Metrik-basierte Auswertung von Software-Entwicklungsarchiven zur Prozessbewertung

Von der Fakultät für Mathematik, Informatik und Naturwissenschaften der RWTH Aachen University zur Erlangung des akademischen Grades eines Doktors der Naturwissenschaften genehmigte Dissertation

vorgelegt von

Dipl.-Inform. Holger Schackmann
aus Bitburg

Berichter: Prof. Dr. rer. nat. Horst Lichter
Prof. Dr. rer. nat. Kurt Schneider

Tag der mündlichen Prüfung: 6. Mai 2010

D 82 (Diss. RWTH Aachen University, 2010)

Aachener Informatik-Berichte, Software Engineering

herausgegeben von
Prof. Dr. rer. nat. Bernhard Rumpe
Software Engineering
RWTH Aachen University

Band 7

Holger Schackmann

Metrik-basierte Auswertung von Software-Entwicklungsarchiven zur Prozessbewertung

Shaker Verlag
Aachen 2010

Bibliografische Information der Deutschen Nationalbibliothek
Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über http://dnb.d-nb.de abrufbar.

Zugl.: D 82 (Diss. RWTH Aachen University, 2010)

Das Bildmotiv/Umschlag basiert auf einer Vorlage von Joaquim Alves Gaspar.
Quelle: http://commons.wikimedia.org/wiki/File:Using_the_caliper_new.gif

Printed in Germany.

ISBN 978-3-8322-9405-2
ISSN 1869-9170

Shaker Verlag GmbH • Postfach 101818 • 52018 Aachen
Telefon: 02407 / 95 96 - 0 • Telefax: 02407 / 95 96 - 9
Internet: www.shaker.de • E-Mail: info@shaker.de

Sonja gewidmet

Kurzdarstellung

Die Entwicklung und Wartung von komplexen Softwareprodukten erfordert Transparenz bei Prozessen und Kosten. Messungen im Entwicklungsprozess sind ein Mittel um diese Transparenz zu schaffen. Sie ermöglichen die Bewertung und kontinuierliche Überwachung von Prozessen. Dem versprochenen Nutzen stehen die Kosten der Messungen gegenüber. Systematisches Messen im Prozess verlangt eine an das Unternehmen angepasste Infrastruktur zur Erfassung und Aufbereitung der Messergebnisse. Der dazu nötige Aufwand stellt gerade für kleine und mittelgroße Organisationen eine Hürde dar.

Viele Daten aus Entwicklungsprozessen werden heute schon routinemäßig erfasst, beispielsweise in Issue-Tracking-Systemen und Konfigurationsmanagement-Systemen. Die in solchen Software-Entwicklungsarchiven ohnehin gesammelten Daten spiegeln den Ablauf von Teilen der Entwicklungsprozesse wieder. Ihre Auswertung stellt also potentiell eine kostengünstige Alternative gegenüber einer manuellen Erfassung von Statusinformationen dar. In der industriellen Praxis wird diese Datenbasis allerdings nur unzureichend für Messungen zum Prozess genutzt. Gründe dafür sind zum einen fehlende methodische Unterstützung für die Entwicklung von Metriken, zum anderen unflexible Werkzeugunterstützung.

Die in dieser Arbeit entwickelten Methoden und Werkzeuge zielen darauf ab, die in Software-Entwicklungsarchiven gesammelten Daten besser für die Bewertung von Prozessen nutzbar zu machen. Kern der entwickelten Lösung ist die deklarative Sprache ITMS zur Spezifikation von Metriken auf Issue-Tracking-Systemen. Diese Sprache ermöglicht eine kompakte und präzise Beschreibung von Metriken auf einem hohen Abstraktionsniveau. Die vorgestellte Referenzimplementierung der Sprache kann flexibel an unterschiedliche Software-Entwicklungsarchive angebunden werden. Weiterhin sind die in ITMS spezifizierten Metriken leicht anpassbar. Dies ermöglicht ein iteratives Verfahren zur Entwicklung und Validierung von Metriken, welches in dieser Arbeit beschrieben wird.

Um eine systematische Interpretation von Messergebnissen zu erleichern, wird ein Metamodell für Qualitätsmodelle vorgestellt. Ein solches Qualitätsmodell stellt den Bezug zwischen subjektiven Qualitätsmerkmalen und den Messungen dar, und ist operativ zur Prozessbewertung einsetzbar. Diese Konzepte wurden in einem Qualitätsmodell-Editor und Auswertungs-Werkzeug umgesetzt. Das Auswertungswerkzeug unterstützt die Klassifikation von Messergebnissen auf Basis der Werteverteilung empirischer Vergleichsdaten. Dies ermöglicht eine pragmatische und realistische Einordnung der Messergebnisse.

Anwendbarkeit und Skalierbarkeit der entwickelten Methoden und Werkzeuge wird am Beispiel der Prozessbewertung in der industriellen Softwareentwicklung als auch bei der Bewertung von Prozessen im Open Source Bereich demonstriert.

Le vrai voyage ce n'est pas de chercher des nouveaux paysages mais un nouveau regard.

(Marcel Proust, 1871-1922)

Danksagung

Softwareentwicklung als Expedition ist eine gebräuchliche Metapher. Dieses Bildnis lässt sich auch auf eine Forschungsarbeit übertragen. Genauso wie eine Expedition verfolgt eine Forschungsarbeit ein Ziel. Sie verläuft in verschiedenen Etappen, und gelegentlich ist es notwendig bei Problemen zu verweilen. Die Reiseroute lässt sich nicht vollständig planen, da ein neues Territorium betreten wird. Die Planung muss an die vor Ort anzutreffenden Verhältnisse angepasst werden. An dieser Stelle möchte ich allen danken, die zum Gelingen dieser Expedition beigetragen haben.

Mein herzlicher Dank gilt Prof. Horst Lichter für die konstruktive inhaltliche Begleitung und auch dafür, dass er schon im Studium das richtige Training für eine solche Expedition vermittelt hat. Prof. Kurt Schneider danke ich für die bereitwillige Übernahme des Zweitgutachtens.

Viele Ideen aus dieser Arbeit sowie deren praktische Evaluierung wurden durch die langfristige Zusammenarbeit mit der KISTERS AG möglich. Mein Dank gilt Dr. Heinz-Josef Schlebusch und Dr. Juliane Ruhland, stellvertretend für alle Mitarbeiter der KISTERS AG, mit denen ich zusammenarbeiten konnte. Klaus Kisters danke ich für die Unterstützung des Kooperationsprojekts.

