen und anderen Fällen gibt einen Aufschluß darüber, ob diesem Prinzip Rechnung getragen wird.

Klassifikation (Dokument-bezogene Sicht): Da dieser Wert ebenfalls von der Organisation und Geschwindigkeit der Entwicklung beeinflußt wird, wird diese Metrik zunächst dem Qualitätskriterium *Sonstiges, Entwicklungsgeschwindigkeit* zugeordnet. Zusätzlich ist diese Metrik auch aussagekräftig für die Aufwandsverteilung bei der Entwicklung, weshalb sie zusätzlich dem Qualitätskriterium *Sonstiges, Aufwand* zugeordnet wird.

Klassifikation (Gruppen-bezogene Sicht): Gemeinsam mit den beiden vorangegangenen Metriken (*Verweildauer offener Fälle* und *Verweildauer offener Fälle*) wird diese Metrik zu einem Qualitätsmerkmal *Bearbeitungszeit* zusammengefaßt, das seinerseits dem Qualitätsmerkmal *Planung* der Gruppen-bezogenen Sicht zugeordnet ist.

Bedeutung: Ein Metrikwert größer als 1 bedeutet, daß Fälle mit hoher Priorität sogar einen längeren Bearbeitungszeitraum beanspruchen als solche mit niedriger Priorität. Dieses schlechte Abschneiden eines Projekts kann darauf hindeuten, daß keine vordringliche Bearbeitung besonders wichtiger Fälle im Entwicklungsteam stattfindet. Auch ein Metrikwert zwischen 0,8 und 1 kann nicht als zufriedenstellend angesehen werden, da dies darauf hindeutet, daß Fälle mit hoher Priorität zwar im Durchschnitt schneller bearbeitet werden als Fälle mit niedrigerer Priorität, daß sich die Bearbeitungszeiten zwischen den Bugs mit niedriger und hoher Priorität aber nicht deutlich voneinander unterscheiden, so daß ein solches Ergebnis sehr wahrscheinlich zufällig bedingt ist.

Rate von Fällen mit Priorität 1 oder 2

Definition: Betrachtet wird der Anteil derjenigen Fälle, die eine hohe Prioritätseinstufung von 1 oder 2 besitzen im Verhältnis zu allen im betrachteten Zeitraum vorhandenen Fällen. Die Zielvorgabe für diese Metrik ist eine Minimierung dieses Anteils.

Beschreibung: Bei der ersten Bearbeitung der Fälle soll der Bearbeiter eine Prioritätseinstufung des Falles vornehmen [EBU], um die Bearbeitungsreihenfolge und Dringlichkeit der Fälle anhand der Fehlerbeschreibung und des Feldes `Severity` grob festzulegen. Ein hoher Anteil an Fällen, die als hochprior eingestuft werden deutet darauf hin, daß im Produkt schwerwiegende Probleme auftreten, von denen viele Benutzer betroffen sind.

Klassifikation (Dokument-bezogene Sicht): Da sich diese Metrik ausschließlich mit der Prioritätsverteilung von Fällen befaßt, ist diese Metrik zunächst dem Qualitätskriterium *Bugzilla-Verwendung, Priorisierung* zuzuordnen. Da sich die Prioritäten jedoch auch auf die Aufwandsverteilung auswirken, besitzt diese Metrik ebenfalls Aussagekraft für das Qualitätskriterium *Sonstiges, Aufwand*.

Klassifikation (Gruppen-bezogene Sicht): Diese Metrik wird mit der folgenden (*Verhältnis von Priorität und Schwere*) zu einem Qualitätsmerkmal *Priorisierung* zusammengefaßt. Die Festlegung verschiedener Prioritätsstufen für einzelne Bugs erfolgt durch die Entwickler mit Hilfe des Bugzilla-Werkzeuges. Aus diesem Grund wird dieses Qualitätsmerkmal in der Gruppen-bezogenen Sicht dem übergeordneten Qualitätsmerkmal *Bugzilla-Verwendung durch die Entwickler* zugeordnet.

Bedeutung: Wie in der Beschreibung der Metrik bereits angedeutet kann ein hoher Anteil hochpriorer Fehlerberichte auf die Existenz grober Fehler in wichtigen Modulen des Produkts hinweisen. Darüber hinaus ist ein besonders hoher Anteil als besonders wichtig eingestufter Fälle darauf hindeuten, daß die Wichtigkeit einzelner Fälle generell als zu hoch eingestuft wird, was eine feingranulare Abstufung zwischen den Prioritätsstufen verhindert und somit den Sinn der Vergabe von Prioritäten in Frage stellt.

Verhältnis von Priorität und Schwere

Definition: Betrachtet wird die Zuordnung von Bugs, die vom Reporter in unterschiedliche Schweregrade eingstuft wurden, zu den entsprechenden Prioritätsklassen. Hierbei kommt ein Mapping zum Einsatz, das einem Schweregrad eine Soll-Prioritätsklasse zuordnet. Die Zielvorgabe dieser Metrik ist ein Wert nahe 1.

Beschreibung: Bei der Priorisierung von Bug sollen diese auch anhand ihres Schweregrads in die zur Verfügung stehenden Prioritätsklassen eingeordnet werden. Hierbei gilt daß ein vom Reporter als schwer empfundener Fehler in eine höhere Prioritätsklasse eingestuft wird als ein als weniger gravierend empfundener Fehler. Bugzilla definiert neben fünf Schweregraden ebenfalls fünf Prioritätsklassen, so daß eine Bijektion zwischen Schweregraden und Prioritätsklassen besteht. Das Verhältnis zwischen Schweregrad (1 bis 5) und der korrespondierenden Prioritätsklasse (5 bis 1) wird mit Hilfe eines Mappings vermessen, das bei optimaler Zuordnung der Bugs den Wert 1 annimmt. Werden Fehler in eine für ihre Schwere zu hohe Prioritätsklasse eingestuft ergibt sich ein Wert kleiner als 1, werden Fehler in eine zu geringe Prioritätsklasse eingestuft, ergibt sich ein Wert größer als 1.

Klassifikation (Dokument-bezogene Sicht): Diese Metrik wird von der Einhaltung des Entwicklungsaufwands beeinflußt, weshalb eine Zuordnung zum Qualitätskriterium *Sonstiges, Aufwand* vorgenommen wird.

Klassifikation (Gruppen-bezogene Sicht): Diese Metrik wird hinsichtlich der Gruppen-bezogenen Sicht mit der vorangegangenen (*Rate von Fällen mit Priorität 1 oder 2*) zu einem Qualitätsmerkmal *Priorisierung* zusammengefaßt.

Bedeutung: Ein Wert größer als 1 kann darauf hindeuten, daß dem (subjektiv vom Reporter) empfundenen Schweregrad von den Entwicklern nicht mit einer entsprechend hohen Prioritätseinstufung Rechnung getragen wird. Ein Wert kleiner als 1 ist hingegen in den meisten Fällen unproblematisch, da in diesem Fall Bugs auch bei geringerer Schwere eine angemessene Prioritätseinstufung erhalten.

Verhältnis von Bugs und Enhancements

Definition: Gemessen wird das Verhältnis zwischen der Anzahl derjeniger Bugs, die sich im betrachteten Zeitraum in einem 'offenen' Zustand (d.h. nicht `CLOSED`, `RESOLVED` oder `VERIFIED`) befinden und den Typ `Bug` bestitzen und denjenigen, die sich ebenfalls in einem 'offenen' Zustand befinden und den Typ `Enhancement` besitzen. Zielvorgabe dieser Metrik ist ein möglichst hoher Anteil von `Enhancements`.

Beschreibung: Die Forderung nach einer ständigen Fortentwicklung der Projekte [GW05] ist nur dann erfüllt, wenn regelmäßig neue Funktionen in die bestehenden Produkte integriert werden. Aus diesem Grund ist es notwendig, stets einen hohen Anteil an `Enhancements` zu bearbeiten, um die Weiterentwicklung des Projekts nicht zu behindern.

Klassifikation (Dokument-bezogene Sicht): Da diese Metrik die stetige Weiterentwicklung eines Projekts vermißt, wird sie dem Qualitätsmerkmal *Prozesseinhaltung, Always Beta* zugeordnet.

Klassifikation (Gruppen-bezogene Sicht): Hinsichtlich des Gruppen-bezogene Qualitätsmodell wird die vermessene Qualitätseigenschaft dem Qualitätsmerkmal *Planung* zugeordnet, da die Metrik durch die nachhaltige Planung der Weiterentwicklung des vermessenen Projekts beeinflußt wird.

Bedeutung: Ein geringer Anteil an `Enhancements` kann darauf hindeuten, daß die Weiterentwicklung des Projekts zugunsten von Fehlerbehebungen benachteiligt wird.

5.4.3 Metriken mit informativem Charakter

Anzahl der Übergänge in den Zustand `CLOSED`

Definition: Gemessen wird der Anteil derjeniger Bugs, die im betracheteten Zeitraum in den Zustand `CLOSED` übergehen im Verhältnis zu denjenigen Bugs, die im betracheteten Zeitraum in den Zustand `RESOLVED` übergehen.

Beschreibung: Nach erfolgter Abarbeitung eines Bugs soll idealerweise zunächst die Verifikation mit anschließendem Zustandsübergang in `VERIFIED` erfolgen. Schließlich, spätestens jedoch mit Release der Produktversion, die die Fehlerbehebung enthält, sollen alle in diesem Release enthaltenen Bugs auf den Zustand `CLOSED` gesetzt werden.

Klassifikation (Dokument-bezogene Sicht): Da es sich bei dem von dieser Metrik vermessenen Sachverhalt um die Befolgung des korrekten Lebenszyklus eines Bug-Eintrags handelt, ist diese Metrik dem Qualitätskriterium *Bugzilla-Verwendung, Bug-Lebenszyklus* zuzuordnen.

Klassifikation (Gruppen-bezogene Sicht): Diese Metrik eignet sich dazu, festzustellen, ob der Zustand `CLOSED` von einem Projekt benutzt wird. Es handelt sich hierbei um eine informative Metrik, die in der Gruppen-bezogenen Sicht dem Qualitätskriterium *Verwendung von `CLOSED`* zugeordnet wird.

Bedeutung: Ein geringer Anteil an geschlossenen Bugs kann darauf hindeuten, daß das Projekt den Zustand `CLOSED` nicht konsequent verwendet. Dies kann sich negativ auf Transparenz und Übersichtlichkeit in der Bug-Datenbank auswirken.

Anzahl der geschlossenen Fälle ohne Zustand `VERIFIED`

Definition: Betrachtet wird der Anteil derjenigen Bugs, die sich im Zustand `RESOLVED` befunden haben und in den Zustand `CLOSED` übergehen, ohne vorher im Zustand `VERIFIED` gewesen zu sein im Verhältnis zu sämtlichen im betrachteten Zeitraum geschlossenen Bugs.

Beschreibung: Die Beschreibung des Lebenszyklus eines Fehlerberichts [EBU] fordert, daß ein behobener Fehler, der sich im Zustand `RESOLVED` befindet, von anderen Team-Mitgliedern auf korrekte Behebung überprüft wird. Sobald diese Überprüfung erfolgt ist, setzt das testende Team-Mitglied den Fehler auf den Zustand `VERIFIED`. Erst wenn dies geschehen ist, darf der Eintrag in den Zustand `CLOSED` überführt werden.

Klassifikation (Dokument-bezogene Sicht): Da es sich bei dem von dieser Metrik vermessenen Sachverhalt um die Befolgung des korrekten Lebenszyklus eines Bug-Eintrags handelt, ist diese Metrik dem Qualitätskriterium *Bugzilla-Verwendung, Bug-Lebenszyklus* zuzuordnen.

Klassifikation (Gruppen-bezogene Sicht): Diese Metrik gibt Aufschluß über die Einhaltung eines definierten Test-Prozesses. Ein geringer Anteil an Fällen, die in den Zustand `CLOSED` übergehen ohne vorher im Zustand `VERIFIED` gewesen zu sein deutet darauf hin, daß das Projekt Wert darauf legt, jeden behobenen Bug vor der Veröffentlichung der korrigierten oder erweiterten Programmversion von anderen Team-Mitgliedern auf korrekte Behebung bzw. Umsetzung testen zu lassen. Es handelt sich hierbei um eine informative Metrik, die in der Gruppen-bezogenen Sicht dem Qualitätskriterium *Definierter Testprozess* zugeordnet wird.

Bedeutung: Ein mangelhaftes Ergebnis bei dieser Metrik deutet auf eine unterentwickelte Test-Kultur hin, da die vorgegebenen und notwendigen gegenseitigen Überprüfungen der Arbeit anderer Team-Mitglieder durch Kollegen unterbleibt.

Aus den in den vorangegangenen Abschnitten vorgestellten Metriken, die jeweils bestimmte Qualitätseigenschaften der betrachteten Projekte vermessen, ergibt sich der vollständige Qualitätenbaum gemäß Abbildung 5.5:

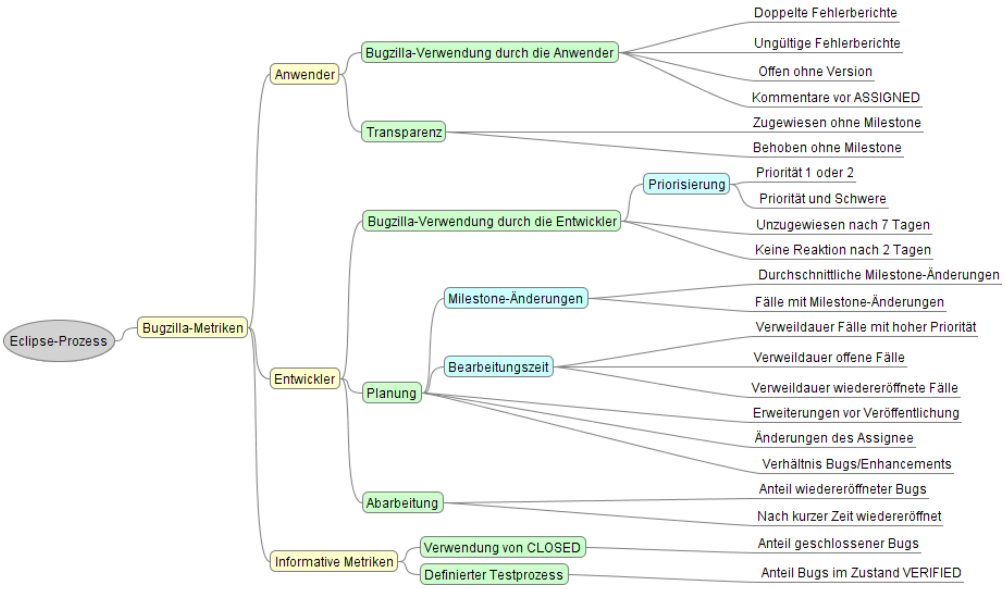

Abbildung 5.5: Vollständiger Qualitätenbaum für die Gruppen-bezogene Sicht

5.5 Vermessung der Eclipse-Projekte

In diesem Abschnitt wird die Auswertung einiger Projekte innerhalb des Eclipse-Ecosystems anhand der in den vorangegangenen Abschnitten vorgestellten Methoden und Metriken vorgenommen und ein Vergleich zwischen den betrachteten Projekten und Zeiträumen gezogen. Darüber hinaus wird in Abschnitt 5.5.9 auf die Besonderheiten einiger Projekte eingegangen, für die ein solcher Vergleich nicht möglich ist.

5.5.1 Ausgewertete Projekte

Wie in Abschnitt 1.5 bereits erwähnt ist das gesamte Eclipse-Projekt in eine Anzahl von Untereinheiten, sogenannte *Classifications* aufgeteilt, die sich wiederum in einzelne Projekte (*Projects*) gliedern. Meist fassen die Classifications thematisch verwandte, aber prinzipiell unabhängige Projekte unter einem Dach zusammen, teilweise entsprechen die Classifications jedoch auch einem Projekt im Sinne des Ecosystems (dies ist z.B. bei der Classification *BIRT* der Fall).

Da eine Auswertung sämtlicher im Eclipse-Projekt zusammengefassten Subprojekte den Rahmen dieser Arbeit sprengen würde und die Vergleichbarkeit von Projekten nur

dann sinnvoll gegeben ist, wenn die Projekte einen nicht allzu großen Größenunterschied aufweisen, werden in dieser Arbeit ausschließlich diejenigen Projekte behandelt, die am sogenannten *Simultaneous Release* (siehe Abschnitte 1.5 und 5.1.3) teilnehmen. Da als Datenbasis für die Auswertung im Rahmen dieser Arbeit lediglich die Bugzilla-Einträge des Zeitraums zwischen Juni 2005 und August 2008 vollständig vorliegen, beschränkt sich die zeitliche Komponente der Auswertung auf die drei Eclipse-Versionen *Callisto* (Juli 2005 bis Juni 2006), *Europa* (Juli 2006 bis Juni 2007) und *Ganymede* (Juli 2007 bis Juni 2008). Als Auswertungszeiträume werden jeweils die vorgenannten Entwicklungszeiträume dieser Versionen herangezogen. Da zum *Europa*-Release eine größere Anzahl kleinerer Projekte erstmals am Simultaneous Release teilgenommen haben, beschränkt sich die Auswertung der Version *Callisto* auf eine geringere Anzahl von Projekten.

Die im Zeitraum der Entwicklung des *Callisto*-Releases betrachteten Projekte [Cal] sind:

BIRT (Business Intelligence and Reporting Tools), CDT (C/C++ IDE), DTP (Data Tools Platform), EMF (Eclipse Modeling Framework), GEF (Graphical Editing Framework), GMF (Graphical Modeling Framework), TPTP (Test and Performance Tools Platform), WTP (Web Tools Platform) und VE (Visual Editor).

Hinzu kommen die Projekte der 'Eclipse'-Classification: Platform (Core Framework), PDE (Plugin Development Environment), JDT (Java Development Tools), Incubator (neue Technologien) sowie Equinox (OSGi community).

Diese Projekte werden im Folgenden abkürzend als *Callisto-Projektgruppe* bezeichnet. Für die Auswertungszeiträume der Releases *Europa* und *Ganymede* kommen folgende Projekte hinzu [Eur]:

Buckminster (Component Assembly Project), DLTK (Dynamic Languages Toolkit), DSDP (Device Software Development Platform) mit den Subprojekten DD (Device Debugging) und TM (Target Management), ECF (Eclipse Communication Framework), M2T (Model-To-Text Transformation Project), MDT (Model Development Tools), Mylyn (Task-focused interface) und STP (SOA Tools Platform).

Analog bezieht sich die Bezeichnung *Europa-Projektgruppe* im weiteren Verlauf auf die zum *Europa*-Release hinzugekommenen Projekte. Auf die Betrachtung der Projekte AJDT (AspectJ Development Tools) sowie Dash (Community Awareness) wurde verzichtet, da diese Projekte ausschließlich am *Europa*-Simultaneous Release teilnahmen. Darüber hinaus wurden die im *Ganymede*-Simultaneous Release hinzugekommenen Projekte EPP (Eclipse Packaging Project), RAP (Rich Ajax Platform) und Subversive (SVN Team Provider) nicht berücksichtigt, da entsprechende Vergleichsdaten fehlen.

Unter den Projekten des *Callisto*-Releases nehmen die Projekte BIRT, DTP und TPTP eine Sonderstellung ein, da sie eine vollständige *Classification* mit weiteren, stark zusammengehörigen Subprojekten umfassen. Während die Auswertung für diese Projekte nach ihrer Classification zusammengefaßt stattfindet (d.h. es wird keine nähere

Aufschlüsselung zu den Unterprojekten vorgenommen), werden die übrigen Projekte einzeln ausgewertet.