In einer Reihe von Diplom- und Masterarbeiten wurden nicht nur wichtige Beiträge zu verschiedenen Forschungsvorhaben geleistet. Bei der Begleitung einer solchen Abschlussarbeit findet auch immer ein voneinander Lernen statt. Mein Dank gilt Georg Richlofsky, Henning Schäfer, Martin Jansen, Christoph Lischkowitz, Netiya Ounjai, Sofia Adamanova, Andreas Hermanns und nicht zuletzt Lars Grammel, der die grundlegende Arbeit zu ITMS geleistet und den Anstoß für das Open Source Projekt *BugzillaMetrics* gegeben hat. Eine besondere Freude ist es, dass mich Veit Hoffmann, Malek Obaid, Andreas Ganser und Matthias Vianden auch als Kollegen am Lehr- und Forschungsgebiet Informatik 3 weiter begleiteten. Desgleichen möchte ich mich bei Bärbel Kronewetter, Dr. Thomas von der Maßen, Thomas Weiler und Dr. Alexander Nyßen für die vielfache Unterstützung und die freundschaftliche Atmosphäre bedanken.

Ausdrücklich danke ich meinen Eltern, sowie meinen Geschwistern Heike, Frank und Jens, die in vieler Hinsicht zu der nötigen Ausrüstung für diese Expedition beigetragen haben. Besonders danke ich meiner Freundin Sonja für ihre Liebe und ihre Unterstützung.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1. Einleitung

Life was simple before World War II.
After that, we had systems.

(Grace Hopper, 1906-1992)

Inhaltsangabe

In den traditionellen Ingenieursdisziplinen ist heute Transparenz bei Prozessen und Kosten unverzichtbar, damit ein Unternehmen langfristig im Wettbewerb besteht [Sch05]. Messungen im Prozess sind ein Mittel, um diese Transparenz zu schaffen.

Auch im Software Engineering unterstützen Messungen die Planung, Überwachung und Bewertung von Prozessen [BMB02]. Dies ist insbesondere bei hoher Komplexität von Prozessen und Produkten erforderlich, beispielsweise bei der Software-Produktlinienentwicklung [CN02] oder im Kontext von Multiprojektmanagement [Küt06].

Dem Nutzen von Messungen stehen die Kosten eines Messprogramms gegenüber. Ebert und Dumke bewerten die Kosten für den Aufbau und Betrieb eines Messprogramm mit ca. 0,5 bis 2% der Kosten des gesamten Entwicklungs- oder IT-Budgets [ED07]. Nach Jones können die Kosten 4 bis 6% des Entwicklungsbudgets betragen [Jon08]. In einer breit angelegten Untersuchung der NASA wurde zwischen den Kosten der Datenerfassung, den Kosten für die Infrastruktur und den Kosten für die Analyse und Aufbereitung der Ergebnisse unterschieden (vgl. Abbildung 1.1) [NAS95]. Es zeigt sich, dass die Kosten der Datenerfassung deutlich geringer sind als die Kosten für die technische Infrastruktur und die Analyse und Aufbereitung der Ergebnisse. Dabei ist zu berücksichtigen, dass unter den letzten Punkt auch Kosten für die Schulung der Mitarbeiter fallen.

Der prozentuale Anteil der Kosten des Messprogramms am gesamten Entwicklungsbudget wird von sehr vielen Faktoren beeinflusst und variiert deswegen stark. Es ist allerdings erkennbar, dass die prozentualen Kosten in einer mittelgroßen Organisation (100-500 Mitarbeiter) deutlich höher liegen als in einem Großunternehmen. Der Grund dafür ist, dass die Gemeinkosten für die Etablierung und den Betrieb eines Messprogramms in einem Großunternehmen auf mehr Mitarbeiter umgelegt werden können. Deshalb kann diese Aussage auch heute als gültig angesehen werden.

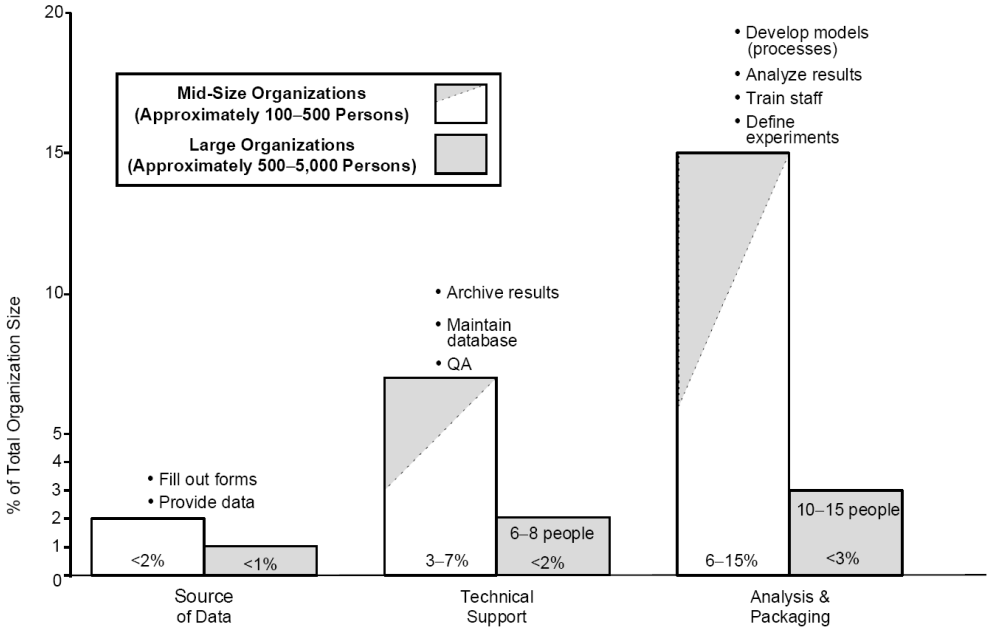


Abbildung 1.1.: Kosten von Software-Messungen [NAS95]

Der Aufbau eines Messprogramms und die nötige Entwicklung und Pflege eigens angepasster Werkzeuge ist also eher für Großunternehmen realistisch. Für mittlere Unternehmen erscheint der Aufwand dafür als zu hoch. Daher stellt sich die Frage, wie Software-Messungen auch für solche Organisationen besser nutzbar gemacht werden können.

Harjumaa et al. identifizierten im Rahmen einer Literaturrecherche Erfolgsfaktoren bei der Einführung von Messungen [HMO08]. Dazu gehören unter anderem:

- Die Messungen sind auf die Ziele der Organisation abgestimmt [Pfl93, BDR96, HF97, HG98, NvV01].
- Bereits verfügbare Daten werden in die Auswertung einbezogen [Pfl93, HF97].
- Daten werden automatisiert erfasst [Pfl93, HF97, KKJC06].
- Die Messungen verursachen wenig zusätzlichen Aufwand [Pfl93, KKJC06].
- Anwender können auf die Richtigkeit und Vollständigkeit der Daten vertrauen [IM03, Dek99].

In Übereinstimmung mit diesen Erfolgsfaktoren schlagen Cook et al. die Auswertung von Entwicklungsarchiven vor, in denen routinemäßig Informationen über den Entwicklungsprozess erfasst werden [CVW98]. Zu solchen Software-Entwicklungsarchiven zählen beispielsweise Konfigurationsmanagement-Systeme und Issue-Tracking-Systeme.

Die in Software-Entwicklungsarchiven gesammelten Daten werden in der industriellen Praxis allerdings nur unzureichend für Metrikauswertungen verwendet. Gängige Werkzeuge sind zu unflexibel bezüglich der möglichen Auswertungen. Somit werden typischerweise eigene Werkzeuge beziehungsweise Auswertungsskripte entwickelt [KA08]. Die Definition, Implementierung und Validierung von eigenen Metriken ist somit arbeitsaufwändig. Zudem gibt es nur unzureichende methodische Unterstützung für die Entwicklung von Metriken und deren präzise Beschreibung [HALS08].

1.1. Beitrag dieser Arbeit

Die in dieser Arbeit entwickelten Methoden, Sprachen und Werkzeuge zielen darauf ab, die in Software-Entwicklungsarchiven erfassten Daten besser für die Bewertung von Prozessen nutzbar zu machen. Dazu wird eine deklarative Sprache eingeführt, die es ermöglicht, Metriken auf Software-Entwicklungsarchiven auf einem hohen Abstraktionsniveau zu beschreiben. Dies erleichtert die iterative Entwicklung und Validierung von Metriken.

Weiterhin wird ein Ansatz zur Definition von organisationsspezifischen Qualitätsmodellen vorgestellt. Solche Qualitätsmodelle dienen einer systematischen Bewertung der Prozessqualität. Die entwickelten Methoden werden durch eine Werkzeuginfrastruktur unterstützt, die unabhängig von den auszuwertenden Software-Entwicklungsarchiven ist.

Diese Arbeit erbringt für das Gebiet Software Engineering die folgenden Beiträge:

- Eine deklarative Sprache zur präzisen Beschreibung von Metriken auf Software-Entwicklungsarchiven
- Ein bewährtes Vorgehen zur Entwicklung und Validierung von Metriken auf Software-Entwicklungsarchiven
- Ein Ansatz zur Erstellung von benutzerdefinierten metrik-basierten Qualitätsmodellen

Die entwickelte Sprache sowie die zugehörigen Konzepte und Werkzeuge wurden in mehreren Fallstudien im Open Source Bereich und im industriellen Kontext evaluiert.

1.2. Überblick über die Arbeit

Kapitel 2 führt relevante Grundbegriffe ein und stellt Forschungsgebiete im Umfeld dieser Arbeit vor. In Kapitel 3 werden die Ziele dieser Arbeit anhand von drei Fragestellungen charakterisiert.

Die deklarative Sprache ITMS zur Definition von Metriken auf Software-Entwicklungsarchiven wird in Kapitel 4 eingeführt. Weiterhin wird eine denotationelle Semantik für ITMS beschrieben. Ein bewährtes Vorgehen zur Entwicklung und Validierung von Metriken in ITMS wird in Kapitel 5 vorgestellt. Kapitel 6 beschreibt ein Metamodell für benutzerdefinierte Qualitätsmodelle. Die Anforderungen an die entsprechende Werkzeug-Suite QMetric sowie deren Architektur wird in Kapitel 7 dargestellt.

Kapitel 8 ist der Bewertung der entwickelten Konzepte und Werkzeuge gewidmet. Die Arbeit schließt mit einer Zusammenfassung und einem Ausblick in Kapitel 9.

2. Grundlagen

Denn eben wo Begriffe fehlen,
Da stellt ein Wort zur rechten Zeit sich ein.

(Johann Wolfgang von Goethe, 1749-1832
Faust I, Vers 1995 f.)

Inhaltsangabe

2.1. Grundbegriffe

In diesem Abschnitt wird eine Reihe von Grundbegriffen eingeführt, die in dieser Arbeit verwendet werden. Nach Möglichkeit lehnen sich die Begriffe an relevante Standards der Organisationen ISO oder IEEE an.

2.1.1. Metriken und Messung

Messung im Software Engineering ist nur bedingt mit Messung in der Physik oder im Ingenieurswesen vergleichbar [Zus97]. Die im Software Engineering zentrale Rolle menschlicher Expertentätigkeit rückt den Einsatz von Messungen näher an die Sozialwissenschaften [BMB02]. Messungen müssen daher als ein Teil der Unternehmenskultur verstanden werden. Nach Ebert und Dumke zeichnen sich leistungsstarke Unternehmen in der Informationstechnologie dadurch aus, dass Messungen ein integraler Bestandteil der technischen Prozesse und Management-Prozesse der Organisation sind [ED07]. Weiterhin werden Messungen nicht als lästige Zusatzarbeit verstanden, sondern als ein nützliches Werkzeug betrachtet, um Dinge besser zu erledigen.

Im Folgenden werden nun die beiden verwandten Begriffe Messung und Metrik genauer definiert.

Eine **Metrik** ist ganz allgemein eine Abbildung von einer Entität auf eine quantitative Größe [IEE90, IEE98, OMG09]. Diese Kenngröße dient dazu, eine bestimmte Eigenschaft der Entität zu beschreiben.

Weiterhin gibt es in der Literatur zwei sich ergänzende Sichten auf den Begriff der Metrik:

- Aus einer operationellen Sicht ist eine Metrik das Verfahren zum Messen einer Kenngröße [ISO07b, ISO05b, GC87, Bal98].
- Aus einer theoretischen Sicht ist eine Metrik die Abbildung von einem empirischen Relationssystem in ein numerisches Relationssystem [FP97, Zus97].

Im Englischen wird statt *Metric* häufig der Begriff *Measurement* verwendet [ISO07b, ISO05b, SMB08]. Dadurch besteht keine Verwechslungsgefahr zu der mathematischen Abstandsfunktion [Zus97]. Im Deutschen ist Metrik der gängige Begriff [Bal98, LL07, Sch07]. Der Begriff Maß als Gegenstück zum englischen *Measurement* findet sich nur selten.