Eine weitere Sonderstellung nehmen die Projekte *Visual Editor* (*VE*) und *Incubator* ein, da sie nicht während der gesamten Auswertungsperiode aktiv waren. Diese Projekte finden in der Auswertung der übrigen Projekte keine Berücksichtigung, die sich ergebenden Besonderheiten werden jedoch in Abschnitt 5.5.9 näher erläutert.

5.5.2 Das Auswertungsverfahren

Da sich die Auswertung der Projekte des Eclipse-Ecosystems einer Auswertung mittels *Quality Benchmark Levels* gemäß Abschnitt 2.3 entzieht (siehe hierzu auch Abschnitt 5.5.10), erfolgt die Auswertung anhand eines Punktesystems, wie es auch in anderen Verfahren zur Ermittlung der Prozessqualität (siehe Abschnitt 6.1) zum Einsatz kommt. Als Referenzwerte werden jeweils die Quartile der ermittelten Metrikwerte der im *Callisto*-Release enthaltenen Projekte, zusammengefaßt nach ihren *Classifications* und betrachtet über den Zeitraum von Juli 2006 bis Juni 2007, herangezogen. Durch die Zusammenfassung zu Classifications wird der Einfluß jedes Projekts auf die Ergebnisse anhand der Projektgröße relativiert, so daß größere Projekte ein höheres Gewicht erhalten als kleine oder inaktive Projekte. Dies führt gleichzeitig zu einer Glättung der Werte, so daß die Wahrscheinlichkeit von Ausreißern, die die Referenzwerte verfälschen würden, gering gehalten wird.

Die Wahl der Ergebnisse des *Callisto*-Releases als Ersatz für nicht existierende feste Vorgaben an die Projekte ist dahingehend legitim, als die Forderung, daß sich die Qualität sämtlicher Projekte im Laufe der Zeit kontinuierlich verbessern soll, im Vordergrund steht. Würde man die gemittelten Werte aller betrachteten Releases als Referenzwerte zugrunde legen, so wäre bei der Auswertung ausschließlich eine Abweichung vom Mittelwert zu beobachten; durch die Wahl eines einzelnen Releases sind hingegen weitaus präzisere Angaben zur tatsächlichen Entwicklung der Qualität möglich. Selbstverständlich wäre auch die Wahl der Werte der *Ganymede*- oder *Callisto*-Releases zur Bildung der Referenzwerte möglich, jedoch besitzt das *Callisto*-Release den Vorteil, daß nur eine recht überschaubare Anzahl von Projekten in diesem Release enthalten sind.

Ausgehend von diesen Referenzwerten wird der ermittelte Metrikwert eines jeden Projekts zum entsprechenden Auswertungszeitraum mit den fünf Referenzwerten (Minimum, 1. Quartil q_1, Median q_2, 3. Quartil q_3 und Maximum) verglichen. Handelt es sich um eine Metrik, die eine Minimierung des Metrikwerts f fordert, so dienen die Referenzwerte als obere Schranke, bei Metriken, die eine Maximierung des Metrikwerts fordern, dienen die Referenzwerte als untere Schranke. Gemäß der größten unteren bzw. kleinsten oberen Schranke, die vom betrachteten Metrikwert noch erfüllt wird, wird zur Berechnung der Qualitätseigenschaft eine Punktzahl zwischen 0 und 3 gemäß der folgenden Tabelle 5.1 vergeben:

Maximierung		Minimierung	
Bedingung	Punkte	Bedingung	Punkte
$f \geq q_3$	3	$f > q_3$	0
$f \geq q_2$	2	$f \leq q_3$	1
$f \geq q_1$	1	$f \leq q_2$	2
$f < q_1$	0	$f \leq q_1$	3

Tabelle 5.1: Punktevergabe für Maximierungs- und Minimierungs-Metriken

Nach dieser Tabelle erhält ein Projekt, dessen Metrikwert mit dem Referenzwert zusammenfällt, die jeweils höhere Punktzahl.

Zur Ermittlung der Werte für die Qualitätskriterien dritter Ebene ('Priorisierung', 'Milestone-Änderungen' und 'Bearbeitungszeit'), deren untergeordnete Qualitätseigenschaften sich jeweils mit nur einem Aspekt der Prozessqualität befassen, wird der Durchschnittswert der diesen Qualitätskriterien zugeordneten Qualitätseigenschaften zugeordnet, da es sich hierbei prinzipiell um um ein einzelnes, auf mehrere unterschiedliche Weisen vermessenes Qualitätsmermkal handelt und dieser Wert auf der dritten Ebene mit anderen, ebenfalls in Form von Punktzahlen bewerteten Qualitätsmerkmalen verrechnet werden muß.

Zur Ermittlung der Werte für die Qualitätskriterien der zweiten Ebene wird der Prozentsatz der erreichten Punktzahl in den untergeordneten Qualitätskriterien und -merkmalen in Relation zur maximal erreichbaren Punktzahl berechnet. Die erste Ebene der Qualitätskriterien ('Anwender', 'Entwickler' und 'Informative Metriken') und die Gesamtqualität ergibt sich jeweils als Durchschnittswert der Prozentsätze der untergeordneten Qualitätskriterien zweiter bzw. erster Ebene. Dieses Vorgehen wurde gewählt, um ein intuitiveres Verständnis der durch den Qualitätswert getroffenen Aussage zu wählen; ein Qualitätswert von 71% erschließt sich dem Betrachter leichter als die Aussage 'der Wert des betrachteten Qualitätskriteriums beträgt 2,13 von 3 Punkten'.

5.5.3 Beispiel zum Auswertungsverfahren

Zur Verdeutlichung des Berechnungsverfahrens für Qualitätseigenschaften soll an dieser Stelle exemplarisch die Ermittlung der Werte für die Qualitätseigenschaft 'Unzugewiesen nach 7 Tagen', die dem Qualitätskriterium 'Bugzilla-Verwendung durch die Entwickler' zugeordnet ist, demonstriert werden.

Aus Abschnitt 5.4 geht hervor, daß der Wert für die Qualitätseigenschaft 'Unzugewiesen nach 7 Tagen' möglichst gering sein soll, da neu berichtete Bugs nach spätestens 7 Tagen einem *Assignee* zugewiesen oder als ungültig oder doppelt vorhanden in den Status `RESOLVED` übergehen sollen. Die Ausführung der Metrik mittels *BugzillaMetrics* für einige Projekte ergibt die folgende Wertetabelle 5.2 für die Releases *Callisto*, *Europa* und *Ganymede*:

	BIRT	**DTP**	**Eclipse**	**Modeling**	**TPTP**	**Tools**	**WTP**
Callisto	0,07	0,07	0,07	0,1	0,14	0,03	0,11
Europa	0,02	0,05	0,13	0,13	0,14	0,05	0,12
Ganymede	0,03	0,04	0,07	0,13	0,19	0,02	0,12

	CDT	**EMF**	**GEF**	**GMF**	**VE**	**Equinox**	**Incubator**
Callisto	0,03	0,11	0,03	0,09	0,01	0,21	0
Europa	0,05	0,12	0,01	0,14	0,02	0,07	0
Ganymede	0,02	0,17	0,06	0,07	0,02	0,07	0

	JDT	**PDE**	**Platform**
Callisto	0,07	0,06	0,06
Europa	0,07	0,03	0,17
Ganymede	0,05	0,05	0,08

Tabelle 5.2: Werte der Metrik ‘Unzugewiesen nach 7 Tagen’

Die Referenzwerte für die Ermittlung der Punktzahlen ergeben sich aus den Quartilen der *Callisto*-Werte der *Classifications BIRT*, *DataTools*, *Eclipse*, *Modeling*, *TPTP*, *Tools* und *WebTools* zu den Werten der folgenden Tabelle 5.3:

Maximum	max	0,21
3. Quartil	q_3	0,105
Median	q_2	0,07
1. Quartil	q_1	0,07
Minimum	min	0

Tabelle 5.3: Referenzwerte der Metrik ‘Unzugewiesen nach 7 Tagen’ bezüglich des *Callisto*-Releases

Auffällig ist, daß aufgrund einer Häufung nahe des Wertes 0,07 die Werte für Median und 1. Quartil aufeinander fallen. Gemäß der Definition, daß bei Gleichheit von Metrikwert und Referenzwert die jeweils höhere Punktzahl vergeben wird, kann in der Auswertung dieser Metrik nie die Punktzahl 2 (für das Intervall zwischen q_2 und q_1 vergeben werden.

Graphisch stellen sich die ermittelten Werte wie in Abbildung 5.6 dar, wobei die Intervalle zwischen den Referenzwerten durch umrandete Balken hervorgehoben wurden:

Betrachtet man nun die Werte des *Graphical Editing Frameworks* (*GEF*), so ergibt sich wegen $f_1 = 0,09 < 0,105 = q_3$ und $f_1 = 0,09 > 0,07 = q_2$ eine Punktzahl von 1 für das *Callisto*-Release. Die Verschlechterung des Metrikwerts f_2 zum *Europa*-Release schlägt sich auch in der Punktbewertung nieder: Wegen $f_2 = 0,14 > 0,105 = q_3$ ergibt sich eine Punktzahl von 0 für dieses Release. Im *Ganymede*-Release verbessert sich der zugehörige Metrikwert f_3 wieder und führt wegen $f_3 = 0,07 = q_1$ zu einer Punktzahl von 3 Punkten.

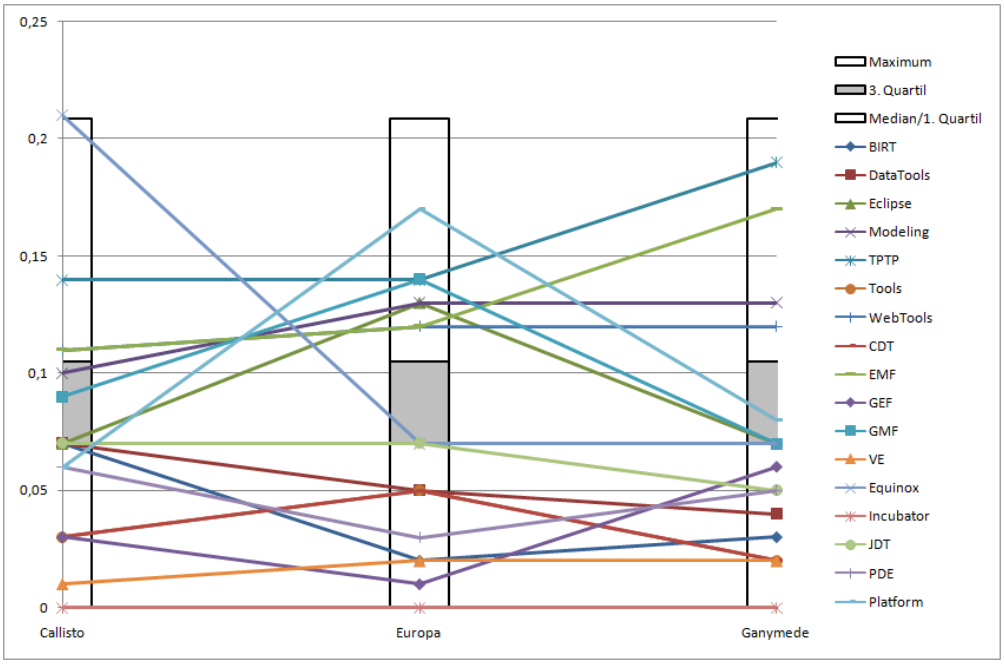

Abbildung 5.6: Graphische Darstellung der Werte der Metrik ‘Unzugewiesen nach 7 Tagen’

Wendet man dieses Verfahren zur Ermittlung der Punktzahl auf alle Projekte an, so ergibt sich eine Punktetabelle gemäß Tabelle 5.4.

	BIRT	**DataTools**	**Eclipse**	**TPTP**	**WebTools**	**CDT**	**EMF**
Callisto	3	3	3	0	0	3	0
Europa	3	3	0	0	0	3	0
Ganymede	3	3	3	0	0	3	0

	GEF	**GMF**	**VE**	**Equinox**	**Incubator**	**JDT**	**PDE**	**Platfm**
Callisto	3	1	3	0	3	3	3	3
Europa	3	0	3	3	3	3	3	0
Ganymede	3	3	3	3	3	3	3	1

Tabelle 5.4: Punkteverteilung zur Metrik ‘Unzugewiesen nach 7 Tagen’

Wendet man dieses Verfahren nun auch auf die übrigen Projekte und die verbleibenden Metriken an, so erhält man die Werte aller Qualitätskriterien des gesamten Qualitätenbaums, auf die sich sämtliche im folgenden getätigten Aussagen beziehen.

5.5.4 Zeitlicher Verlauf der Qualität

Zunächst soll betrachtet werden, wie sich die Gesamtqualität aller Projekte des *Eclipse-Ecosystems* im Laufe der Zeit verändert hat. Hierzu werden Mittelwert sowie Medi-

an, Maximum und Minimum der betrachteten Qualitätskriterien bestimmt. Die Auswertung erfolgt jeweils getrennt für die Gruppen von Projekte, die bereits im *Callisto*-Release enthalten (*Callisto-Projektgruppe*) bzw. erst zum *Europa*-Release hinzugekommen sind (*Europa-Projektgruppe*), da deutliche Unterschiede zwischen beiden Projektgruppen bestehen.

Abbildung 5.7 zeigt den zeitlichen Verlauf der Gesamtqualität der *Callisto*-Projektgruppe, Abbildung 5.8 den Verlauf der Gesamtqualität der *Europa*-Projektgruppe.

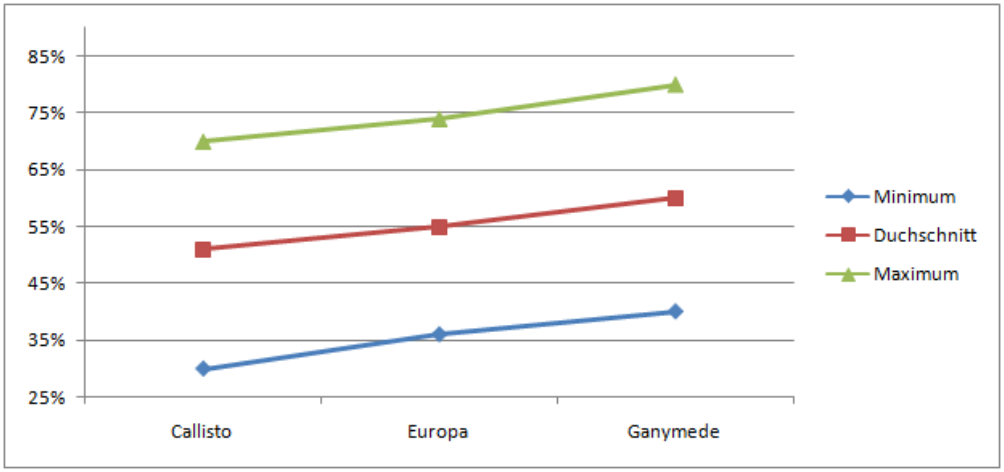

Abbildung 5.7: Zeitlicher Verlauf der Qualität der *Callisto*-Projektgruppe, gemittelt über alle Projekte

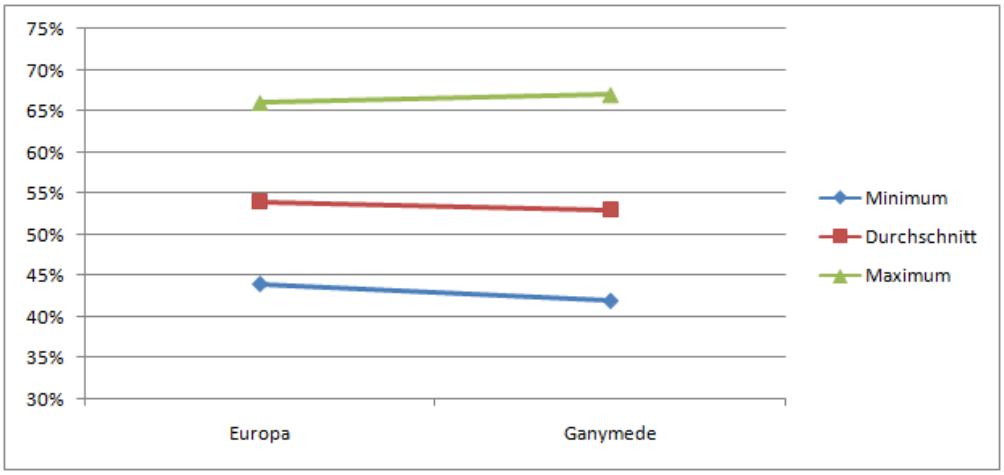

Abbildung 5.8: Zeitlicher Verlauf der Qualität der *Europa*-Projektgruppe, gemittelt über alle Projekte

Das Schaubild in Abbildung 5.7 zeigt einen klar erkennbaren Trend: Im Fortlauf der Entwicklung der *Callisto*-Projektgruppe steigt die gemittelte Gesamtqualität im Laufe der Zeit kontinuierlich an. Auch die Streuung der unterschiedlichen Qualitäten verändert sich nur geringfügig, was an einer ebenfalls positiven Entwicklung der Minima und Maxima erkennbar ist. Betrug die Differenz zwischen Minimum und Maximum

der *Callisto*-Projektgruppe zum *Callisto*-Release 40%, so beträgt sie zum *Ganymede*-Release ebenfalls 40%, nach einer vorübergehenden Reduktion auf 38% im *Europa*-Release an.

Die Überprüfung der statistischen Signifikanz dieser Entwicklung mit Hilfe eines *t-Tests für gepaarte Stichproben* [BT94], wobei für die Stichprobenpaare die Werte des *Callisto*- und des *Ganymede*-Releases herangezogen wurden, ergibt eine Wahrscheinlichkeit $P(T \leq t) = 5,8\%$ für eine rein zufällig positive Entwicklung der Qualität der Projektgruppe. Bei der Wahl des meist angewandten Signifikanzniveaus von $\alpha = 5\%$ müßte die Signifikanz der positiven Gesamtentwicklung knapp abgelehnt werden. Jedoch ist zu beachten, daß derartige statistische Tests erst bei einer hinreichend großen Stichprobenanzahl aussagekräftig werden – über die Signifikanz der Entwicklung kann daher an dieser Stelle keine klare Aussage getroffen werden.

Die Projekte der *Europa*-Projektgruppe zeigen dagegen keinen klar erkennbaren Trend: neben einer fast konstanten Durchschnittsqualität (54% bei *Europa*, 53% bei *Ganymede*) steigt die Streuung der einzelnen Projekte zwischen Minimum und Maximum von zunächst 22% im *Europa*-Release leicht auf 25% an. Die Anwendung des *t-Tests* auf diese Stichprobenmenge ergibt dementsprechend einen Wahrscheinlichkeitswert $P(T \leq t) = 36,2\%$ für die Annahme, daß der zu erkennende leicht abfallende Qualitätswert zufälliger Natur ist. Hier ist also eindeutig keine signifikante Entwicklung zu beobachten.

Der Vergleich zwischen den Durchschnittsqualitäten der *Callisto*-Projektgruppe mit denen der *Europa*-Projektgruppe zeigt, daß hier keine gravierenden Unterschiede in der Qualität festzustellen sind. Der Unterschied im *Europa*-Release beträgt lediglich 1% während er im *Ganymede*-Release auf 7% anwächst, was hauptsächlich auf die positive Entwicklung der *Callisto*-Projektgruppe zurückzuführen ist. Bedenkt man, daß in der *Callisto*-Projektgruppe hauptsächlich große Projekte, die eine vollständige Classification umfassen (z.B. *BIRT*, *DSDP* und *TPTP*) vorhanden sind, die von intuitiven Erwartungen her eine deutlich bessere Qualität entwickelt haben sollten als die teilweise recht kleinen Projekte der *Europa*-Projektgruppe, so überrascht diese Beobachtung.