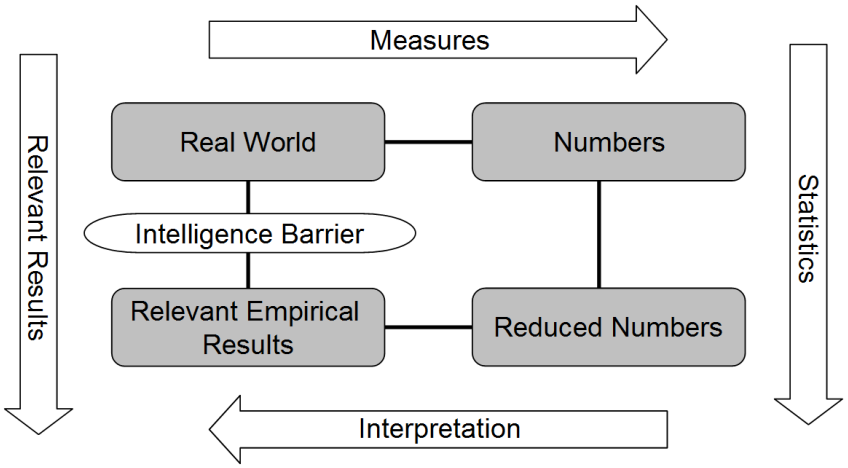


Abbildung 2.1.: Messung nach Kriz ([ED07], siehe auch [Zus97, Kri88])

Die Begriff **Messung** bezeichnet die Anwendung einer Metrik [ISO05b, IEE98]. Abbildung 2.1 stellt die Zusammenhänge bei einer Messung auf intuitive Weise dar. Die von Kriz mit dem Begriff *Intelligence Barrier* [Kri88] benannte Problematik besteht darin, dass bestimmte Eigenschaften von Entitäten aus der realen Welt vom Menschen nicht unmittelbar erfasst und bewertet werden können (z.B. die Wartbarkeit einer Software). Deswegen werden Metriken als Hilfsmittel verwendet. Entitäten aus der realen Welt werden mit Hilfe von Metriken auf quantitative Größen abgebildet. Mathematische Hilfsmittel aus dem Bereich der Statistik (z.B. Mittelwerte, Varianz, Korrelation) dienen dazu, diese Ergebnisse aufzubereiten, um gezielt Fragestellungen beantworten können. Dazu müssen die aufbereiteten Ergebnisse interpretiert werden, um zu relevanten Aussagen über die betrachteten Entitäten zu gelangen. Den Hintergrund dazu gibt die Messtheorie. Beispielsweise wird durch den Skalentyp einer Metrik bestimmt, welche Operationen auf den Ergebnissen zulässig sind [Zus97, LL07]. Diese auf Basis der Messergebnisse gewonnenen Aussagen können wiederum gegen eine Expertenbewertung der betrachteten Entitäten aus der realen Welt validiert werden.

2.1.2. Messprozess

Die notwendigen Tätigkeiten und Arbeitsschritte für die Vermessung von Produkten und Prozessen lassen sich in Form eines Prozessmodells beschreiben. Ein solches Modell findet sich im Standard ISO/IEC 15939. Es soll im Folgenden kurz vorgestellt werden, da es eine gute begriffliche Einordnung der einzelnen Tätigkeiten im Messprozess ermöglicht.

Der Messprozess bezieht sich auf **technische Prozesse und Management-Prozesse** einer Organisation. Aus diesen Prozessen ergeben sich Informationsbedürfnisse und Anforderungen an den Messprozess. Das Prozessmodell für den Messprozess ist grob in die folgenden vier Aktivitäten gegliedert (vgl. Abb. 2.2).

1. **Schaffung von verbindlichen Rahmenbedingungen**: Dazu gehören die Festlegung des Anwendungsbereichs der Messungen (z.B. ein Projekt, ein Standort, ein Geschäftsbereich oder das gesamte Unternehmen), die Bereitstellung von Personal und die Zuordnung von Verantwortlichkeiten.
2. **Planung des Messprozesses**: Informationsbedürfnisse werden identifiziert und priorisiert. Darauf basierend müssen Metriken beschrieben werden, sowie Verfahren zu deren Ermittlung, Analyse und dem Berichtswesen. Weiterhin müssen bereits jetzt Kriterien zur Bewertung der Qualität der Messung, sowie des Messprozesses bestimmt werden. Schließlich müssen für die Durchführung der Messung Personal sowie weitere Ressourcen, wie etwa Weiterbildungen oder unterstützende Werkzeuge, bereitgestellt werden.
3. **Durchführung der Messungen**: Dazu müssen die Tätigkeiten Datenerhebung, Analyse und Berichtswesen in die im Anwendungsbereich der Messungen relevanten Prozesse integriert werden.
4. **Messbewertung**: Die Messergebnisse und der Messprozess werden gegen die in Schritt 2 festgelegten Kriterien evaluiert. Auf Basis der gesammelten Erfahrungen muss der Messprozess weiter verbessert werden.

Die genannten Aktivitäten laufen iterativ ab, um Erfahrungen aus dem Messprozess und Rückmeldungen der Anwender zu berücksichtigen. Bei der Planung des Messprozesses ist es wichtig, dass die Messungen sich an den Zielen der Organisation orientieren. Ein Ansatz, um dies sicherzustellen, ist der *Goal-Question-Metric-Approach*, auf den im folgenden Abschnitt eingegangen wird.

2.1.3. Zielgerichtete Definition von Metriken

Der *Goal-Question-Metric-Approach* (GQM-Ansatz) ist ein systematisches Vorgehen, um Metriken zielgerichtet auszuwählen und zu definieren [BW84, BR88]. Dabei werden zunächst die für eine Organisation oder ein Projekt relevanten Ziele definiert. Ausgehend von diese Zielen werden Fragen abgeleitet, die geklärt werden müssen, um das Erreichen der Ziele zu überprüfen. Schließlich

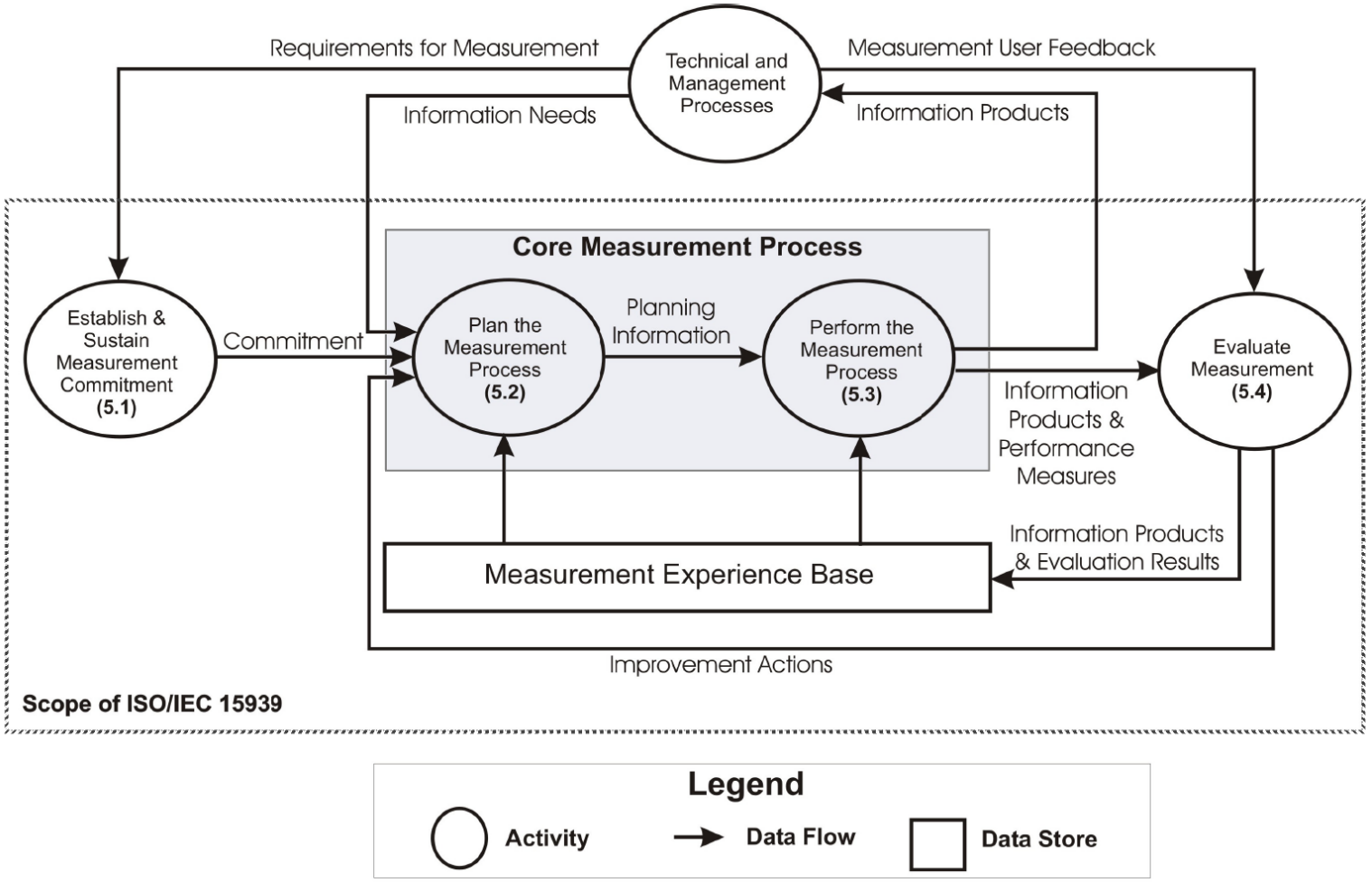


Abbildung 2.2.: Messprozess nach ISO/IEC 15939 [ISO07b]

werden Metriken definiert, die dabei helfen sollen, diese Fragen zu beantworten. Mit diesem Ansatz wird die Menge der zu erhebenden Metriken möglichst klein gehalten. Gleichzeitig wird gewährleistet, dass die gesammelten Messdaten im Kontext des Messziels aussagekräftig sind.

Ziele, abgeleitete Fragen und die zugehörigen Metriken lassen sich in einem Baum oder einem gerichteten azyklischen Graphen darstellen. Für die Schritte im GQM-Ansatz existieren zudem eine Reihe von Hilfsmitteln [Sch07]. Ziele können mit Hilfe von Facetten genauer klassifiziert werden. Bewährt haben sich die Facetten Zweck, Qualitätsaspekt, Betrachtungsgegenstand und Perspektive. Der Zweck der Messung ist beispielsweise die Analyse, das Überwachen oder das Verbessern eines Prozesses. Qualitätsaspekte sind zum Beispiel Zuverlässigkeit, Wartbarkeit, Effizienz oder Benutzbarkeit. Die Perspektive bezieht sich auf die beteiligten Rollen, wie etwa Entwickler, Tester, Projektleiter oder Kunde.

Abstraction Sheets unterstützen die genaue Analyse eines Ziels [vSB99]. Sie dienen insbesondere als Kommunikationsmittel zwischen den Beteiligten. In einem *Abstraction Sheet* werden die folgenden Elemente erfasst:

- von den Projektbeteiligten vorgeschlagene Metriken, um die Zielerreichung zu messen,
- eine Hypothese über die erwarteten Ergebnisse,
- mögliche Einflussfaktoren auf das Ergebnis in dem betrachteten Kontext,
- sowie deren Auswirkung auf die Messergebnisse.

Es gibt eine Reihe von Erweiterungen, die den GQM-Ansatz um bestimmte Aspekte ergänzen. Bei dem GQIM-Ansatz werden zusätzlich Indikatoren zu den Fragen und den abgeleiteten Metriken betrachtet [PGF96]. Ein Indikator ist in diesem Zusammenhang eine bestimmte Darstellungsform der Messergebnisse. Damit soll sichergestellt werden, dass die Messungen sinnvoll kommuniziert und verwendet werden. Der Ansatz GQM++ enthält unter anderem eine mehrstufige Verfeinerung der Ziele und Fragen, sowie eine Kosten- und Nutzenbetrachtung der Messungen [Mac93, GM97]. Mit dem Ansatz *GQM+Strategies* wird darauf abgezielt, die Verbindung zwischen Geschäftszielen, Strategien, organisationsbezogenen Annahmen sowie den Messzielen explizit zu modellieren [BHL+07].

Der GQM-Ansatz kann eingebettet werden in einen kontinuierlichen Verbesserungsprozess nach dem *Quality Improvement Paradigm* (QIP) [Bas89]. In diesem Prozess wird systematisch Erfahrungswissen über Prozessverbesserungsmaßnahmen gesammelt, analysiert, für die Organisation aufbereitet und bereitgestellt. Das Konzept der *Experience Factory* [BCR84, BCM+92] beschreibt einen organisatorischen Rahmen mit entsprechenden Rollen und Verantwortlichkeiten für die Umsetzung des *Quality Improvement Paradigm*.

Die detaillierte Definition von Metriken und deren Erhebungsmethoden sind stark abhängig vom Kontext der Messungen. Daher kann ein allgemeiner Ansatz wie GQM hier nur wenig Unterstützung bieten [Sch07].

In dieser Arbeit wird die Entwicklung von Metriken auf Issue-Tracking-Systemen betrachtet, insbesondere im Hinblick auf die detaillierte Beschreibung und Validierung von Metriken. Auf verwandte Arbeiten im Bereich der Beschreibung von Metriken wird in Abschnitt 4.2 eingegangen. Verwandte Arbeiten in Bezug auf die Entwicklung und speziell die Validierung von Metriken werden in Kapitel 5 vorgestellt.

2.1.4. Software-Entwicklungsarchive

Fast alle Aktivitäten in der Softwareentwicklung werden durch Werkzeuge unterstützt [Som07]. Dabei werden routinemäßig Informationen über den Ablauf des Entwicklungsprozesses erfasst und archiviert. Systeme in denen solche Informationen erfasst und abgelegt werden, möchten wir mit dem Oberbegriff **Software-Entwicklungsarchive** bezeichnen. Im Englischen ist der Begriff *Software Repository* gebräuchlich [DGPH06].