Festzustellen ist nun, welche Gruppe von Projektbeteiligten (Anwender oder Entwickler) von der Verbesserung der Gesamtqualität besonders betroffen war; Die Abbildungen 5.9 und 5.10 zeigen den zeitlichen Verlauf der Qualitätskriterien der ersten Ebene, die eine Unterscheidung zwischen Anwender- und Entwicklerbezogenen Qualitätskriterien vornehmen.

Auffällig ist, daß sich die anwenderbezogenen Qualitätskriterien bei allen Projekten stetig verbessert haben. Hingegen erfolgte bei den Projekten der *Callisto*-Projektgruppe im Zeitraum zwischen *Callisto*- und *Europa*-Release eine klare Verschlechterung der Entwickler-bezogenen Qualitätskriterien; auch bei der *Europa*-Projektgruppe läßt sich dieser Trend zwischen *Europa*- und *Ganymede*-Release feststellen, wenn auch der Abstand zwischen Anwender- und Entwickler-bezogenen Qualitätskriterien hier deutlich größer ist. Über den gesamten Zeitraum betrachtet läßt sich jedoch erkennen, daß insbesondere die anwenderbezogenen Qualitätskriterien von der Verbesserung der Ge-

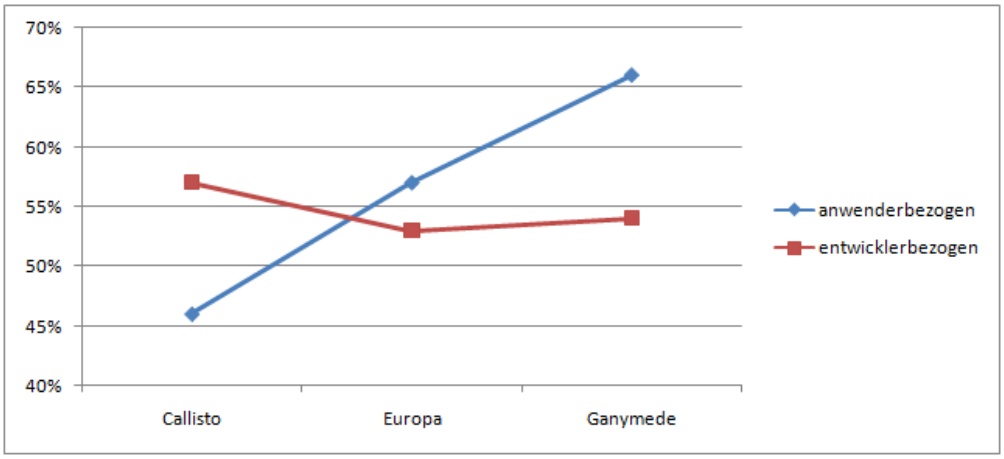

Abbildung 5.9: Zeitlicher Verlauf der ersten Ebene der *Callisto*-Projektgruppe, gemittelt über alle Projekte

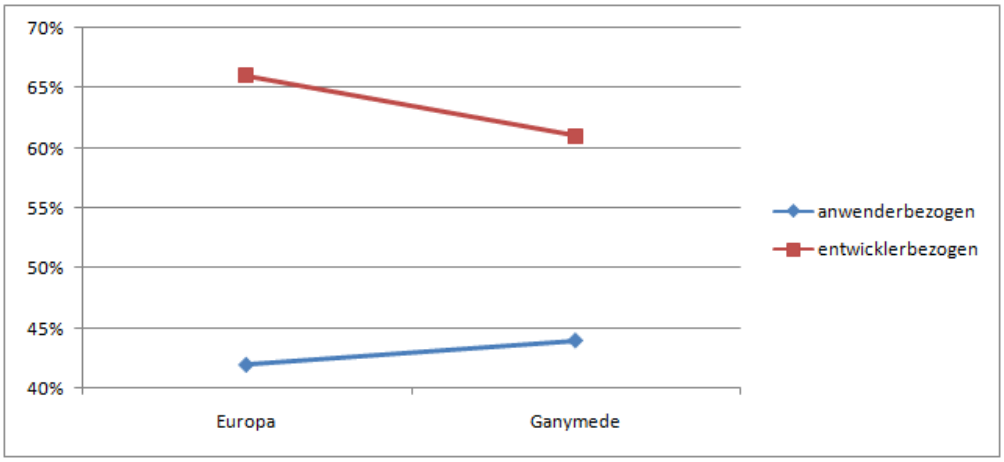

Abbildung 5.10: Zeitlicher Verlauf der ersten Ebene der *Europa*-Projektgruppe, gemittelt über alle Projekte

samtqualität profitieren konnten. Dies wird in Abschnitt 5.5.7 noch näher untersucht. Die Verschlechterung der Qualität der entwicklerbezogenen Qualitätskriterien bei den Projekten der *Callisto*-Projektgruppe ist auf eine Verschlechterung der Qualitätskriterien ‘Abarbeitung’ (55 Prozentpunkte im *Callisto*-Release zu 47 Prozentpunkten im *Europa*-Release) und ‘Planung’ (51 zu 48 Prozentpunkten) zurückzuführen; Bei den Projekten aus der *Europa*-Projektgruppe findet diese Verschlechterung ausschließlich hinsichtlich des Qualitätskriteriums ‘Abarbeitung’ statt.

Eine kontinuierlich positive Entwicklung der Durchschnittsqualität läßt jedoch keineswegs auf eine konstant positive Entwicklung der einzelnen, in den jeweiligen Projektgruppen zusammengefaßten Projekte schließen. Daher soll im Folgenden die Entwicklung der Einzelprojekte näher betrachtet werden. Die Abbildungen 5.11 und 5.12

zeigen die qualitative Entwicklung der Einzelprojekte, getrennt nach *Callisto*- und *Europa*-Projektgruppe.

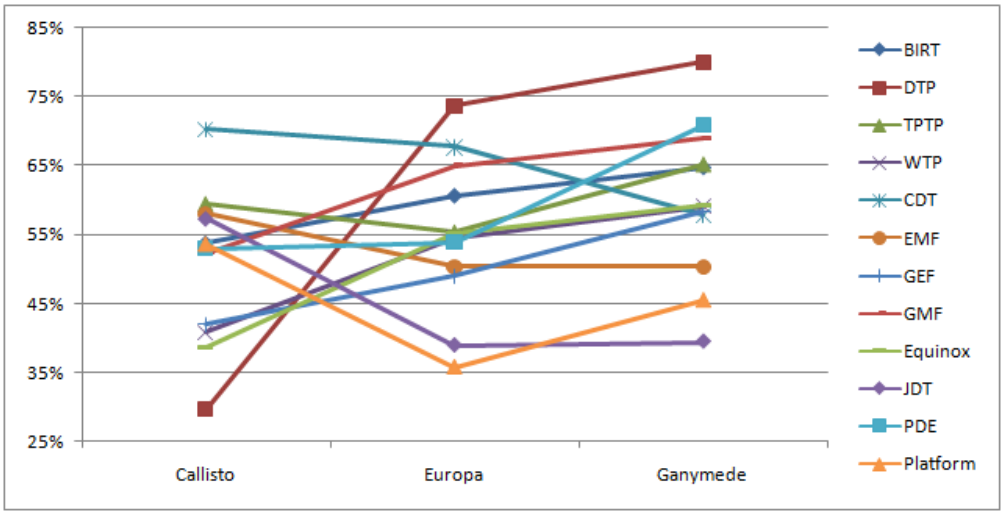

Abbildung 5.11: Qualitative Entwicklung der *Callisto*-Projektgruppe

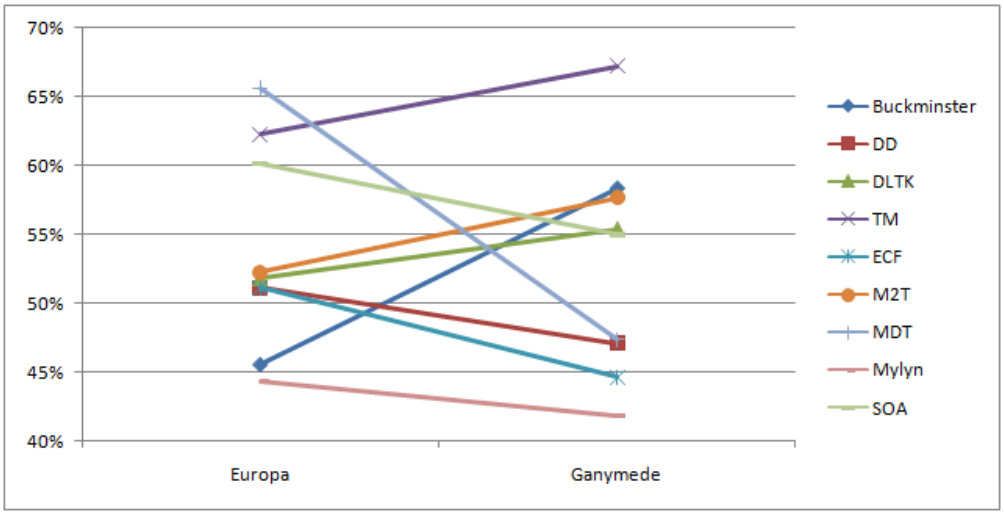

Abbildung 5.12: Qualitative Entwicklung der *Europa*-Projektgruppe

Wie unschwer zu erkennen ist, gibt es in beiden Gruppen sowohl Projekte, die sich im Verlauf der Zeit verbessert haben, als auch Projekte, die einer kontinuierlichen Verschlechterung unterlagen. Bemerkenswert ist jedoch die relative Stabilität der qualitativen Entwicklung: Lediglich das Kernprojekt *Platform* zeigt durch seine Verschlechterung zwischen *Callisto*- und *Europa*-Release und anschließender Verbesserung zum *Ganymede*-Release eine deutlich nicht-kontinuierliche Entwicklung seiner Prozessqualität. Dies belegt die bereits in Kapitel 3 zitierte Aussage von Bauer und Pizka [BP02], daß die Open Source-Entwicklung nicht unbedingt den in der Wahrnehmung der Öffentlichkeit vorherrschenden Vorurteilen hinsichtlich eines chaotischen Entwicklungsprozesses entspricht.

Über die Projekte der *Europa*-Projektgruppe läßt sich eine Kontinuitätsaussage aufgrund der lediglich zwei ausgewerteten Zeiträume nicht treffen. Allerdings ist zu erkennen, daß unter diesen Projekten ein höherer Anteil sich über den Gesamtzeitraum verschlechternder Projekte (56%) vorherrscht als bei der *Callisto*-Projektgruppe (33%). Die durchschnittliche Verbesserung und Verschlechterung der Projekte fällt hier jedoch auch moderater (6,8 Prozentpunkte durchschnittlicher Verbesserung und 7,0 Prozentpunkte durchschnittlicher Verschlechterung) aus als bei der *Callisto*-Projektgruppe (19,6 Prozentpunkte durchschnittliche Verbesserung gegenüber 11,8 Prozentpunkten durchschnittlicher Verschlechterung). Insgesamt überwiegt jedoch leicht der Anteil der Projekte, die sich über den gesamten Auswertungszeitraum verbessern konnten; 9 von 21 betrachteten Projekten waren einer Verschlechterung der Prozessqualität unterworfen, während deren durchschnittliche Verschlechterung bei 9,0 Prozentpunkten lag; die durchschnittliche Verbesserung der übrigen Projekte betrug hingegen 15,3 Prozentpunkte, was sich auch in der steigenden Gesamtqualität wiederspiegelt.

Eine Sonderstellung unter den Projekten, die im gesamten Auswertungszeitraum eine Prozessverbesserung erzielen konnten nimmt das Projekt *Data Tools Platform* (*DTP*) ein, das mit 50 Prozentpunkten Verbesserung eine mehr als doppelt so starke Verbesserung erreichte als das ‘zweitplatzierte’ Projekt *Equinox* mit 21 Prozentpunkten Verbesserung. Eine weitere Besonderheit stellt das Projekt *Platform* dar, da ausschließlich hier eine nicht-kontinuierliche Entwicklung der Qualität zu beobachten ist: Zwischen *Callisto*- und *Europa*-Release findet hier zunächst eine Qualitätsverschlechterung statt, zum *Ganymede*-Release hin verbessert sich die Qualität jedoch wieder leicht. Diese beiden Projekte sollen nun kurz einer näheren Betrachtung hinsichtlich der Hauptursachen dieser Entwicklung unterzogen werden.

Abbildung 5.13 zeigt den zeitlichen Verlauf der Qualitätskriterien der zweiten Ebene des *DTP*-Projekts.

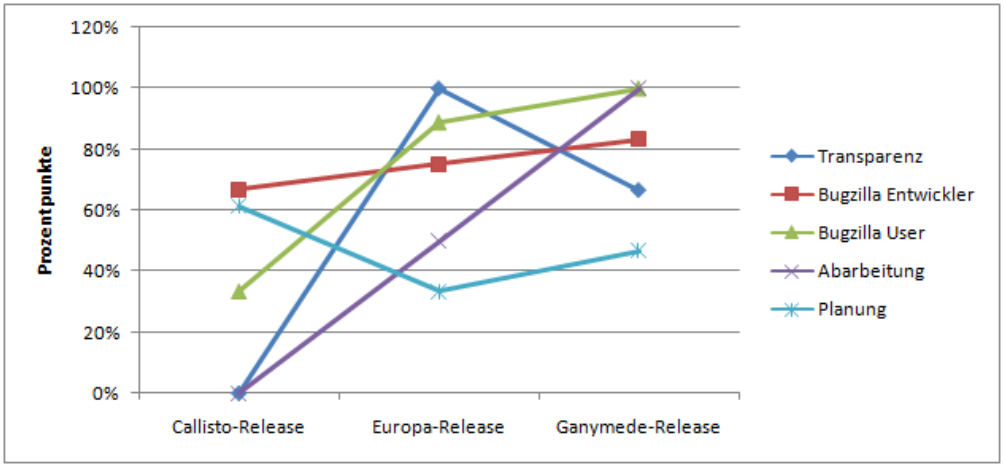

Abbildung 5.13: Verlauf der Qualitätskriterien des *DTP*-Projekts

Aus dem Diagramm ist ersichtlich, daß sich mit Ausnahme des Qualitätskriteriums ‘Planung’ insbesondere zwischen *Callisto*- und *Europa*-Release alle Kriterien verbessert haben. Eine besonders deutliche Verbesserung fand jedoch bei den Qualitätsmerk-

malen *Transparenz* und *Abarbeitung* statt. Insbesondere wird dies bei den Qualitätsmerkmalen des Kriteriums ‘Transparenz’ deutlich, wo zwischen *Callisto*- und *Europa*-Release eine Verbesserung von 0 auf 3 Punkte in beiden diesem Kriterium zugeordneten Qualitätsmerkmalen zu verzeichnen ist. So konnte das *DTP*-Projekt die Rate der zugewiesenen Fälle ohne *Target Milestone* von 85% aller Bugs im *Callisto*-Release auf 35% im *Europa*-Release mehr als halbieren während die Rate der behobenen Fälle ohne zugewiesenen *Target Milestone* im gleichen Zeitraum von 82% auf lediglich 20% sank; kein anderes Projekt konnte in diesem Zeitraum eine derartige Verbesserung des ‘Transparenz’-Qualitätskriteriums erzielen.

Anders stellt sich der Verlauf der Qualitätskriterien des *Platform*-Projekts dar, der in Abbildung 5.14 dargestellt ist.

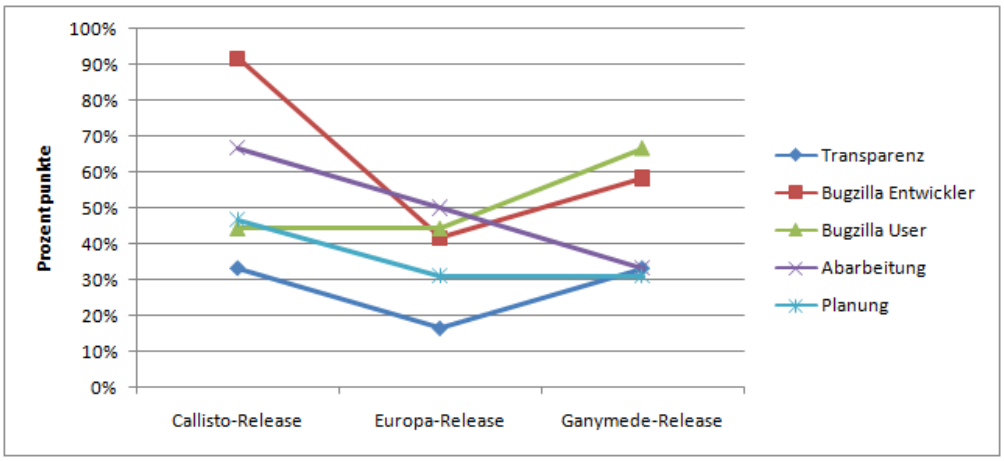

Abbildung 5.14: Verlauf der Qualitätskriterien des *Platform*-Projekts

Hier konnte im gesamten betrachteten Zeitraum nur das Qualitätskriterium ‘Bugzilla-Verwendung durch die Anwender’ effektiv an Qualität hinzugewinnen. Ein besonders deutlicher Einbruch der Qualität ist zwischen *Callisto*- und *Europa*-Release im Hinblick auf das Kriterium ‘Bugzilla-Verwendung durch die Entwickler’ festzustellen. Diese Verschlechterung ist ausschließlich auf das Qualitätsmerkmal ‘Unzugewiesene Bugs nach 7 Tagen’ zurückzuführen, da die beiden anderen in diesem Kriterium zusammengefaßten Qualitätsmerkmale ‘Rate der Bugs mit hoher Priorität’ und ‘Verhältnis von Priorität zu Schwere’ im gesamten Auswertungszeitraum stabil bei 3 bzw. 2 Punkten blieben. Gleichzeitig erhöhte sich jedoch der Anteil nach 7 Tagen noch nicht zugewiesener Bugs von 6% (*Callisto*-Release) auf 17% im *Europa*-Release und erreichte damit den höchsten Wert aller Projekte der *Callisto*-Projektgruppe in diesem Release.

Betrachtet man den Verlauf der Qualitätskriterien dieser beiden Projekte, so fällt auf, daß eine Verbesserung bzw. Verschlechterung der Gesamtqualität eines Projekts meist nicht auf eine einzelne, isolierte Ursache zurückzuführen ist; stattdessen ist bei einer Gesamtverbesserung bzw. -verschlechterung festzustellen, daß sich der Großteil aller Qualitätskriterien analog der Entwicklung der Gesamtqualität verhält. Dies trifft auch auf den Großteil der übrigen, hier nicht im Detail betrachteten Projekte zu. Daraus läßt

sich die Annahme herleiten, daß sich qualitative Verbesserungen dauerhaft nicht durch isolierte Verbesserung einzelner Qualitätskriterien oder -merkmale erzielen lassen – stattdessen führen Anstrengungen, die mit dem Ziel begangen werden, sämtliche Qualitätsmerkmale zu verbessern, effektiver und nachhaltiger an dieses Ziel, da zwischen den einzelnen Qualitätsmerkmalen und -kriterien offenbar eine gewisse Abhängigkeit besteht.

Wie in Abschnitt 2.1.3 bereits erwähnt, dient die vergleichende Auswertung mehrerer Projekte in erster Linie dem Ziel, Projekte mit überdurchschnittlich positiver Entwicklung zu identifizieren, um die Verfahrensweisen dieser Projekte zu analysieren und anhand der Ergebnisse dieser Analyse ähnliche Verfahrensweisen auf andere Projekte anzuwenden, um die Qualität dieser Projekte ebenfalls zu verbessern. Zur Identifikation dieser Projekte zeigt Tabelle 5.5 eine Rangfolge aller im Rahmen dieser Auswertung betrachteter Projekte, die anhand der durchschnittlichen Qualitätsverbesserung über sämtliche Auswertungszeiträume ermittelt wurde.