Beispiele für Software-Entwicklungsarchive sind Issue-Tracking-Systeme, Konfigurationsmanagement-Systeme, Test- und *Build*-Systeme, sowie Ablagesysteme für elektronische Kommunikation.

Der Schwerpunkt dieser Arbeit liegt auf der Auswertung von Daten aus Issue-Tracking-Systemen und Konfigurationsmanagement-Systemen. Die relevanten Grundbegriffe in diesem Bereich werden im Folgenden erläutert.

Konfigurationsmanagement befasst sich mit der Identifizierung, Ablage und Verwaltung von Software-Einheiten, sowie mit der Rückverfolgbarkeit von Änderungen an Software-Einheiten (nach [ISO05a]).

Unter Issue-Tracking verstehen wir die Verwaltung von Anfragen in Bezug auf ein Softwaresystem. Diese Anfragen können in folgende Kategorien eingeteilt werden [ISO05a]:

- Ein **Incident** ist ein Ereignis, das nicht zum standardmäßigen Betrieb gehört, und das tatsächlich oder potentiell eine Unterbrechung oder eine Minderung der Qualität der durch eine Software erbrachten Dienstleistungen verursacht [ISO05a]. *Incidents* können untergliedert werden in Support-Anfragen und Störungsmeldungen. Eine **Support-Anfrage** (auch *Service-Request*) ist die Anfrage eines Anwenders zur Unterstützung, Service-Erweiterung, Lieferung, Information, zum Rat oder Dokumentation [vB05]. Eine **Störungsmeldung** beschreibt eine Abweichung zwischen dem tatsächlichen und dem erwarteten Verhalten des Systems. Andere gebräuchliche Begriffe sind unter anderem Abweichungsmeldung [SHT05] und *Trouble Ticket*.
- Eine **Problemmeldung** beschreibt eine unerwünschte Situation, hinweisend auf die noch unbekannte Ursache einer oder mehrerer (potenzieller) Störungen [ISO05a]. Andere gebräuchliche Begriffe sind Mängelmeldung [SHT05], *Problem Report* [CMM06] und *Bug Report* [Bug09a]. Oft wird

auch keine Unterscheidung zwischen Problemmeldung und Störungsmeldung vorgenommen.

- Ein **Änderungsantrag** ist die Forderung nach einer Änderung an Software-Einheiten. Ein Änderungsantrag kann auf einen Erweiterungswunsch, eine Support-Anfrage oder eine Störungsmeldung zurückgehen. Andere gebräuchliche Begriffe für Änderungsantrag sind auch Änderungsanforderung [VXT08], *Change Request* [CMM06], *Modification Request* [ABD+04] und *Request for Change* [vB05].

Ein Issue-Tracking-System unterstützt die Erfassung, Priorisierung, Klassifizierung, Verfolgung und kontrollierte Durchführung von Änderungen, bzw. die Dokumentation der Lösung zu Support-Anfragen, Störungsmeldungen und Problemberichten. Oft wird nicht scharf zwischen allen genannten Kategorien von Anfragen unterschieden, da die Behandlung innerhalb des Issue-Tracking-Systems sehr ähnlich ist. Aus Störungsmeldungen können sich Problemberichte ergeben, und aus diesen wiederum neue Änderungsanträge. Im Deutschen gibt es keinen etablierten Oberbegriff für diese Kategorien. Aus Gründen der Lesbarkeit wird im Folgenden durchgängig der Begriff Änderungsantrag verwendet. Wenn sich eine Aussage jeweils nur auf Support-Anfragen, Störungs- oder Problemmeldungen bezieht, wird dies explizit kenntlich gemacht.

Ein Issue-Tracking-System gibt einen **Workflow** für die bearbeiteten Änderungsanträge vor. Abbildung 2.3 zeigt als Beispiel den standardmäßig vorgegebenen Workflow im Werkzeug Bugzilla. Typischerweise werden die verwendeten Zustände und erlaubten Zustandsübergänge organisationsspezifisch angepasst.

Beim Erstellen eines Änderungsantrags wird versucht, die zur Bearbeitung benötigten Informationen möglichst vollständig zu erfassen. Dazu gehören eine Beschreibung des Sollverhaltens sowie die beteiligten Personen und ihre Verantwortlichkeiten. Bei Störungs- und Problemmeldungen kommen unter anderem eine Beschreibung des Ist-Verhaltens, Informationen zur Umgebung sowie eine Klassifizierung des Fehlers hinzu [SHT05]. Während der Abarbeitung werden dann weitere Informationen ergänzt oder angepasst, wie beispielsweise geschätzte und tatsächliche Aufwände und Termine.

Die im Issue-Tracking-System erfasste Änderungshistorie der zu einem Änderungsantrag erfassten Daten und seiner Statuswechsel im Workflow bezeichnen wir im Folgenden als **Lebenslauf** eines Änderungsantrags.

2.2. Qualitätsmodelle

Eine Vielzahl sehr unterschiedlicher Modelle wird mit dem Begriff Qualitätsmodell bezeichnet. Hilfreich zur Begriffsklärung ist die von Deissenboeck et al. vorgeschlagene Klassifikation nach dem Einsatzzweck [DJLW09].