Rang	**Projekt**	**Veränderung**	**Rang**	**Projekt**	**Veränderung**
1	DTP	50%	12	DLTK	4%
2	Equinox	21%	13	Mylyn	-2%
3	WTP	18%	14	DD	-4%
3	PDE	18%	15	SOA	-5%
5	GMF	17%	16	ECF	-6%
6	GEF	16%	17	EMF	-8%
7	Buckminster	13%	17	Platform	-8%
8	BIRT	11%	19	CDT	-13%
9	TPTP	6%	20	JDT	-18%
10	M2T	5%	20	MDT	-18%
10	TM	5%			

Tabelle 5.5: Rangfolge der Projekte hinsichtlich Qualitätsentwicklung

Unerwartet an dieser Rangfolge ist das vergleichsweise schlechte Abschneiden der beiden in der Kern-*Classification Eclipse* enthaltenen Projekte *Platform* und *JDT*, die zusammen mit dem Projekt *PDE*, das sich jedoch auf den vorderen Plätzen befindet, die Entwicklung des eigentlichen Kerns der Eclipse-Platform vorantreiben.

Eine sorgfältige Analyse der Vorgehensweisen in den Projekten der ersten 8 Plätze, insbesondere jedoch der Projekte *Equinox* und *DTP* könnte nun wichtige Anhaltspunkte liefern, auf welche Weise auch die anderen Projekte ihre Prozessqualität deutlich steigern könnten. Eine derartige Analyse würde jedoch deutlich über den Rahmen dieser Arbeit hinausführen – aus diesem Grund befassen sich die nächsten Abschnitte nicht mit einer tiefergehenden Analyse dieser Projekte sondern zeigen weitere Zusammenhänge zwischen der Prozessqualität und verschiedenen Projekteigenschaften sowie zwischen den unterschiedlichen erreichten Werten im Zusammenhang mit den unterschiedlichen Qualitätskriterien.

5.5.5 Qualität in Relation zur Projektaktivität

Eine Frage, die sich bei der Betrachtung eines Vergleichs der Qualitäten unterschiedlich großer und unterschiedlich aktiver Projekte unmittelbar stellt ist die nach einem Zusammenhang zwischen Projektqualität und Projektgröße bzw. -aktivität. Dieser Frage soll in diesem Abschnitt nachgegangen werden. Sofern tatsächlich ein erkennbarer Zusammenhang besteht, könnten zu diesem aufgrund verschiedener Überlegungen gewisse Erwartungen erwachsen, die im Folgenden überprüft werden sollen.

Ein großes, besonders aktives Projekt (wobei sich die Aktivität eines Projekts einerseits durch die Anzahl der *Commits*, d.h. Codeeinreichungen in das Versionskontrollsystem, andererseits aber auch durch die Anzahl in einem bestimmten Zeitraum behobener Bugs bestimmen läßt) an dem viele Entwickler beteiligt sind benötigt eine straffere Organisation als ein kleines Projekt, an dem nur wenige Personen beteiligt sind und in dem nur wenige *Commits* und Fehlerbehebungen stattfinden. Von daher könnte man davon ausgehen, daß die Qualität eines Projekts mit dessen Aktivität zunimmt. Andererseits haben kleinere, weniger aktive Projekte den Vorteil, daß nur eine geringe Anzahl von Personen (sowohl Entwickler wie auch Anwender) koordiniert werden müssen und aufgrund der geringeren Anzahl von Bearbeitern ein konsequenteres Einhalten der Prozessvorgaben möglich ist.

Es ergeben sich also zwei prinzipiell mögliche Annahmen (unter der Voraussetzung, daß ein deutlicher Zusammenhang zwischen Qualität und Projektaktivität erkennbar ist): Kleinere, wenig aktive Projekte besitzen aufgrund des ‘Chaos-Faktors’ in großen Projekten eine bessere Qualität, oder größere, sehr aktive Projekte besitzen eine höhere Qualität, da die Einhaltung eines hohen Qualitätsfaktors eine unbedingte Voraussetzung für den Erfolg eines großen Projekts ist. Die Abbildung 5.15 stellt den Zusammenhang zwischen der Projektaktivität, gemessen an der Anzahl der *Commits* im Verhältnis zur erreichten Qualität im *Ganymede*-Release dar.

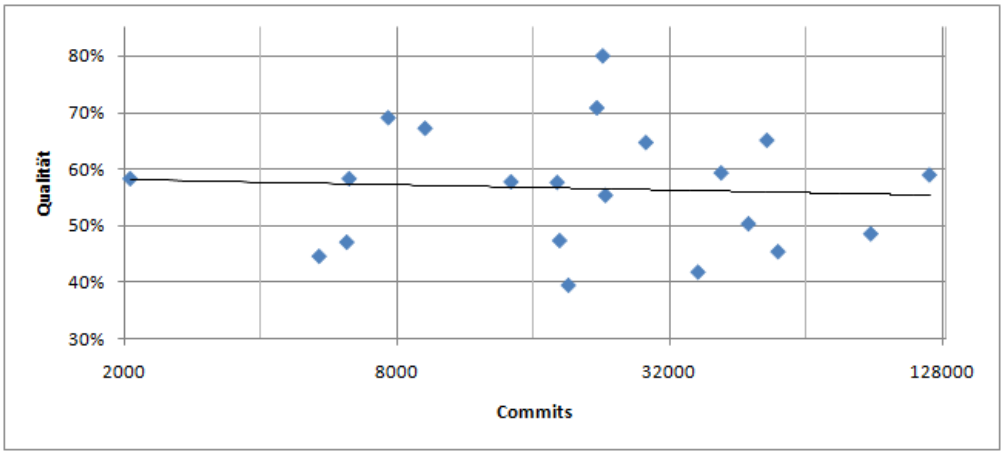

Abbildung 5.15: Verhältnis von Projektaktivität (Anzahl der Commits) und Qualität (*Ganymede*-Release)

Die zugunsten der Übersichtlichkeit logarithmisch skalierte Ordinate enthält die Anzahl der *Commits* (aus Eclipse Dash [Das]), die linear skalierte Abszisse gibt die Qualität des Produkts in Prozentpunkten an. Zu erkennen ist hier, daß Projekte mit mittlerer Aktivität zwischen 8000 und 32000 *Commits* tendenziell am besten abschneiden; in dieser Gruppe befinden sich nur zwei Projekte unterhalb der Trendlinie, während 7 Projekte eine eher höhere Qualität aufweisen. Im Bereich der wenig aktiven sowie den besonders aktiven Projekte verringert sich die Streuung zwischen qualitativ hochwertigen Projekten und solchen mit geringerer Qualität. Die Trendlinie weist jedoch auf eine leichte Tendenz zu sich verringernder Prozessqualität bei steigender Aktivität hin. Zu beachten ist jedoch, daß Ausgleichsgeraden dieser Form (in der Abbildung wurde eine logarithmische Ausgleichsgerade gewählt) grundsätzlich mit einem gewissen Fehler behaftet sind – so besteht durchaus die Möglichkeit, daß es sich bei dem beobachteten Effekt um ein Zufallsprodukt handelt. Aufgrund der geringen Zahl an Meßpunkten entzieht sich diese Beobachtung jedoch einer Signifikanzüberprüfung mit Hilfe gängiger statistischer Tests.

Ein ähnliches Bild ergibt der Zusammenhang zwischen der Anzahl behobener Bugs und der Prozessqualität, die in Abbildung 5.16 dargestellt ist.

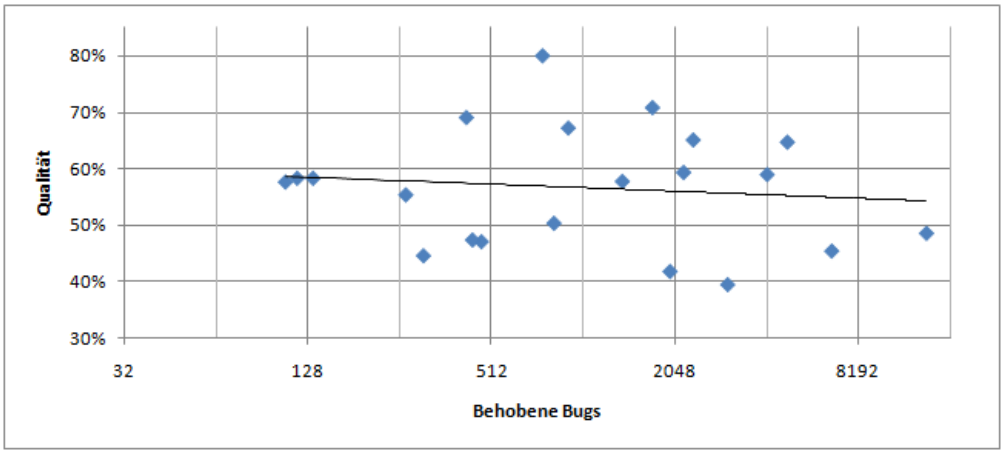

Abbildung 5.16: Verhältnis von Projektaktivität (Anzahl der behobenen Bugs) und Qualität (*Ganymede*-Release)

Auch hier zeigt sich ein ähnliches Ergebnis; der Abfall der Qualität bei steigender Projektaktivität ist hier jedoch noch stärker zu erkennen als bei der Auswertung des Verhältnisses zwischen *Commit*-Aktivität und Prozessqualität. Beide Auswertungen können jedoch nur dann als äquivalent angesehen werden, wenn ein annähernd linearer Zusammenhang zwischen der Anzahl der *Commits* und der Anzahl der im selben Zeitraum behobenen Bugs besteht. Die Überprüfung dieses Zusammenhangs (Abbildung 5.17) bestätigt diesen Zusammenhang weitgehend.

Die Tatsache, daß Projekte mit mittlerer Aktivität tendenziell eine bessere Qualität aufweisen als solche mit geringer respektive sehr hoher Aktivität läßt sich durch eine Überlegung leicht verdeutlichen: Bei Projekten, die sehr geringe Aktivität aufweisen, arbeitet meist nur eine geringe Anzahl von Entwicklern an einer ebenfalls geringen

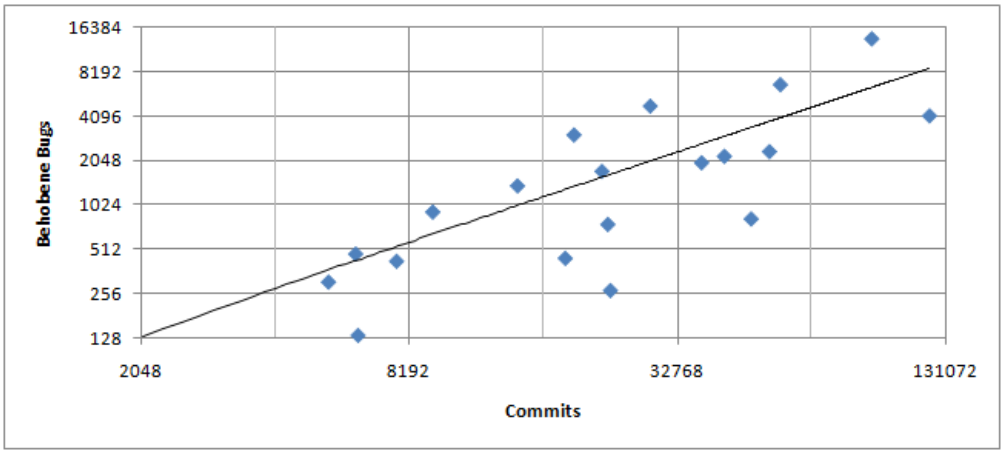

Abbildung 5.17: Verhältnis von behobenen Bugs und Anzahl der Commits (*Ganymede*-Release)

Anzahl von Funktionen; eine starke Steuerung des Projekts ist hier meist nicht erforderlich, so daß es nicht notwendig ist, auf die Einhaltung bestimmter Prozesskriterien besonders großen Wert zu legen – hierzu zählen beispielsweise die Funktionen zur Priorisierung von Bug-Einträgen; auch ist davon auszugehen, daß es in einem wenig aktiven Projekt länger dauert, bis ein gemeldeter Bug behoben wurde (dieser Effekt tritt beispielsweise bei *GEF* auf: Bei nur 2065 Commits dauert es im Schnitt 28 Tage, bis ein Fehler behoben wird; im Vergleich dazu dauert dies im Projekt *BIRT* bei 28455 *Commits* lediglich knapp 4 Tage). Nicht zu vernachlässigen ist auch die Tatsache, daß es sich bei wenig aktiven Projekten oftmals um sehr junge Projekte handelt, die noch kein intensives Projektmanagement etabliert haben; bei älteren Projekten, die nur noch sporadisch weiterentwickelt werden (und dadurch ebenfalls eine geringe Aktivität aufweisen) wurde das Projektmanagement möglicherweise bereits aufgelöst.

Bei Projekten mit hoher Aktivität macht sich hingegen die Tatsache bemerkbar, daß solche Projekte aufgrund der großen Zahl von Entwicklern und Anwendern nur noch schwer zu steuern sind. Auch bei einem etablierten und aktiven Projektmanagement führt diese Diversität von Projektmitgliedern und den daraus resultierenden Schwierigkeiten dazu, daß sich diese Projekte qualitätiv eher im Mittelmaß bewegen.

Die Projekte mit mittlerer Aktivität hingegen stellen ein Zwischending zwischen den beiden anderen Projektgruppen dar. Einerseits sind sie zu groß und zu aktiv, um auf ein klares Projektmanagement zu verzichten, andererseits treten die Probleme, die sich bei sehr aktiven Projekten ergeben, hier noch nicht auf; dies führt dazu, daß diese Projekte eine überdurchschnittlich hohe Qualität erreichen können. Letzlich besteht aber weiterhin Grund zu der Annahme, daß sich höhere Aktivität tendenziell eher negativ auf die Prozessqualität eines Projekts auswirkt. Eine eindeutige statistische Aussage läßt sich jedoch nur dann treffen, wenn eine größere Anzahl von Meßpunkten zur Verfügung steht.

5.5.6 Verteilung der Qualitätskriterien

Dieser Abschnitt befaßt sich mit dem zeitlichen Verlauf der Qualitätskriteriene der zweiten Ebene. Eine weitestgehend konstante Gesamtqualität muß nicht zwangläufig darauf hindeuten, daß sich unter den verschiedenen untergeordneten Qualitätskriterien keine Veränderungen oder Verschiebungen ergeben haben; diese Veränderungen werden hier näher beleuchtet. Zu diesem Zweck werden Netzdiagramme mit 5 Achsen (jeweils eine für eines der 5 Qualitätskriterien der zweiten Ebene) verwendet; die Fläche innerhalb des Diagramms gibt dabei annähernd die Gesamtqualität der Projekte wieder – je weiter ein Wert am Rand des Diagramms liegt, desto höher die Qualität dieses Qualitätskriteriums. Je symmetrischer das Diagramm zum Mittelpunkt ist, desto ausgeglichener ist die Verteilung der Qualität auf die einzelnen Qualitätskriterien.

Verglichen werden hierbei jeweils die Durchschnittswerte der Projekte der *Callisto*- bzw. der *Europa*-Projektgruppe, da es sich bei ersteren um überwiegend große und etablierte Projekte handelt, die nahe am Kern des Eclipse-Projekts angesiedelt sind; hingegen bestehen die Projekte der *Europa*-Projektgruppe hauptsächlich aus jüngeren Projekten, die noch größere Instabilitäten aufweisen. Eine gemeinsame Betrachtung dieser beiden Projektgruppen würde daher verfälschend auf die gewonnenen Aussagen wirken.

Abbildung 5.18 zeigt die Entwicklung der Qualitätskriterien der *Callisto*-Projektgruppe im Verlauf der Zeit.

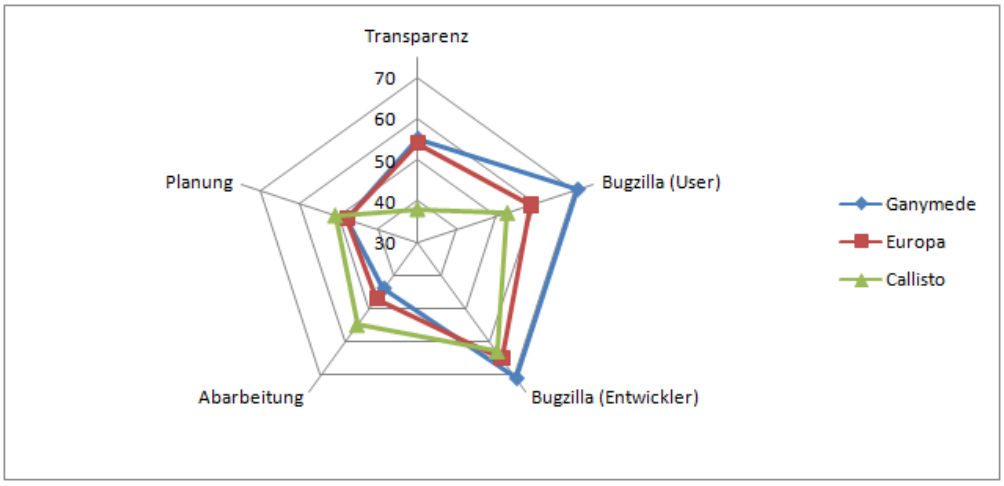

Abbildung 5.18: Verteilung der Qualitätskriterien der *Callisto*-Projektgruppe im Vergleich der Releases

Deutlich zu erkennen ist ein gravierender Unterschied in der Verteilung der Qualitäten zwischen den Releases *Callisto* und *Europa*: Während das Qualitätskriterium ‘Transparenz’ einen deutlichen Sprung nach oben macht, fallen die Qualitätskriterien ‘Abarbeitung’ und ‘Planung’ zwischen diesen Releases ab. Offenbar hat zwischen diesen beiden Releases bei der überwiegenden Zahl von Projekten ein Wechsel des Fokus hin zu anderen Qualitätsmerkmalen stattgefunden. Eine kontinuierliche Steige-

rung ist hingegen zwischen allen drei Releases hinsichtlich der Bugzilla-bezogenen Qualitätskriterien zu erkennen: Sowohl das Kriterien ‘Bugzilla-Verwendung durch die Entwickler’ als auch ‘Bugzilla-Verwendung durch die Anwender’ haben sich zwischen den Releases kontinuierlich verbessert. Der erkennbare Schwerpunkt auf diesen beiden Qualitätskriterien überrascht nicht: Zur Verwendung des Bugzilla-Werkzeugs existiert ein Dokument [EBU], das sowohl Entwicklern wie auch Anwendern den korrekten Umgang mit diesem Werkzeug sowie die im Rahmen der Verwendung benötigen Arbeitsabläufe detailliert erläutert. Abgesehen vom Schwerpunkt auf die Verwendung des Bugzilla-Werkzeugs weisen die Diagrammlinien keine ‘Dellen’ auf, die auf eine stark unausgeglichene Verteilung der Qualität auf die einzelnen Qualitätskriterien hindeuten.

Vollkommen unterschiedlich stellt sich die Situation hingegen bei den Projekten aus der *Europa*-Projektgruppe dar (Abbildung 5.19).

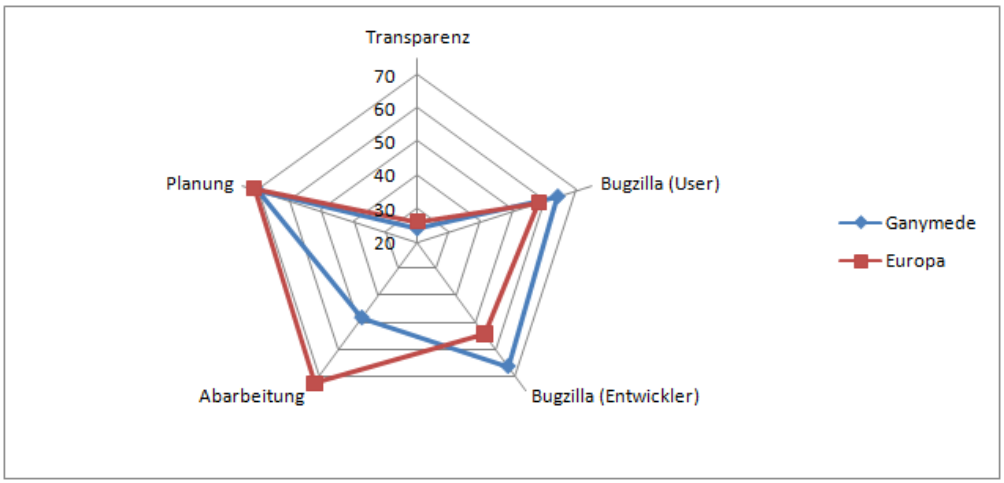

Abbildung 5.19: Verteilung der Qualitätskriterien der *Europa*-Projektgruppe im Vergleich der Releases

Hier ist nicht nur eine Asymmetrie der Diagrammlinien, insbesondere aufgrund äußerst geringer Werte beim Qualitätskriterium ‘Transparenz’ zu erkennen; zwischen *Europa*- und *Ganymede*-Release ändert sich die Form des Diagramms erheblich, da eine Steigerung der Qualität in den *Bugzilla*-Qualitätskriterien mit einem starken Einbruch der Qualität im Kriterium ‘Abarbeitung’ einhergeht. Die Werte für ‘Planung’ und ‘Transparenz’ bleiben weitestgehend konstant. Dieser Trend ist zwar auch in Abbildung 5.18 für die *Callisto*-Projektgruppe erkennbar, jedoch fällt er dort bei weitem nicht so drastisch aus. Offenbar besitzen die älteren, bereits im *Callisto*-Release enthaltenen Projekte eine deutlich stärkere Stabilität hinsichtlich der Qualität als die im Vergleich junge *Europa*-Projektgruppe.

Auch wenn man die Diagramme von *Callisto*- und *Europa*-Projektgruppe direkt miteinander vergleicht (Abbildungen 5.20 und 5.21), fallen Unterschiede unmittelbar auf: lediglich die Qualitäten bei der Verwendung von Bugzilla sind annähernd vergleichbar, dagegen unterscheiden sich die übrigen Qualitätskriterien zwischen den zwei be-

trachteten Projektgruppen erheblich. Zum *Ganymede*-Release nähert sich das Qualitätskriterium ‘Abarbeitung’ der *Europa*-Projektgruppe jedoch bereits dem der *Callisto*-Projektgruppe an – auf diese Annäherung wird im folgenden Abschnitt noch näher eingegangen.

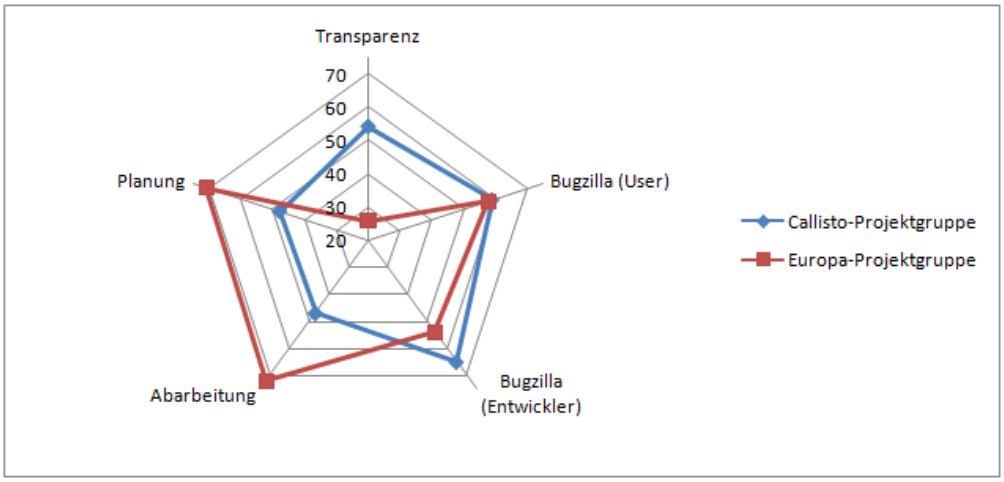

Abbildung 5.20: Verteilung der Qualitätskriterien des *Europa*-Release verglichen zwischen *Callisto*- und *Europa*-Projektgruppe

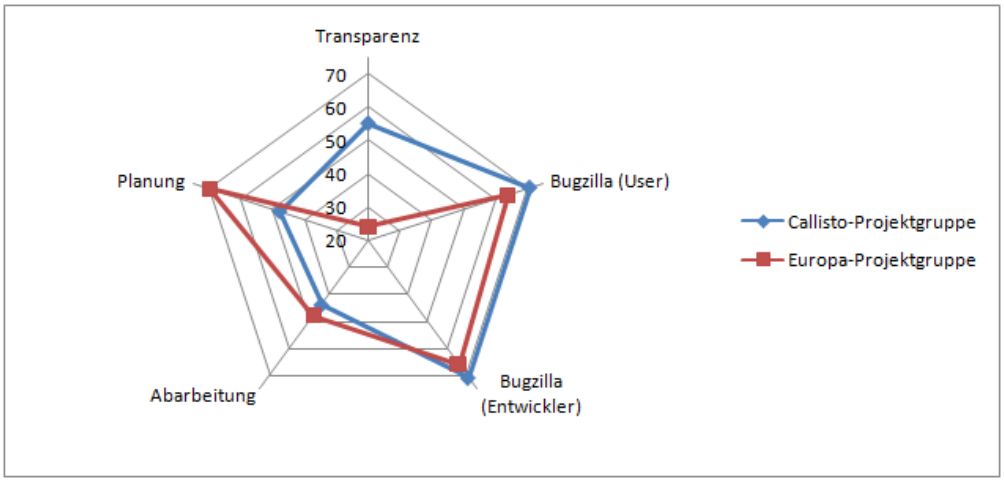

Abbildung 5.21: Verteilung der Qualitätskriterien des *Ganymede*-Release verglichen zwischen *Callisto*- und *Europa*-Projektgruppe

Der Vergleich zwischen den Verteilungen der Qualität auf die verschiedenen Qualitätskriterien der zwei betrachteten Projektgruppen macht deutlich, daß erhebliche Unterschiede zwischen den Projekten hinsichtlich dieser Verteilung bestehen. Dies ist nicht ungewöhnlich, da es den Projekten des Eclipse-Ecosystems ausdrücklich freigestellt ist, die tatsächlich angewandten Prozesse an die Anforderungen der jeweiligen Prozesse anzupassen – lediglich bei den Qualitätskriterien, die sich auf sorgsam dokumentierte Vorgänge (Verwendung des Bugzilla-Werkzeugs) beziehen, erreichen alle Projekte vergleichbare Werte.

5.5.7 Committer-bezogene im Vergleich zu User-bezogenen Qualitätskriterien

Wie im vorangegangenen Abschnitt beschrieben, besteht insbesondere bei der *Europa*-Projektgruppe eine starke Asymmetrie zwischen den verschiedenen Qualitätskriterien. Aus diesem Grund soll in diesem Abschnitt der Zusammenhang zwischen der Qualität bei anwenderbezogenen und der Qualität bei entwicklerbezogenen Qualitätskriterien untersucht werden. Aus Abbildung 5.22 geht hervor, daß die Verteilung der Qualität zwischen anwender- und entwicklerbezogenen Qualitätskriterien recht ausgeglichen ist.

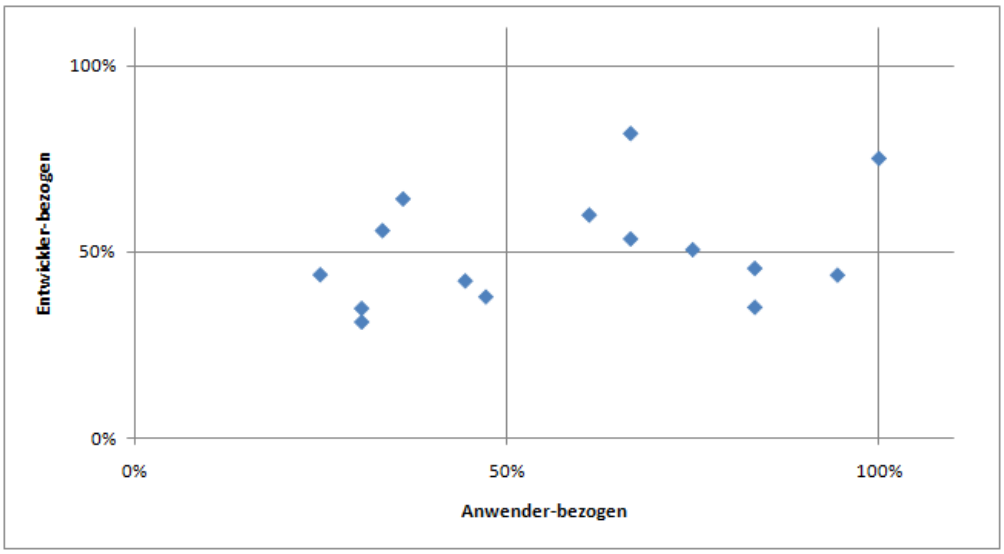

Abbildung 5.22: Relation zwischen Anwender- und Entwicklerbezogenen Qualitätskriterien (*Callisto*-Projektgruppe im *Europa*-Release)

In den Quadranten links unten und rechts oben, die auf eine ausgeglichene Verteilung der Qualität auf diese Qualitätskriterien hindeuten befinden sich jeweils 5 Projekte, während bei lediglich 5 weiteren Projekten ein (wenn auch nur leichter) Trend zu den Entwickler-bezogenen (2 Projekte) bzw. Anwender-bezogenen (3 Projekte) Qualitätskriterien festzustellen ist. Anders stellt sich die Situation jedoch bei den im *Europa*-Release erstmals dem *Simultaneous Release* beigetretenen Projekten dar (Abbildung 5.23).

Wie aus den Diagrammen im vorangehenden Abschnitt bereits ersichtlich ist hier ein deutlicher Trend zum Quadranten links oben (hohe Qualität bei Entwickler-bezogenen Qualitätskriterien, hingegen geringe Qualität bei Anwender-bezogenen Qualitätskriterien) zu erkennen. Aus dieser Tatsache ließe sich der Schluß ziehen, daß jüngere Projekte zunächst eher in der Lage sind, die entwicklerspezifischen Qualitätskriterien zu erfüllen und Forderungen an die anwenderbezogene Qualität anfangs weniger stark berücksichtigt werden. Zwecks Überprüfung dieser Annahme stellt Abbildung 5.24 den gleichen Zusammenhang, betrachtet über den Zeitraum des *Ganymede*-Release dar.

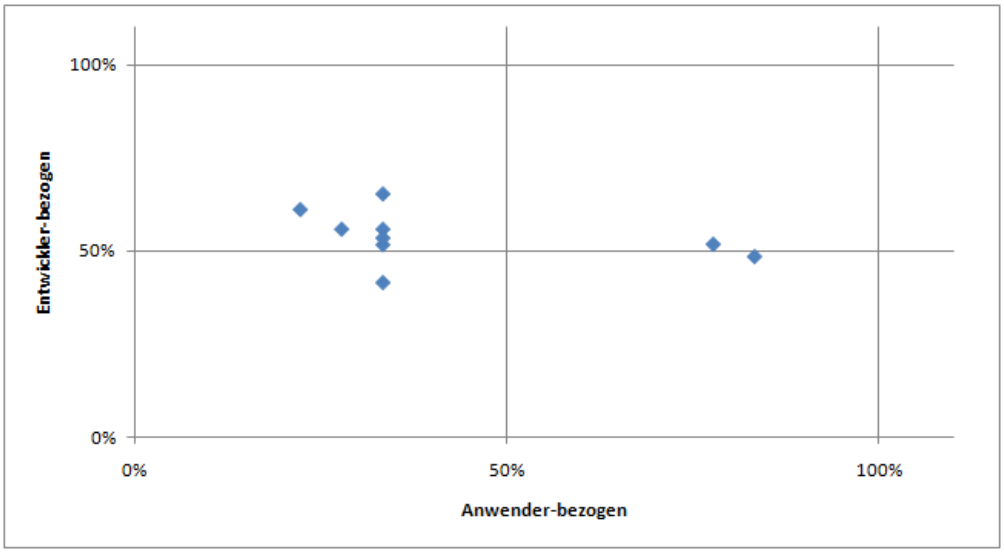

Abbildung 5.23: Relation zwischen Anwender- und Entwicklerbezogenen Qualitätskriterien (*Europa*-Projektgruppe im *Europa*-Release)

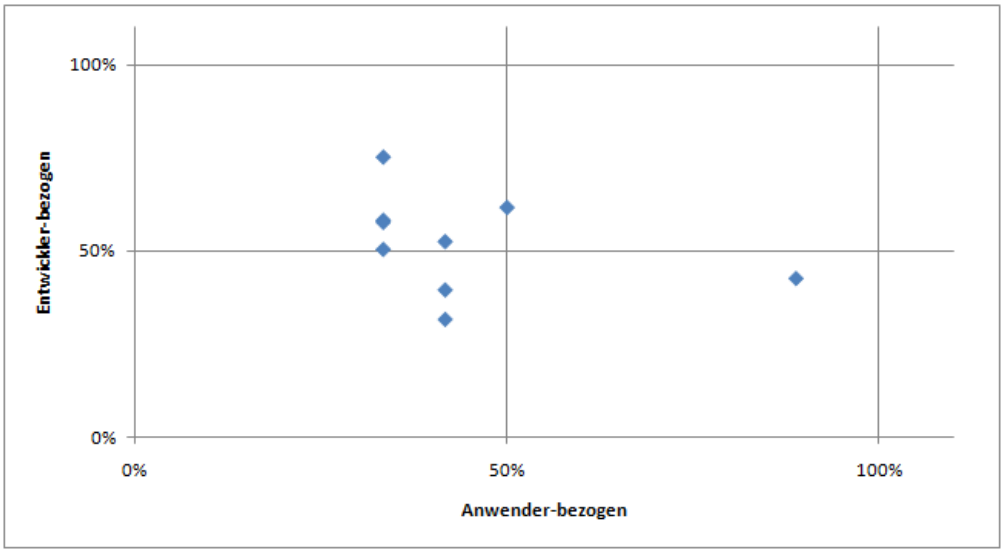

Abbildung 5.24: Relation zwischen Anwender- und Entwicklerbezogenen Qualitätskriterien (*Europa*-Projektgruppe im *Ganymede*-Release)

Hier ist bereits deutlich eine Verschiebung der Qualitäten hin zu den Anwender-bezogenen Qualitätskriterien zu erkennen, jedoch hat sich auch die mittlere Bewertung der entwicklerbezogenen Qualitätskriterien leicht verringert. Einen Ausreißer stellt lediglich das Projekt *Target Management* dar, das mit 89% Erfüllung der anwenderbezogenen und 42% Erfüllung der entwicklerbezogenen Qualitätskriterien einziges Projekt im rechten unteren Quadranten ist.

Anhand dieser Tatsache läßt sich die Korrektheit der oben getroffenen Annahme belegen: Im Laufe der Zeit findet bei den betrachteten Projekten eine kontinuierliche Entwicklung weg von den Entwickler-bezogenen Qualitätskriterien (die sich im Durchschnitt sogar verschlechtern) hin zu den Anwender-bezogenen Qualitätskriterien statt. In wiefern sich dieser Trend jedoch fortsetzen wird entzieht sich derzeit noch der Beurteilung eines Beobachters, da die entsprechenden Werte erst mit Erscheinen des nächsten Eclipse-Releases im Juli 2009 meßbar sind.

5.5.8 Informative Metriken

Zwei Metriken des Qualitätsmodells nehmen gegenüber den anderen eine Sonderstellung ein: die Qualitätsmerkmale 'Anteil geschlossener Bugs' sowie 'Anteil Bugs im Zustand `VERIFIED`', die zum Qualitätskriterium 'Informative Metriken' zusammengefaßt wurden. Diese Metriken befassen sich mit Eigenschaften der betrachteten Projekte, die keine verrechenbaren Qualitätsmerkmale im Sinne der übrigen Metriken darstellen. Vielmehr befassen sie sich mit der Analyse des Entwicklungsprozesses im Sinne der Fragestellung, ob sämtliche Möglichkeiten, die das Werkzeug *Bugzilla* zur Steuerung des Prozesses bietet, tatsächlich genutzt werden. Aus diesem Grund fließen die Ergebnisse dieser Auswertungen nicht in die Gesamtqualität ein.

Beide Metriken befassen sich mit der Verwendung vordefinierter Zustände in *Bugzilla*. Gemäß des Zustandsdiagramms (Abbildung 5.2) und der Anleitung zur Verwendung von Bugzilla [EBU] können als `RESOLVED` gekennzeichnete Bugs, nachdem durch einen Tester sichergestellt wurde, daß der Fehler tatsächlich korrekt behoben wurde, in den Zustand `VERIFIED` überführt werden. Abschließend sollten Bugs zur Wahrung der Übersichtlichkeit spätestens zum Zeitpunkt eines jeden Releases auf den Status `CLOSED` gesetzt werden. Die Idealvorstellung besteht darin, daß jeder Fehler, der als `Resolution` den Wert `FIXED` besitzt, zunächst den Status `VERIFIED` und schließlich den Status `CLOSED` erhält. Der Anteil von Bugs, der in den Zustand `CLOSED` übergeht ohne vorher als `VERIFIED` gekennzeichnet worden zu sein, wird durch die Metrik 'Anzahl der geschlossenen Fälle ohne Zustand `VERIFIED`' gemessen. Zusätzlich sollte überprüft werden, welcher Anteil von Bugs überhaupt im Laufe seines Lebenszyklus den Zustand `CLOSED` erreicht; dies wird durch die Metrik 'Anzahl von Übergängen in den Zustand `CLOSED`' überprüft.

Interessant sind in diesem Zusammenhang besonders die Fragestellungen 'Wird der Zustand `CLOSED` in den Projekten konsequent verwendet?', 'In welchem Ausmaß wird vom Zustand `VERIFIED` gebrauch gemacht?' und 'Besteht ein Zusammenhang zwischen der Verwendung des Zustands `CLOSED` und dem Zustand `VERIFIED`?'

Antworten auf diese Fragen liefert die folgende Auswertung der vorab erwähnten Metriken.

Verwendung des Zustands CLOSED

Zunächst soll ermittelt werden, wie verbreitet die Verwendung des Zustands CLOSED allgemein und in den verschiedenen Projekten ist. Als Referenzwerte für die zugrundeliegende Metrik ergeben sich folgende Prozentzahlen von Bugs, die den Zustand CLOSED erreichen:

Maximum	63,1%
3. Quartil	43,82%
Median	43,2%
1. Quartil	13,73%
Minimum	2,88%

Tabelle 5.6: Referenzwerte der Metrik 'Übergänge in den Zustand CLOSED'

Die Betrachtung dieser Referenzwerte offenbart bereits die erste Erkenntnis über die Verwendung von CLOSED: Kein Projekt des *Callisto*-Releases verwendet diesen Zustand in letzter Konsequenz. Darüber hinaus fällt auf, daß die Bandbreite der Werte von 'So gut wie kein Bug wird geschlossen' bis hin zu 'Etwa jeder zweite Bug wird geschlossen' verhältnismäßig groß ausfällt.