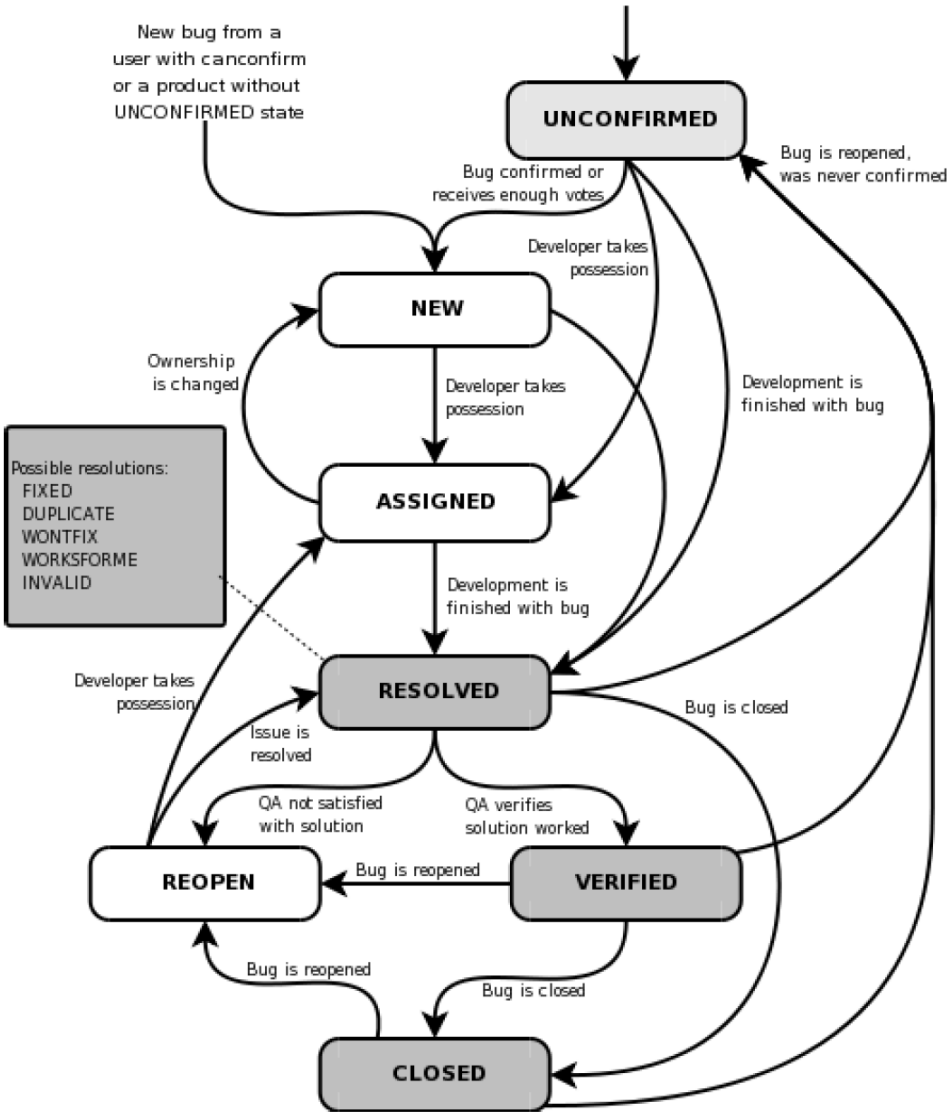


Abbildung 2.3.: Beispiel für mögliche Zustandswechsel von Änderungsanträgen [Bug09a]

- Modelle zur **Definition** von Qualität dienen dazu, genauer zu bestimmen, was unter Qualität verstanden wird, etwa durch die Definition von einheitlichen Begriffen für Qualitätsziele und deren hierarchische Strukturierung. Ein Beispiel ist der Standard ISO/IEC 9126-1 [ISO01].

- Modelle zur **Qualitätsbewertung** beschreiben ein systematisches Vorgehen, um Qualitätseigenschaften eines Produktes oder Prozesses zu bewerten. Beispiele sind das Modell CMMI zur Bewertung des Reifegrades von Software-Prozessen [CMM06] oder der Code-Quality-Index zur Bewertung der technischen Qualität eines Softwaresystems [SSM06].

- Modelle zur **Vorhersage** von Qualität dienen dazu, Prognosen über Qualitätseigenschaften zu treffen. Beispiele sind Zuverlässigkeitsmodelle [Lyu96] oder das Modell COQUALMO zur Vorhersage der Fehlerdichte in einem Softwareprodukt [CB99].

Idealerweise sollen die Modelle aufeinander aufbauen. Also müssen Modelle zur Qualitätsbewertung auch die Definition von Qualität betrachten. Ebenso müssen Vorhersagemodelle auch Bewertung und Definition von Qualität berücksichtigen.

Im Folgenden konzentrieren wir uns auf Modelle zur Qualitätsbewertung. Neben allgemein verwendbaren Referenzmodellen können Modelle zur Qualitätsbewertung auch speziell für einen Einsatzkontext entwickelt oder angepasst werden. Dazu müssen Qualitätsanforderungen des Kunden abgefragt, priorisiert, sowie für den konkreten Kontext detailliert beschrieben und strukturiert werden [Sch07]. Im Sinne der allgemeinen Modelltheorie [Sta73] kann dabei sowohl eine deskriptive als auch eine präskriptive Modellbildung stattfinden. Eine deskriptive Modellbildung zielt auf ein besseres Verständnis unterschiedlicher Qualitätsmerkmale ab. Präskriptive Modellbildung unterstützt Qualitätsplanung und Qualitätssicherung.

2.2.1. Historische Entwicklung

Frühe Ansätze zur Qualitätsbewertung konzentrieren sich auf die Bewertung von Software-Produkten, insbesondere auf Basis von Code-Metriken.

Vermutlich der erste dokumentierte Ansatz zur quantitativen Bewertung von Software stammt von Rubey und Hartwick [RH68]. In diesem Beitrag werden Qualitätseigenschaften von Software definiert, die sich überwiegend auf die technische Qualität von Softwaresystemen beziehen. Jede Qualitätseigenschaft wird dann in mehrere weitere Eigenschaften verfeinert, die durch Vermessung des Quellcodes bewertbar gemacht werden sollen. Rubey und Hartwick sprechen schon explizit von einem Qualitätsmodell und diskutieren die Gewichtung der unterschiedlichen Qualitätseigenschaften durch den Anwender. Weitere Ansätze aus dieser Zeit befassen sich mit der Definition und Abgrenzung unterschiedlicher Qualitätsziele [Wul73, AMPS74].

Cavano und McCall [CM78] stellen ein hierarchisches Qualitätsmodell vor, bei dem zwischen management-orientierten Qualitätszielen auf einer höheren Ebene (*Factors*) und software-bezogenen Qualitätskriterien (*Criteria*) unterschieden wird. Diese sollen dann schließlich mit Hilfe von Metriken bewertet werden. Zu diesem Zeitpunkt waren allerdings sehr wenige Beziehungen zwischen Metriken und Qualitätszielen empirisch validiert.

Höhere Bekanntheit erreichte der von Boehm et al. vorgeschlagene Qualitätenbaum [BBL76]. In der Studie wurden 151 Metrik-Kandidaten definiert und ihre Korrelation mit den Qualitätszielen im Qualitätenbaum bewertet. Weiterhin wird darauf hingewiesen, dass eine Auswahl von Metriken nicht erschöpfend für eine Gesamtbewertung der Qualität sein kann, sodass die Ergebnisse eher einen Hinweischarakter haben.

Eine erste umfassende Untersuchung zur Einführung eines unternehmensweiten Metrik-Programms wurde von Grady und Caswell vorgelegt [GC87]. Zur Strukturierung der verwendeten Metriken wird hier ein mit dem Akronym FURPS benanntes Modell verwendet. Dieses konzentriert sich nicht mehr ausschließlich auf Aspekte der technischen Qualität, sondern rückt auch für den Endbenutzer relevante Aspekte wie Benutzbarkeit und Funktionalität in den Vordergrund.

Aufbauend auf vorher genannten Arbeiten wurde 1991 der Standard ISO/IEC 9126 veröffentlicht. In diesem Standard wird grundsätzlich zwischen interner Qualität, externer Qualität und Gebrauchsqualität (*quality in use*) unterschieden. Weiterhin werden Qualitätsmerkmale auf einer höheren Ebene und entsprechende Submerkmale definiert.