Betrachtet man nun die ermittelten Punkteverteilungen (Abbildung 5.25) aller Projekte, die bereits im *Callisto*-Release enthalten waren, so fällt ein weiteres Charakteristikum auf:

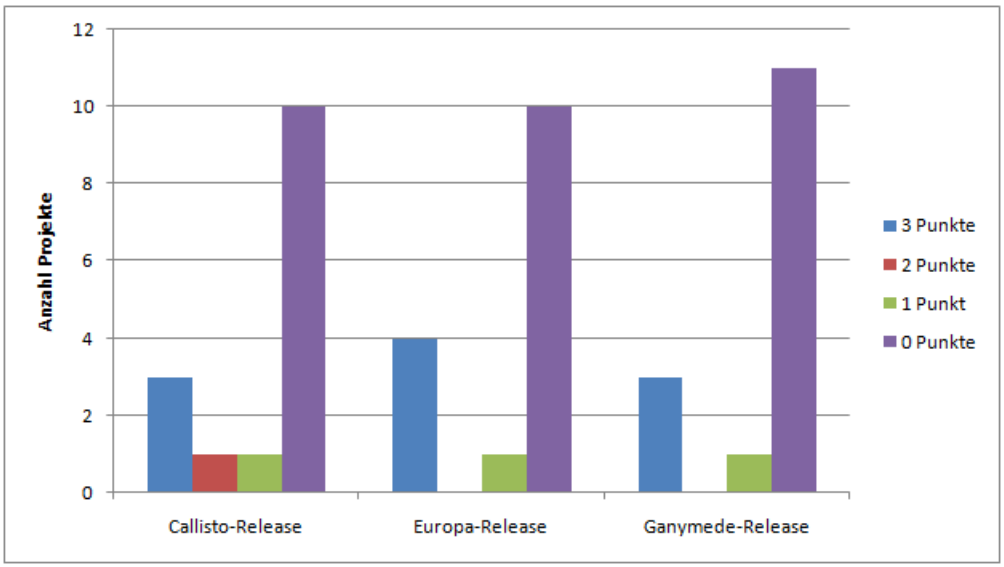

Abbildung 5.25: Punkteverteilung der *Callisto*-Projektgruppe hinsichtlich der geschlossenen Bugs. Die Abszisse benennt die Anzahl der Projekte mit der entsprechenden Punktzahl

Die hohe Anzahl von Projekten mit 0 Punkten bei dieser Qualitätseigenschaft deutet darauf hin, daß der überwiegende Anteil an Projekten weniger als 14% der behobenen

Fehler tatsächlich in den Zustand CLOSED überführt. Darüber hinaus ist zu erkennen, daß o gut wie keine zeitliche Variation der Punktzahlen stattfindet; Projekte, die von Anfang an Gebrauch vom Zustand CLOSED gemacht haben, behielten diese Verfahrensweise bei, die übrigen Projekte führten sie auch im Laufe der Zeit nicht ein. Bemerkenswert hier ist auch das fast vollständige Fehlen von Projekten mit einem oder zwei Punkten; offenbar wird der Zustand CLOSED entweder mit guter Regelmäßigkeit oder sehr konsequent nicht verwendet. Bei den Projekten, die erstmals am *Europa*-Release teilnamen, stellt sich die Situation analog dar (Abbildung 5.26).

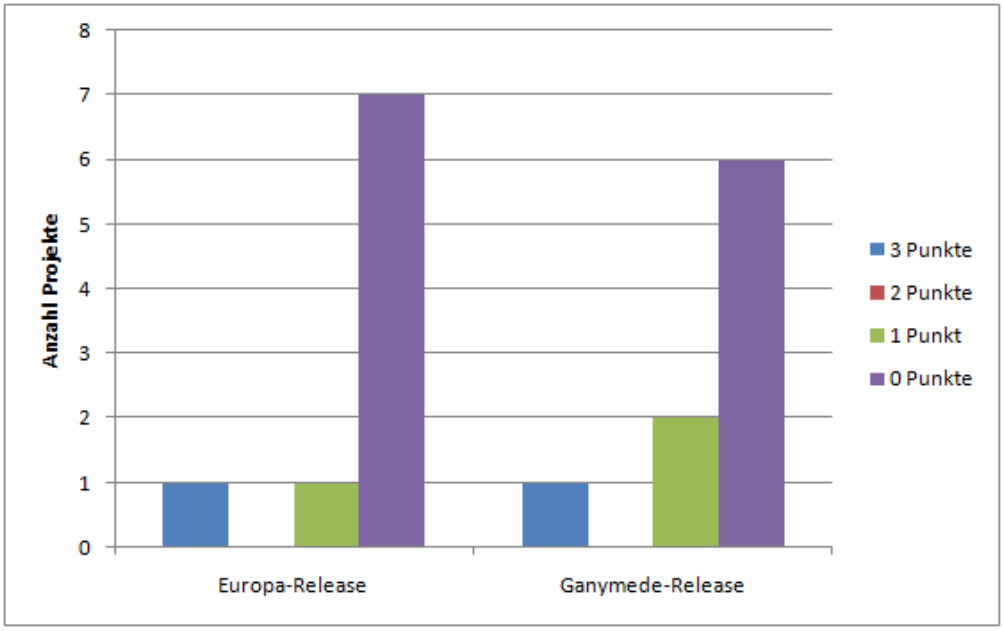

Abbildung 5.26: Punkteverteilung der *Europa*-Projektgruppe hinsichtlich der geschlossenen Bugs

Aus dieser Auswertung läßt sich folgern, daß lediglich 5 Projekte die konsequente Verwendung des Zustands CLOSED verfolgen und dauerhaft mehr als die Hälfte ihrer Bugs in diesen Zustand überführen. Namentlich sind dies *BIRT*, *TPTP*, *WebTools* und *Visual Editor* sowie *Buckminster* aus der Menge der Projekte, die erstmals am *Europa*-Release teilgenommen haben.

Gebrauch des Zustands VERIFIED

Der Zustand VERIFIED dient zur Kennzeichnung von Bugs, die nach ihrer Behebung einen definierten Testprozess durchlaufen haben oder deren korrekte Behebung durch den Reporter oder einen Dritten überprüft wurde. Gemäß Abbildung 5.2 handelt es sich um einen optionalen Zustand, der entsprechend dem Bedarf des Projekts auch übersprungen werden darf. Ein hoher Anteil von Bugs, die vor ihrer Schließung den Zustand VERIFIED besaßen, deutet jedoch auf eine gut entwickelte Testkultur hin. Aus diesem Grund wird mit Hilfe der Metrik ‘Anzahl der geschlossenen Fälle ohne Zustand VERIFIED’ ermittelt, wie verbreitet die Verwendung dieses Zustandes in den

Eclipse-Projekten ist. Die Referenzwerte ergeben sich gemäß der folgenden Tabelle als Prozentsätze der geschlossenen Fälle, die nicht aus dem Zustand `VERIFIED` in den Zustand `CLOSED` übergehen.

Maximum	97,96%
3. Quartil	93,74%
Median	90,39%
1. Quartil	87,63%
Minimum	61,66%

Tabelle 5.7: Referenzwerte der Metrik ‘Geschlossene Fälle ohne Zustand `VERIFIED`’

Die Betrachtung dieser Referenzwerte zeigt unmittelbar eine Häufung zwischen 87% und 98% was bedeutet, daß fast alle Bugs direkt aus dem Zustand `RESOLVED` in den Zustand `CLOSED` übergehen, ohne vorher in *Bugzilla* explizit als verifiziert gekennzeichnet zu werden. Dies bedeutet jedoch nicht zwangsläufig, daß keine Tests hinsichtlich der korrekten Behebung von Bugs durchgeführt werden, jedoch leidet die Transparenz innerhalb des Bugtrackers darunter, daß derartig überprüfte Bugs nicht explizit als verifiziert gekennzeichnet werden.

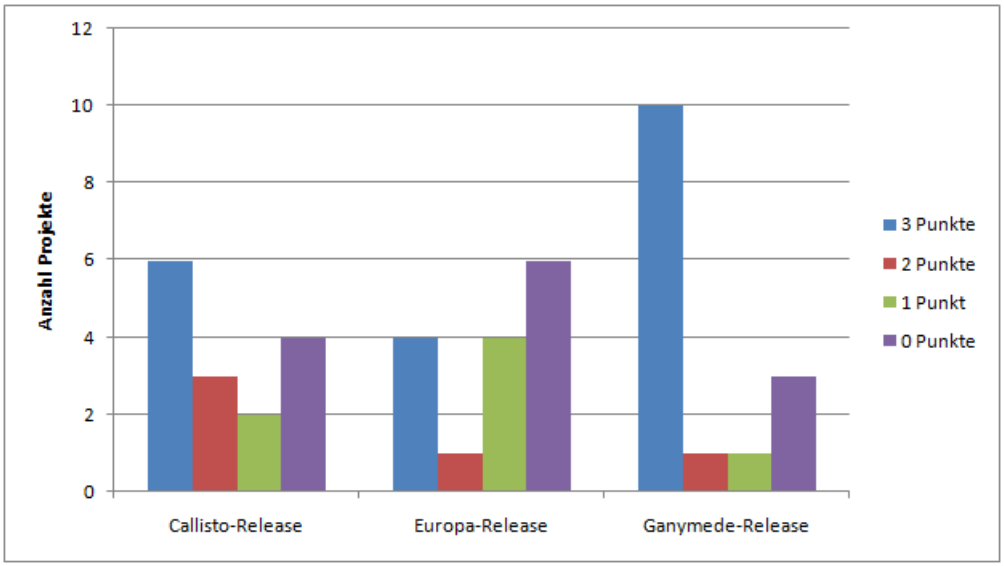

Abbildung 5.27: Punkteverteilung der *Callisto*-Projektgruppe hinsichtlich der verifizierten Bugs

Bei der Betrachtung der Abbildungen 5.27 bzw. 5.28 ergibt sich ein im Gegensatz zur Anzahl der geschlossenen Fälle ein differenzierteres Bild: Insbesondere in Abbildung 5.27 ist ein deutlicher Trend zu erkennen, nach dem zum *Europa*-Release eine deutliche Verschiebung hin zu einem größeren Anteil nicht-verifzierter Bugs stattfand; dieser Trend kehrt sich jedoch zum *Ganymede*-Release um und führte dort zu einer vergleichsweise hohen Verwendung des Zustand *VERIFIED*. Bei diesen Betrachtungen muß man sich jedoch stets vor Augen halten, daß die Referenzwerte sämtlich in

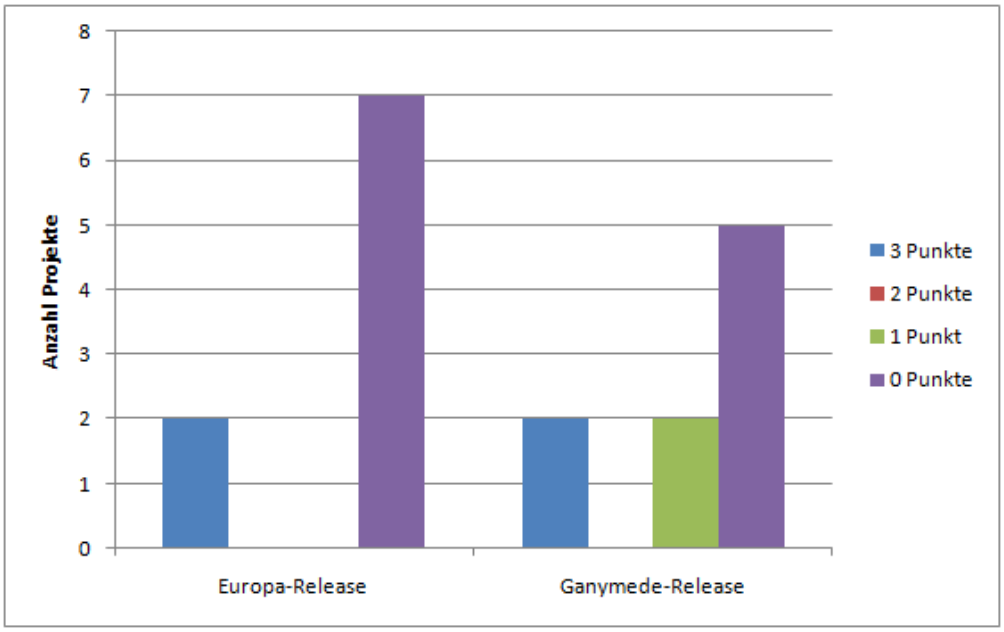

Abbildung 5.28: Punkteverteilung der *Europa*-Projektgruppe hinsichtlich der verifizierten Bugs

einem Bereich oberhalb von 85% angesiedelt sind und daß die Höchstpunktzahl bereits unmittelbar unter diesem Wert erreicht wird.

Zusammenhänge zwischen `CLOSED` und `VERIFIED`

Von Interesse sind auch die Zusammenhänge zwischen den Bewertungen der beiden zuvor ausgewerteten Metriken; intuitiv erwartet man, daß Projekte, die einen sorgfältigen Gebrauch des Zustands `CLOSED` machen, tendenziell auch stärkeren Gebrauch vom Zustand `VERIFIED` machen. Daß diese Annahme nur sehr eingeschränkt korrekt ist, zeigt die folgende Abbildung 5.29, die die Punktzahlen, die für jedes Projekt in den beiden im Vorangegangen bestimmten Qualitätseigenschaften ermittelt wurden, in Verhältnis zueinander setzt – hier am Beispiel des *Ganymede*-Releases. Zur besseren Erkennbarkeit wurden die Markierungspunkte von Projekten, die das gleiche Punktzahl-Verhältnis aufweisen, leicht gegeneinander versetzt.

Wäre die anfangs getroffene intuitive Erwartung korrekt, so müßte sich eine Punkthäufung entlang der ersten Winkelhalbierenden ergeben; stattdessen befinden sich die Punkte überwiegend an den Außenseiten des Diagramms. Der besonders interessierende rechte obere Quadrant, in dem Projekte mit hoher Anzahl geschlossener und ebenfalls hoher Anzahl verifizierter Bugs erscheinen, verzeichnet lediglich zwei Projekte (*TPTP* und *WebTools*). Häufungen treten hingegen im rechten unteren Quadranten (‘Geringe Anzahl geschlossener Bugs, von denen jedoch viele verifiziert werden’) und linken unteren Quadranten (‘Es wird weder Gebrauch vom Zustand `CLOSED` noch von `VERIFIED` gemacht’). In der Auswertung der Releases *Callisto* (1 Projekt, *Web-*

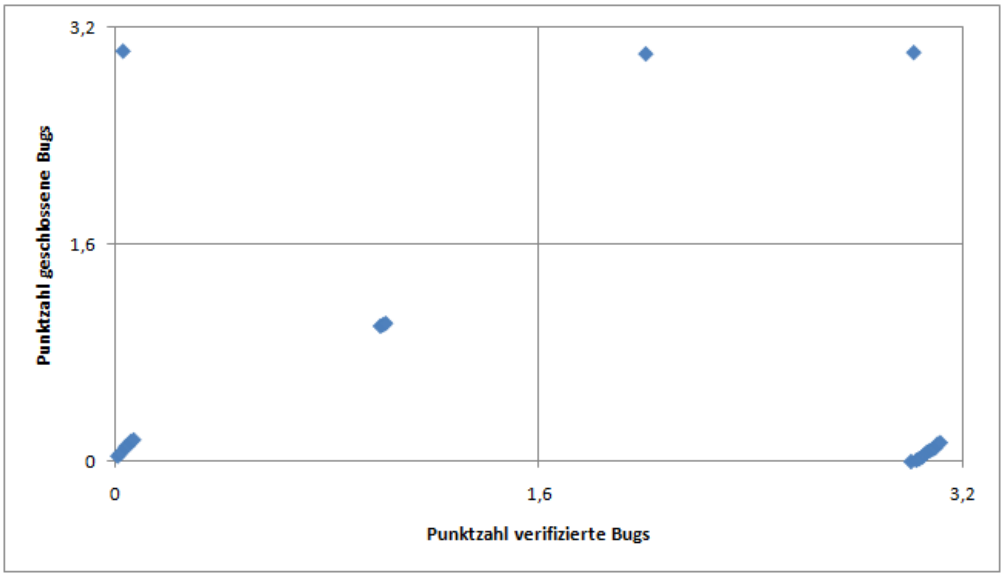

Abbildung 5.29: Zusammenhang zwischen der Punktzahl für verifizierte und geschlossene Bugs (*Ganymede*-Release)

Tools) und *Europa* (3 Projekte, *TPTP*, *WebTools* und *Visual Editor*) erreichen ähnlich wenige Projekte den rechten oberen Quadranten. Daraus läßt sich schließen, daß kein klarer Zusammenhang zwischen der Verwendung eines der Zustände `CLOSED` oder `VERIFIED` und der Verwendung des jeweils anderen besteht.

5.5.9 Gesondert zu betrachtende Projekte

Eine Besonderheit unter den betrachteten Projekten stellen die Projekte *Visual Editor* (*VE*) und *Incubator* dar, da sie nicht während des gesamten Auswertungszeitraums aktiv waren. Um die Ergebnisse der übrigen Projekte nicht zu verfälschen, wurden diese Projekte von der Auswertung der übrigen Projekte und der Ermittlung der Durchschnittswerte ausgenommen. Da sich aufgrund der Inaktivität dieser Projekte jedoch interessante Beobachtungen ergeben, soll an dieser Stelle eine kurze Betrachtung dieser Besonderheiten stattfinden.

Das Projekt *Visual Editor*, das sich mit der Entwicklung eines *WYSIWYG*-Editors für graphische Benutzeroberflächen in Java-Programmen befaßt hat, wurde im Jahr 2006 kurz nach der Veröffentlichung der Version 1.3 im *Callisto Simultaneous Release* weitestgehend eingestellt. Abbildung 5.30 zeigt den Verlauf der Qualität aufgeschlüsselt nach den Qualitätskriterien der 2. Ebene.

Erkennbar ist hier sehr deutlich, daß die Qualitätskriterien ‘Planung’ und ‘Bugzilla-Verwendung durch die User’ nach der Einstellung des Projekts ein Maximum annehmen, während die Kriterien ‘Abarbeitung’ und ‘Transparenz’ im Laufe der Zeit auf 0% sinken. Letzteres läßt sich darauf zurückführen, daß die Metriken des Kriteriums

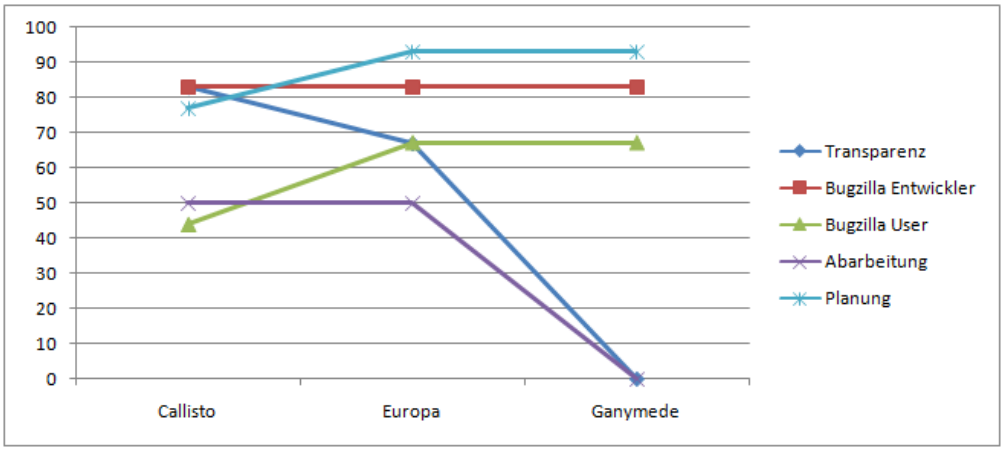

Abbildung 5.30: Verlauf der Qualitätskriterien des Projekts *Visual Editor*

'Transparenz' stark auf die Verwendung von Milestone-Angaben zurückgreifen; ein eingestelltes Projekt veröffentlicht keine Milestones mehr, so daß den wenigen noch eintreffenden Bugs auch kein Milestone zugewiesen werden kann. Das Kriterium 'Abarbeitung' bezieht sich in erster Linie auf die Wiedereröffnung von Bugs - wenn kaum neue Bugs hinzukommen, erhöht sich bei gleichbleibender Anzahl wiedereröffneter Bugs automatisch auch der Anteil der wiedereröffneten Bugs.

Umgekehrt verhält es sich bei den Kriterien 'Planung' und 'Bugzilla-Verwendung durch die User'. Die Metriken, die Einfluß auf das Kriterium 'Planung' ausüben beziehen sich überwiegend auf Änderungen, die an einem Fehlerbericht nach Beginn dessen Bearbeitung gemacht wurden. Wenn sich niemand mehr um die Bearbeitung der Bugs kümmert, können auch nicht viele Änderungen an bestehenden Bugs gemacht werden und es ergibt sich ein hoher Punktwert für diejenigen Metriken, die diese Änderungen vermessen. Eine ähnlicher Effekt liegt der Maximierung des Werts für das Qualitätsmerkmal 'Bugzilla-Verwendung durch die User' zugrunde – eine nicht zufriedenstellende Verwendung des Bug-Tracking-Werkzeugs durch die Benutzer läßt sich erst nach entsprechender Reaktion durch einen Entwickler vermessen. Wenn keine Reaktionen mehr stattfinden, erhält dieses Qualitätskriterium aufgrund fehlender Meßbarkeit einen optimalen Wert, der jedoch nicht zwangsläufig auch auf eine tatsächlich positive Verwendung des Werkzeugs durch die Anwender hindeutet.

Das zweite Projekt, das hinsichtlich seiner Kontinuität aus dem Rahmen fällt, ist *Incubator* aus der *Eclipse*-Classification. Bei diesem Projekt handelt es sich um eine Art 'Spielwiese' für neue Technologien, die später möglicherweise in eines der Projekte *Equinox*, *Platform* oder *PDE* übernommen werden:

> The Eclipse Project Incubator subproject gives the community around the Eclipse Project a forum and set of resources for innovation and investigation of new and alternative ideas. The project is essentially an umbrella for the incubation work associated with the main Eclipse sub-projects. [inc]

Dieses Projekt wurde erstmals in der Entwicklungsphase des *Ganymede*-Release in Bugzilla aktiv, so daß dieses Projekt ebenfalls einen Sonderfall darstellt.

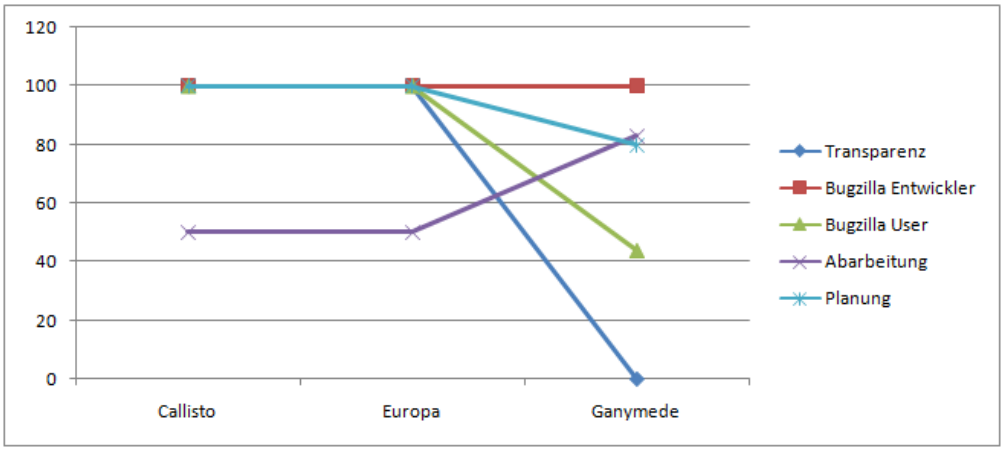

Abbildung 5.31: Verlauf der Qualitätskriterien des Projekts *Incubator*

Bei der Betrachtung des Qualitätsverlaufs fällt auf, daß sämtliche Qualitätskriterien mit Ausnahme von 'Abarbeitung' zu den Release-Zeitpunkten von *Callisto* und *Europa* den Maximalwert erreichen; dies ist darauf zurückzuführen, daß jegliche Metrik, die auf eine leere Bug-Datenbank angewandt wird, auch nur Null-Werte zurückliefert. Da es sich bei sämtlichen diesen Qualitätskriterien zugeordneten Metriken um Minimierungs-Metriken handelt, erfüllt der Wert 0 aufgrund des gewählten Auswertungsverfahrens grundsätzlich die höchsten Anforderungen und erhält bei der Berechnung der Qualitätsmerkmalen die Maximalpunktzahl von 3. Ein umgekehrter Effekt macht sich im Bereich des Kriteriums 'Abarbeitung' bemerkbar: Da die diesem Kriterium zugeordnete Metrik 'Durchschnittliche Dauer bis zur Wiedereröffnung' eine Maximierungs-Metrik ist, erhält das Projekt *Incubator* in diesem Qualitätsmerkmal aufgrund des Nullwerts die geringste mögliche Punktzahl von 0 Punkten. Dies führt dazu, daß das Qualitätskriterium 'Abarbeitung' eines leeren Projekts lediglich 50% der möglichen Höchstpunktzahl erreicht.

Zum Zeitpunkt des *Ganymede*-Release macht sich die beginnende Verwendung des Bugzilla-Werkzeugs deutlich in Form einer Veränderung der Qualitätswerte bemerkbar. Der besonders starke Abfall des Wertes für das Kriterium 'Transparenz' liegt darin begründet, daß das *Incubator*-Projekt keinen Gebrauch von der Möglichkeit, Milestones zu definieren, macht; dies ist bei einem Projekt, das sich als Spielwiese für experimentelle Technologien befaßt, jedoch auch nicht zwingend notwendig.

5.5.10 Quality Benchmark Levels

Wie bereits in Abschnitt 2.3 erwähnt besteht auch die Möglichkeit, Qualität in Form von *Quality Benchmark Levels* auszudrücken. Diese, in verschiedenen Bewertungssystemen [Car] [SSM06] angewandte Methodik läßt sich auch auf die Bewertung von

Prozessqualität anhand von Bug-Tracking-Datenbanken anwenden. Eine Auswertung der Eclipse-Projekte mit Hilfe von *QBLs* ist jedoch in zweierlei Hinsicht nicht sinnvoll.

Zunächst benötigt man zur Definition von *QBLs* eine zuverlässige Quelle, aus der sich die Grenzwerte und relevanten Metriken für die jeweiligen *QBL* ableiten lassen. Dies erfolgt idealerweise anhand einer Spezifikation, die für die betrachteten Projekte im Vorfeld erstellt wird. Diese Spezifikation könnte beispielsweise Anforderungen enthalten, in welcher Reihenfolge und in welchem Ausmaß bestimmte Qualitätsmerkmale zu erfüllen sind, um eine Qualitätsstufe zu erreichen. Aussagen könnten beispielsweise in der Form 'Zunächst sollen die Entwickler des Projekts dafür Sorge tragen, daß mindestens die Hälfte aller neu eingestellten Bugs innerhalb von 7 Tagen einem Bearbeiter zugewiesen wird. Wenn dieses Ziel erreicht ist, soll das Projektteam dafür sorgen, daß maximal 10% aller eingestellten Fehlerberichte als ungültig eingestuft werden' getroffen werden. Das Fehlen einer solchen Spezifikation im Umfeld des Eclipse-Prozesses macht es sehr schwer, eine sinnvolle und den tatsächlichen Anforderungen an die Eclipse-Projekte entsprechende Spezifikation für *QBLs* zu erstellen.

Als Alternative hierzu bestünde die Möglichkeit, anhand der bereits vorliegenden Auswertungsergebnisse eine Rangfolge unter den Metriken zu etablieren und diese Metriken gemäß ihrer Rangfolge zu den jeweiligen *QBLs* zuzuordnen. Diesem Ansatz liegt der Gedanke zugrunde, daß eine große Anzahl von Projekten ein niedriges *QBL* erfüllt, hingegen nur eine kleine Zahl von Projekten einem hohen *QBL* zuzuordnen ist. Bei der Anwendung von 20 Metriken wäre beispielsweise eine Aufteilung in 4 *QBLs* denkbar, wobei zur Erfüllung des ersten *QBL* eine hohe Punktzahl in den 5 Metriken, die die höchste Durchschnittspunktzahl über alle Projekte gemittelt aufweisen, erforderlich wäre; zur Erfüllung des vierten *QBL* wäre dann eine hohe Punktzahl in allen 20 Metriken erforderlich. Bezüglich der Anzahl der Projekte in den *QBLs* ergäbe sich somit eine pyramidenartige Struktur. Dieser Ansatz setzt jedoch voraus, daß sich eine eindeutige Reihenfolge bei der durchschnittlichen Punktzahl der einzelnen Metriken finden läßt. Liegen diese Durchschnittspunktzahlen zu dicht beieinander, so läßt sich keine eindeutige Reihenfolge der Metriken ermitteln. Darüber hinaus kann das Problem bestehen, daß jedes Projekt in genau einer dem jeweiligen *QBL* zugeordneten Metrik die Anforderungen nicht erfüllt und daß sich keine Kombination von Metriken finden läßt, bei der dieser Effekt nicht auftritt. Diese Einteilung in *QBLs* würde in direkter Folge dazu führen, daß sämtliche Projekte, unabhängig von ihrer tatsächlichen Qualität, lediglich die niedrigsten *QBLs* oder gar keine erfüllen würden, was einen Vergleich der Projekte praktisch zunichte macht. Abgesehen davon wäre ein willkürliches Wählen von *QBLs* problematisch, da die tatsächlichen Anforderungen des Prozesses möglicherweise verzerrt oder ganz außer Acht gelassen würden; die Zugehörigkeit eines Projekts zu einem bestimmten *QBL* wäre dann nicht mehr von diesen Anforderungen, sondern ausschließlich von der willkürlichen Wahl der *QBLs* abhängig, was eine weitgehend objektive Beurteilung der Prozesse unmöglich machen würde.

Beide angesprochenen Probleme treten in hohem Maße bei der Auswertung der Projekte des Eclipse-Ecosystems auf; aus diesem Grund wurde in dieser Arbeit darauf verzichtet, ein System von *Quality Benchmark Levels* zur Auswertung dieser Projekte zu entwickeln.

Kapitel 6

Abschließende Betrachtungen

6.1 Zusammenspiel mit anderen Auswertungsverfahren

Das im Rahmen dieser Arbeit entwickelte Auswertungsverfahren zur Bewertung der Prozessqualität von Open Source-Entwicklungsprojekten kann, wie in Kapitel 5 durchgeführt, als eigenständiges Bewertungsverfahren herangezogen werden, um eine begrenzte Menge von Qualitätseigenschaften eines Software-Prozesses aussagekräftig zu vermessen.

In der praktischen Anwendung derartiger Auswertungsverfahren wird jedoch meist eine weitaus umfassendere Sichtweise gefordert, die durch die isolierte Betrachtung eines Auswertungsverfahrens oder der ausschließlichen Auswertung einer Bugtracking-Datenbank selten erreicht werden kann. Zum Zwecke der umfassenden Evaluierung von Software-Prozessen, insbesondere im Bereich der Open Source-Entwicklung stehen bereits einige Ansätze zur Verfügung, die geeignet sind, ein umfassendes Bild sowohl von Qualität wie auch der Tauglichkeit für den kommerziellen Praxiseinsatz eines Software-Produktes zu erlangen. Einige dieser Verfahren werden von Ciolkowski und Soto in [CS08] exemplarisch vorgestellt. Als Motivation für die Entwicklung derartiger Auswertungsverfahren geben die Autoren an, daß sich bei der Evaluierung eines (freien wie auch kommmerziellen) Software-Produkts mehrere Fragen stellen, anhand derer das Risiko der Einführung neuer Softwareprodukte für den kommerziellen Einsatz eingeschätzt werden muß:

> The fact that OSS is not free of cost or risk is often used as a general argument against it. Yet, it should be taken into account that traditional, commercial-off-the-shelf (COTS) softare is similar in many ways. Software acquisition, both for OSS or COTS components, is concerned with questions such as the following:
>
> - Does the component provide the required functionality?
> - Is it ‘good enough’ for the purposes at hand?

- Will we be able to find support for it in five years from now?
- Are the licensing conditions compatible with our intended business model? [CS08]

Während sich einige dieser Fragestellungen mit Aspekten jenseits des eigentlichen Software-Entwicklungsprozesses wie rechtlichen Krtierien beschäftigen, können Beobachtungen zu Produktqualität und -nachhaltigkeit direkt aus den zur Verfügung stehenden Quellen wie Quellcode-Repositories oder auch Bug-Tracking-Datenbanken gewonnen werden. In beiden Fällen bietet Open Source Software meist einen eindeutigen Vorteil gegenüber klassischer kommerzieller Software:

> In this respect, OSS often offers an advantage, because the source code is always available and can be readily analyzed, whereas COTS software is seldom as transparent. [CS08]

Sofern eine Reihe auswertbarer Artefakte eines Projekts zur Verfügung steht, kann mit Hilfe automatisierter Auswertungsverfahren schnell ein objektiver Überblick über viele Aspekte der Qualität eines Produktes sowie dessen Eignung für einen bestimmten Einsatzzweck gewonnen werden. Im Falle von Quellcode-Analysen ist dies mit Hilfe gängiger Verfahren, wie sie beispielsweise in [SSM06] vorgestellt werden, möglich. Bei der Beurteilung der Eignung für einen gewerblichen Zweck ist jedoch meist nicht nur ausschließlich die Qualität des fertiggestellten Produkts ausschlaggebend, sondern auch die Nachhaltigkeit der Weiterentwicklung sowie die Anwender-Unterstützung (*Support*) durch den Hersteller oder die Anwendergemeinde. Dies trifft insbesondere auf sich stetig weiterentwickelnde Open Source-Softwareprodukte zu. Aufgrund dieser Erkenntnis wurden in den vergangenen Jahren einige Bewertungsverfahren speziell für derartige Software-Produkte entwickelt:

> As a reaction to the insight that not only code aspects need to be considered, assessment models for OSS projects have emerged to support potential OSS users. [CS08]

Beispiele für solche Auswertungsverfahren sind das *Open Source Business Readiness Rating* (*OpenBRR*) [BRRa], die *Method for Qualification and Selection of Open Source Software* (*QSOS*) [qso06] sowie das *Open Source Maturity Model* (*OSMM*) [Gol].

Diesen Bewertungsverfahren ist gemeinsam, daß die Bewertung ausschließlich aufgrund subjektiver Einschätzungen eines Bewerters zustande kommt. Sowohl *QSOS* als auch *OpenBRR* geben für die Bewertung eine Hilfestellung in Form eines Qualitätsmodells in Baumstruktur mit Hilfe dessen ein *Score* anhand einer für jede Qualitätseigenschaft zu ermittelnden Punktzahl berechnet werden kann. Die Punktzahlen der Qualitätseigenschaften selbst müssen jedoch insbesondere bei *QSOS* aufgrund subjektiver, manueller Beobachtungen anhand von Referenztabellen vergeben werden. Ziel

jeder Bewertung sollte hingegen die weitestgehende Elimination subjektiver Einschätzungen im Rahmen des Bewertungsprozesses sein, was sich in letzter Konsequenz jedoch, auch aufgrund der Tatsache, daß sich bestimmte Qualitätseigenschaften nicht objektiv mesen lassen, nicht vollständig vermeiden läßt:

> We try to be as objective as possible, but it is not possible to eliminate subjectivity in any assessment process. [BRRb]

Andererseits wird ein gewisses Maß an Subjektivität bei der Bewertung eines Produkts auch als Vorteil angesehen, da durch subjektive Einschätzungen den unterschiedlichen Anforderungen, die unterschiedliche Bewerter an ein Produkt stellen, Rechnung getragen wird:

> Allowing some subjectivity is in some way a strength of BRR. An assessment process is only correct if it is done according to the users' requirements. Because different people in different situations have different needs, it makes sense for BRR to produce different ratings for these different needs. [BRRb]

Negative Auswirkungen hat die Bewertung eines Projekts oder Produkts anhand subjektiver Einschätzungen jedoch auf die Reproduzierbarkeit der Bewertung. Führen zwei unterschiedliche Bewerter zeitnah eine derartige Bewertung eines Produkts aus, so können sie zu völlig unterschiedlichen Ergebnissen gelangen. Eine Möglichkeit zur Elimination zumindest eines gewissen Anteils an subjektiven Einschätzungen ließe sich durch Erweiterung der erwähnten Bewertungsverfahren um die automatisierte Auswertung zusätzlicher Artefakte wie Bug-Tracking-Datenbanken oder Change-Management-Systemen oder eine Integration dieser Auswertungen in die bestehenden Verfahren bewerkstelligen.

Einen stärker auf automatisiert zu ermittelnden Metriken basierenden Ansatz beschreiben Samoladas, Gousios, Spinellis und Stamelos im *SQO-OSS Quality Model* [S$^+$08]:

> The SQO-OSS model was constructed with a focus to automation, while the rest of the models require heavy user interference and lack automation of metrics collection. [S$^+$08]

Das in *SQO-OSS* eingesetzte Qualitätsmodell basiert ebenfalls auf einer Baumstruktur, das in der ersten Ebene aus den zwei Qualitätsmerkmalen *Code Quality* und *Community Quality* besteht. Während die Werte des Qualitätsmerkmals *Code Quality* durch Bewertungsverfahren wie sie in [SSM06] zu ermitteln sind, befassen sich die Metriken des Merkmals *Community Quality* ausschließlich mit Daten, die ohne spezielle, werkzeug-unterstützte Auswertungsverfahren direkt zu ermitteln sind. Dazu zählt unter anderem die Anzahl der Abonnenten der Projekt-Mailinglisten, die Nachrichtenhäufigkeit in diesen Listen, die Anzahl der verfügbaren Projektdokumente und das

Wachstum der Zahl aktiver Entwickler. Die in dieser Arbeit beschriebenen Auswertungsvefahren und -möglichkeiten könnten in Ergänzung hierzu für dieses Qualitätsmodell ein deutlich feineres Bild zeichnen als dies mit den derzeit vorgesehenen Auswertungen des *SQO-OSS*-Auswertungsverfahrens möglich ist.

Metriken auf Bug-Tracking-Datenbanken eignen sich insbesondere, um die für den kommerziellen Praxiseinsatz relevanten Aspekte zur Lebendigkeit, Geschwindigkeit und Nachhaltigkeit der Entwicklung sowie der Kommunikation zwischen Entwicklern und Anwendern sowie deren zeitlichen Verlauf objektiv einzuschätzen. Mit Hilfe der in Kapitel 2 vorgestellten Möglichkeiten zur Modellierung lassen sich durch entsprechende Wahl der Qualitätsmerkmale und Zusammenfassungsregeln nicht nur unterschiedliche Anforderungen und Sichtweisen auf ein Projekt modellieren; durch die zur Verfügung stehende Auswahl an Berechnungsverfahren für übergeordnete Qualitätsmerkmale lassen sich die gewonnenen Qualitätseigenschaften auch entsprechend den Anforderungen des jeweiligen Bewertungsverfahrens zusammenfassen und somit direkt in das Bewertungsverfahren integrieren.

6.2 Aufgetretene Schwierigkeiten

Bei der Entwicklung und Anwendung der in dieser Arbeit vorgestellten Methodiken haben sich einige Schwierigkeiten und Hindernisse gezeigt, auf die an dieser Stelle näher eingegangen werden soll.

Zunächst gestaltet sich die Entwicklung eines Qualitätsmodells sowie dessen Beschreibung mitunter als problematisch. Im Rahmen bidirektionaler Qualitätsmodelle besteht eine verwirrende Vielfalt von Begrifflichkeiten, die nicht klar voneinander abgegrenzt sind; insbesondere die Übertragung der Bezeichnungen aus dem Englischen ins Deutsche macht hier Schwierigkeiten – angemessene deutsche Äquivalente für die englischen Begriffe *quality characteristic*, *quality indicator* und *quality property* [SL08] zu finden gestaltet sich als äußerst schwierig. Die in dieser Arbeit verwendeten deutschen Begrifflichkeiten für *quality characteristic* (‘Qualitätskriterium’), *quality indicator* (‘Qualitätsmerkmal’, vgl. [SSM06]) und *quality property* (‘Qualitätseigenschaft’) liegen von ihrer umgangssprachlichen Bedeutung sehr dicht beieinander, so daß eine erhebliche Verwechslungsgefahr besteht. Darüber hinaus besteht nur eine sehr unscharfe Abgrenzung zwischen den Begriffen *quality characteristic* (‘Qualitätskriterium’) und *quality indicator* (‘Qualitätsmerkmal’), da Qualitätsmerkmale auch als Qualitätskriterien der untersten Ebene (mit höchster Granularität) aufgefaßt werden können.

Zum anderen bereitet die Auswahl des Auswertungs- und Vergleichsverfahrens Probleme: Ziel der Auswahl sollte sein, durch geeignete Wahl des Auswertungsverfahrens eine größtmögliche Aussagekraft für die Auswertung zu erhalten. Hierzu bietet sich insbesondere ein *QBL*-basiertes Verfahren an, da die Anforderungen an ein Projekt, das einem definierten Prozess genügen soll, durch die Festlegung der Qualitätsstufen optimal zur Geltung kommen. Jedoch läßt sich ein solches Auswertungsverfahren nur

dann sinnvoll anwenden, wenn eine eindeutige Priorisierung der verschiedenen Qualitätseigenschaften hinsichtlich ihrer Bedeutung für die Prozessqualität gegeben ist (siehe hierzu auch Abschnitt 2.3) oder, bei Fehlen einer solchen Priorisierung, eindeutig zu erkennen ist, welche Qualitätseigenschaften eine Verbesserung erfahren wenn sich die Gesamtqualität der Prozesseinhaltung eines Projekts verbessert.

Zusätzlich kann die Wahl des Verfahrens zur Bestimmung eines Punktwerts für die Qualitätseigenschaften anhand der Metrikwerte leicht zu Fehlinterpretationen des Ergebnisses führen. Auf diese Gefahr von Fehlinterpretationen wird bereits im Absatz 'Kombination mehrere Methoden' des Abschnitts 2.1.3 hingewiesen.

Eine weitere Schwierigkeit, die sich bei der Anwendung der beschriebenen Verfahrensweisen auf eine große Anzahl von Projekten oder bei einer großen Anzahl von Metriken ergibt, ist die schlechte Handhabbarkeit der entstehenden Datenmengen. Für jeden Auswertungszeitraum fällt pro Metrik und Projekt ein Wert an, so daß sich der Umfang der zu handhabenden Daten aus dem Produkt von Anzahl der Metriken, Anzahl der betrachteten Projekte und Anzahl der Auswertungszeiträume ergibt. Eine manuelle Verarbeitung dieser Werte beispielsweise mit Hilfe eines Tabellenkalkulationsprogramms ist dementsprechend anfällig für schwer aufzuspürende Fehler wie beispielsweise Rechenfehler. Ein Projekt, das diese Problematik zu lösen versucht, wird im folgenden Abschnitt 6.3 vorgestellt.

Speziell bei der Berechnung der Metrikwerte der Eclipse-Projekte mithilfe von *Bugzilla Metrics* ist es darüberhinaus aufgrund des enormen Umfangs der Bug-Datenbank (244252 Einträge insgesamt) zu technischen Problemen mit dem Auswertungswerkzeug gekommen; das Werkzeug war im Rahmen der für die Auswertung zur Verfügung stehenden Hardware-Umgebung nicht in der Lage, bei großen Projekten für solche Metriken, die die Zählung von Kommentaren zu Bugs beinhalten, Ergebnisse zuverlässig und innerhalb eines angemessenen Zeitrahmens zu ermitteln. Aus diesem Grund flossen die nur unvollständig zu ermittelnden Ergebnisse der Metriken 'Anzahl der Kommentare vor Übergang in den Zustand `ASSIGNED`' und 'Bugs ohne Reaktion innerhalb von 2 Tagen' nicht in die Auswertung ein.

Weitere Schwierigkeiten, die sich insbesondere durch fehlende oder sehr allgemein gehaltene Prozessbeschreibungen und -dokumentationen ergeben haben, wurden in den jeweiligen Kapiteln bereits beschrieben.

6.3 Ausblick

Ein Hindernis, das die Einsatzfähigkeit des vorgestellten Bewertungsverfahrens zur Auswertung von Daten aus Bug-Tracking-Systemen im Praxiseinsatz derzeit noch einschränkt ist die schlechte Handhabbarkeit der bei der Auswertung anfallenden Datenmengen. Bei gleichzeitiger Auswertung von 20 Projekten anhand von 20 Metriken fallen pro Auswertungszeitraum 400 Metrik-Ergebnisse an, die manuell oder semiautomatisiert in die jeweiligen Werte der Qualitätseigenschaften und -merkmale um-

gerechnet werden müssen. Hierbei können Fehlerquellen, die die Auswertungsergebnisse möglicherweise erheblich verfälschen, nicht vollständig eliminiert werden. Abhilfe hierzu soll der sich derzeit bei Martin Jansen in Entwicklung befindliche Qualitätsmodell-Editor mit integriertem Auswertungswerkzeug [Jan09], basierend auf dem *Eclipse*-Framework, schaffen. Mit diesem Werkzeug wird es möglich sein, graphisch Qualitätsmodelle in Baumstruktur zu modellieren, die Berechnungsverfahren für Qualitätseigenschaften und -merkmale festzulegen und diesen Metriken zuzuordnen. Mit Hilfe des integrierten Auswertungswerkzeugs können dann die Qualitätswerte für vollständige Qualitätsmodelle vollständig automatisiert ermittelt werden; manuelle Eingriffe sind nur noch bei Änderungen am Qualitätsmodell oder den verbundenen Metriken erforderlich. Durch den Einsatz dieses Werkzeugs kann die Auswertung von Bug-Tracking-Datenbanken nicht nur erheblich beschleunigt werden; auch durch die manuelle Berechnung von Metrikwerten entstehenden Fehlerquellen können so eliminiert werden.

Ein weiteres Hindernis, das dem verbreiteten Einsatz des in dieser Arbeit vorgestellten Auswertungsverfahrens entgegensteht ist die Beschränkung des Auswertungswerkzeugs *BugzillaMetrics* auf das Bug-Tracking-Werkzeug *Bugzilla*. Obwohl der Einsatz dieses Bug Trackers weit verbreitet ist, setzen viele Anwender inzwischen andere Werkzeuge zum *Change Request Management* ein. Ein prominentes Beispiel hierzu ist *Mantis* [mbt], ein auf *PHP* sowie *MySQL*, *Microsoft SQL* oder *PostreSQL* basierendes Bug-Tracking-Werkzeug. Die Verbreitung der automatisierten Auswertung von Bug-Tracking-Datenbanken ist selbstverständlich auch von der Verfügbarkeit entsprechender Auswertungs-Werkzeuge für das jeweils eingesetzte Bug-Tracking-Werkzeug abhängig. Aus diesem Grund wurde am Lehr- und Forschungsgebiet Informatik 3 der RWTH Aachen eine Portierung des *BugzillaMetrics*-Werkzeugs auf das oben erwähnte Bug-Tracking-Werkzeug *Mantis* vorgenommen.

In Abschnitt 6.1 wurde darber hinaus erwähnt, daß sich auch aus der Auswertung weiterer Artefakte, die bei der Entwicklung eines Software-Produkts entstehen, ebenfalls Aussagen über die Qualität des Entwicklungsprozesses treffen lassen. Hierbei spielen vor allem Quellcode-Repositories (auch als *Versionsmanagement-Systeme* bekannt) eine große Rolle. Zudem besteht häufig ein starker Zusammenhang zwischen den in Bug-Tracking-Systemen vorgehaltenen Informationen und den Änderungen an den Software-Repositories; in vielen Entwicklergemeinden ist es üblich, Änderungen, die zur Umsetzung einer gewünschten Funktionalität oder eines Fehler-Reports aus dem Bug-Tracking-Werkzeug geführt haben, im Quellcode-Repository mit der Nummer des Bug-Tracking-Eintrags zu kennzeichnen. Dieser Tatsache trägt die von Christoph Lischkowitz [Lis08] umgesetzte Erweiterung von *BugzillaMetrics* rechnung, die eine Anbindung des Auswertungswerkzeugs an ein Versionsmanagement-System (im konkreten Fall *Subversion* [svn]) realisiert. Mit dieser Erweiterung werden sich nicht nur (beispielsweise mit bereits verfügbaren Werkzeugen wie *StatCVS* [stc]) isolierte Messungen an den Daten des Versionsmanagement-Systems vornehmen lassen, sondern es besteht die Möglichkeit, die Zusammenhänge zwischen den Daten des Bug-Tracking-Werkzeugs und des Versionsmanagement-Systems zu erfassen.

6.4 Fazit

Im Rahmen dieser Arbeit wurde ein Verfahren zur Bewertung der Prozessqualität von Open Source-Entwicklungsprojekten, basierend auf den Daten in einem Change-Request-Management-System entwickelt und auf einen realen Prozess angewandt. Hierzu wurde zunächst eine Möglichkeit der Modellierung von Prozessqualität vorgestellt und es wurden verschiedene Berechnungsverfahren zur Ermittlung vergleichbarer Qualitätswerte anhand dieses Qualitätsmodells eingeführt. Unter Berücksichtigung der für Open Source-Prozesse geltenden Rahmenbedingungen wurden sodann ein Qualitätsmodell für die Bewertung der Prozessqualitäten der Eclipse-Projekte sowie die dazugehörigen Metriken entwickelt. Im Anschluß daran erfolgte die Auswertung dieser Projekte mit den vorgestellten Auswertungsverfahren.

Im Rahmen dieser Arbeit hat sich gezeigt, daß die betrachteten Projekte des Eclipse-Ecosystems eine gute Stabilität aufweisen, was die verbreiteten Vorurteile gegenüber Projekten mit offener Entwicklung im Gegensatz zu kommerziellen Softwaresystemen zumindest für diese Gruppe von Projekten widerlegt. Darüber hinaus konnte festgestellt werden, daß die zeitliche Entwicklung der Qualität der Eclipse-Projekte nicht von einzelnen Merkmalen der Prozesseinhaltung abhängt, sondern daß ein Zusammenhang zwischen den einzelnen Qualitätsmerkmalen feststellbar ist.

Die in dieser Arbeit vorgestellten Methoden zur Auswertung offener Entwicklungsprozesse anhand von Daten aus einem Bug-Tracking-System können, insbesondere in Verbindung mit den in Abschnitt 6.1 erwähnten Bewertungsverfahren für die Prozessqualität von Open Source-Projekten, als Grundlage zur Entwicklung standardisierter Auswertungsverfahren für derartige Projekte dienen. Um einen zuverlässigen und reproduzierbaren Einsatz auf eine größere Anzahl von Open Source-Projekten zu ermöglichen, müssen jedoch zunächst noch anhand einer umfangreichen Datenbasis ermittelte Richtwerte für die einzelnen Qualitätseigenschaften festgelegt werden. In [SSM06] werden beispielsweise die Quartile einer großen Anzahl von Auswertungen als Referenzwerte zugrunde gelegt.

Die gute Verfügbarkeit von Verfahren zur Bewertung der Prozessqualität offener Entwicklungsprojekte, im Zusammenhang mit der meist problemlosen Beschaffung von Prozessdaten über derartige Projekte, ergibt sich eine hervorragende Möglichkeit, diese Projekte (beispielsweise vor einem kommerziellen Einsatz der entwickelten Software in Unternehmen) ausgiebig zu prüfen und die Entscheidung für oder gegen den Einsatz eines betrachteten Produkts durch derartige Auswertungsverfahren zu unterstützen. Hieraus ergibt sich ein klarer Vorteil von Open Source-Produkten gegenüber kommerzieller Closed Source-Software, da zur Bewertung letzterer meist nur subjektive Einschätzungen und Angaben des Herstellers oder Bewertungen der Fachpresse vorliegen, die jedoch in vielen Fällen nicht die Anforderungen und speziellen Sichtweisen derjenigen Personen wiedergibt, die über den Einsatz des Produkts zu entscheiden hat. Durch die in Abschnitt 6.3 erwähnten Werkzeuge sowie Ergänzungen ist darüber hinaus zu erwarten, daß in Kürze eine umfassende Sammlung von Werkzeugen

zur automatisierten Auswertung der verschiedensten Prozess-Artefakte für eine breite Anwenderschicht zur Verfügung steht.

Die Bewertung der Prozessqualität von Open Source-Projekten anhand von Software-Prozessdaten stellt somit eine gut verfügbare, kostengünstige und leicht anzuwendende Möglichkeit dar, in kurzer Zeit zu Aussagen über die Qualität eines Softwareprojekts sowie dessen Eignung zum kommerziellen Unternehmenseinsatz zu gelangen. Dabei muß jedoch stets berücksichtigt werden, daß sich durch jegliche Auswertungsverfahren lediglich ein Teil aller relevanten Aspekte eines Prozesses erfassen, abbilden und auswerten läßt. Außerdem sind bei automatischen Auswertungsverfahren stets die Besonderheiten des jeweils betrachteten Projekts, dessen Prozesses und nicht zuletzt des eingesetzten Werkzeugs, anhand dessen die Prozessdaten gesammelt werden, zu berücksichtigen; so sind Metriken, die beispielsweise auf eine *Bugzilla*-Datenbank anwendbar sind, nicht zwangsläufig in gleicher Form für *Mantis*-Datenbanken geeignet. Auch spielen unterschiedliche Prozessbedürfnisse und -anforderungen bei der Wahl des Auswertungsverfahrens eine Rolle. Vor dem Einsatz eines derartigen Auswertungsverfahrens muß daher immer geprüft werden, ob sich das Auswertungsverfahren zur Abbildung der invidiuellen Sichtweise und den Anforderungen des Beobachters dahingehend eignet, daß aus der Auswertung aussagekräftige Ergebnisse hervorgehen.

Literaturverzeichnis

[BP02] BAUER, ANDREAS und MARKUS PIZKA: *The Contribution of Free Software to Software Evolution*. In: *Proceedings of the Sixth International Workshop on Principles of Software Evolution*. IWPSE03, 2002.

[BRRa] BRR: *Business Readiness Rating: A proposed Open Standard to Facilitate Assessment and Adoption of open Source Software*. http://www.openbrr.org.

[BRRb] BRR: *Business Readiness Rating: Frequently Asked Questions*. http://www.openbrr.org/wiki/index.php/FAQ.

[BT94] BÜNING, HERBERT und GÖTZ TRENKLER: *Nichtparametrische Statistische Methoden*, Kapitel 2.8, Seite 39. de Gruyter, 2 Auflage, September 1994.

[Cal] *Eclipse Callisto Projects*. http://www.eclipse.org/callisto/callistoprojects.php.

[Car] CARNEGIE MELLON UNIVERSITY: *Capability Maturity Model Integration*. http://www.sei.cmu.edu/cmmi/.

[Cer05] CERNOSEK, GARY: *A brief history of Eclipse*. http://www.ibm.com/developerworks/rational/library/nov05/cernosek/, 2005.

[CS08] CIOLKOWSKI, MARCUS und MARTÍN SOTO: *Towards a Comprehensive Approach for Assessing Open Source Projects*. In: DUMKE, RAINER et al. (Herausgeber): *Software Process and Product Measurement*, Seiten 316–330. Springer Verlag, 2008.

[Das] *Eclipse Dash*. http://www.eclipse.org/dash/.

[EBU] *Eclipse Bugzilla Use*. http://wiki.eclipse.org/Development_Resources/HOWTO/Bugzilla_Use.

[ECL] *Eclipse*. http://www.eclipse.org.

[ED] EBERT, CHRISTOF und REINER DUMKE: *Software Measurement: Establish - Extract - Evaluate - Execute*.

[EDP08] *The Eclipse Development Process, Revision 2.4*. http://www.eclipse.org/projects/dev_process/development_process.php, August 2008.

[Eur] *Europa Simultaneous Release*. http://wiki.eclipse.org/Europa_Simultaneous_Release.

[G+] GRAMMEL, LARS et al.: *Bugzilla Metrics*. http://www.bugzillametrics.org.

[Gan] *Ganymede Simultaneous Release*. http://wiki.eclipse.org/Ganymede_Simultaneous_Release.

[GLS07] GRAMMEL, LARS, HORST LICHTER und HOLGER SCHACKMANN: *Bugzilla Metrics - An adaptable tool approach for evaluating metric specifications on change requests*. In: *Proceedings of 9th International Workshop on Principles of Software Evolution*, IWPSE, 2007.

[Gol] GOLDEN, B.: *Open Source Maturity Model*. http://www.navicasoft.com/pages/osmm.htm.

[Gra07] GRAMMEL, LARS: *Development of a Tool for the Evaluation of Change Requests*. Diplomarbeit, RWTH Aachen University, Februar 2007.

[GW05] GAMMA, ERICH und JOHN WIEGAND: *The Eclipse Way - Processes that Adapt*. EclipseCon, 2005.

[inc] *Eclipse Incubator Project*. http://www.eclipse.org/eclipse/incubator/.

[Jan09] JANSEN, MARTIN: *Entwicklung eines generischen Qualitätsmodell-Editors und Auswertungswerkzeugs*. Diplomarbeit, RWTH Aachen University, Januar 2009.

[KCM07] KAGDI, HUZEFA, MICHAEL L. COLLARD und JONATHAN I. MALETIC: *A survey and taxonomy of approaches for mining software repositories in the context of software evolution*. JSME, (19):77–131, 2007.

[Law07] LAWTON, MATT: *Eclipse Community Survey*, Oktober 2007.

[Lis08] LISCHKOWITZ, CHRISTOPH: *Erweiterung von BugzillaMetrics zur Auswertung von Daten aus Versionskontrollsystemen*. Diplomarbeit, RWTH Aachen University, Dezember 2008.

[mbt] *Mantis Bug Tracker*. http://www.mantisbt.org.

[Moza] MOZILLA ORGANIZATION, THE: *About Bugzilla*. http://www.bugzilla.org/about.

[Mozb] MOZILLA ORGANIZATION, THE: *Bugzilla*. http://www.bugzilla.org.

[qso06] *Method for Qualification and Selection of Open Source Software*. http://qsos.org, October 2006.

[S+08] SAMOLADAS, IOANNIS et al.: *The SQO-OSS Quality Model: Measurement Based Open Source Software Evaluation*. In: *Open Source Development, Communities and Quality*, Band 275 der Reihe *IFIP International Federation for Information Processing*, Seiten 237–248. Springer Boston, Juli 2008.

[SL08] SCHACKMANN, HOLGER und HORST LICHTER: *Comparison of Process Quality Characteristics based on Change Request Data*. In: DUMKE, RAINER et al. (Herausgeber): *Software Process and Product Measurement*. Springer Verlag, 2008.

[SSM06] SIMON, FRANK, OLAF SENG und THOMAS MOHAUPT: *Code Quality Management*. dpunkt Verlag, 2006.

[stc] *StatCVS*.
http://statcvs.sourceforge.net.

[svn] *Subversion*.
http://subversion.tigris.org.