Dromey unterscheidet zwischen Qualitätsmerkmalen auf einer höheren Ebene und direkten Eigenschaften des Produkts (z.B. Verwendung eines aussagekräftigen Namens für ein Software-Modul) [Dro95]. Er kritisiert, dass sich die vorgenannten Modelle im Wesentlichen auf geforderte Qualitätsmerkmale konzentrieren, und somit nicht konstruktiv eingesetzt werden können, um die Qualität bei der Erstellung von Software zu erhöhen. Sein Vorschlag eines Qualitätsmodells basiert auf einem *Bottom-up*-Vorgehen, bei dem zunächst direkte Eigenschaften der Software analysiert und strukturellen Elementen (z.B. Modul, Anweisungssequenz, Variable) zugeordnet werden. Dann wird die Beziehung dieser Eigenschaften zu übergeordneten Qualitätsmerkmalen modelliert.

Kitchenham et al. kritisieren die Dekomposition der Qualitätsmerkmale im Standard ISO/IEC 9126 als willkürlich [KLPN97]. Der von Kitchenham vorgestellte SQUID-Ansatz basiert darauf, dass Qualitätsziele spezifisch für ein Softwareprodukt ausgeprägt werden. Für den Einsatzkontext des Softwareproduktes müssen dann entsprechende quantifizierbare Qualitätsanforderungen erstellt werden.

Nachdem zunächst der Fokus bei der Qualitätsbewertung auf der Produktqualität lag, erweiterte sich die Perspektive in den Achtzigern auf die Bewertung von Prozessen. Damit vollzog sich in der Softwareindustrie eine ähnliche Entwicklung wie einige Jahrzehnte früher im traditionellen industriellen Produktionsbereich. Eine erste Untersuchung zur Bewertung der Prozessqualität wurde 1985 bei

IBM vorgestellt [RHMP85]. Dazu wurde das von Crosby eingeführte fünfstufige Reifegrad-Modell für das Qualitätsmanagement [Cro79] auf den Bereich der Softwareentwicklung übertragen. 1987 wurde von SEI und MITRE ein Rahmenwerk zur Bewertung der Prozessreife entwickelt, basierend auf einem Fragebogen von Ja/Nein-Fragen [HS87]. Die Erfahrungen beider Ansätze bildeten schließlich den Ausgangspunkt zur Entwicklung des Modells CMM [PCC91]. Das Nachfolgemodell CMMI besteht aus drei Teilmodellen für die Anwendungsgebiete Softwareentwicklung, IT-Dienstleistung und Einkauf von Hard- und Software [CMM06].

In diesem Abschnitt wurde ein kurzer Überblick über die historische Entwicklung vom Modellen zur Qualitätsbewertung gegeben. In Abschnitt 2.3.2 werden aktuellere Entwicklungen im Bereich der Prozessbewertung vorgestellt, und detaillierter auf die Informationsbedürfnisse zur Prozessbewertung eingegangen.

2.2.2. Werkzeugunterstützung für Qualitätsmodelle

Der überwiegende Teil der Werkzeuge zur metrik-basierten Qualitätsbewertung findet sich im Bereich der Quellcode-Metriken. Viele dieser Werkzeuge zeigen nur die Messergebnisse und gegebenenfalls Verletzungen von Grenzwerten an. Einige Werkzeuge stützen die Bewertung auf ein Qualitätsmodell, welches in begrenzter Weise angepasst werden kann. Das Werkzeug *Logiscope* [IBM08] verwendet beispielsweise ein Qualitätsmodell ähnlich dem von Cavano und McCall [CM78]. Das Werkzeug *Swat4j* verwendet ein an den Standard ISO/IEC 9126 angelehntes Qualitätsmodell mit linearen Gleichungen zur Aggregation von Merkmalswerten und konfigurierbaren Gewichtungen. Plösch et al. stellen das Werkzeug *SPQR* vor [PGP+08], welches das Anlegen von benutzerdefinierten Qualitätsmodellen und die Anbindung von Werkzeugen zur Quellcode-Vermessung unterstützt. Weiterhin ermöglicht das Werkzeug eine Kommentierung der Messergebnisse durch Experten. Damit können beispielsweise Einschätzungen für den Schweregrad oder den Aufwand zur Behebung eines Problems erfasst werden.

Eine ganze Reihe von Werkzeugen unterstützt die Integration und Visualisierung von Messwerten aus heterogenen Datenquellen [SMN09, KHL01, LK03]. Ein Beispiel ist das Werkzeug *ConQAT* [DJH+08]. Es unterstützt die Komposition von vorgegebenen oder selbst entwickelten Komponenten für die Ermittlung von Metriken, deren Aggregation und Visualisierung. Somit kann mit diesem Werkzeug implizit auch ein Qualitätsmodell beschrieben werden.

Das Werkzeug *DesCOTS-QM* unterstützt die Definition von Qualitätsmodellen zur Bewertung von *Commercial-off-the-shelf*-Komponenten [CFGQ04]. Die zu Grunde liegenden Metriken müssen allerdings manuell erhoben werden und es ist unklar, ob das Werkzeug die Aggregation der Messergebnisse unterstützt.

2.3. Überwachung und Bewertung von Prozessen

In diesem Abschnitt werden verschiedene Tätigkeitsfelder und damit verbundene Referenzmodelle vorgestellt, die sich mit der Überwachung und Bewertung von Prozessen befassen. Dabei soll herausgearbeitet werden, welche unterschiedlichen Informationsbedürfnisse bestehen. Anhand von Beispielen wird vorgestellt, welche Metriken zur Überwachung und Bewertung herangezogen werden. Insbesondere wird verdeutlicht, wie die Auswertung von Software-Entwicklungsarchiven herangezogen werden kann, um eine Reihe der bestehenden Informationsbedürfnisse zu erfüllen.

In Abschnitt 2.3.1 wird die Überwachung von Prozessen im Rahmen des Projekt- und Produktmanagements betrachtet. Abschnitt 2.3.2 widmet sich dann Ansätzen zur Bewertung der Prozessreife innerhalb einer Organisation. Bei diesen Ansätzen werden schwerpunktmäßig Prozesse der Softwareentwicklung betrachtet. Daneben können auch die Prozesse zum Erbringen von IT-Dienstleistungen betrachtet werden. Diese aus dem IT-Controlling motivierte Sichtweise und entsprechende Referenzmodelle werden in Abschnitt 2.3.3 vorgestellt. Schließlich folgt in Abschnitt 2.3.4 eine Beschreibung von Bewertungsansätzen für Open Source Software und deren Entwicklungsprozesse.

2.3.1. Projekt- und Produktmanagement

Nach der DIN 69901 ist ein Projekt folgendermaßen definiert:

> Ein Projekt ist ein Vorhaben, das im Wesentlichen durch Einmaligkeit der Bedingungen in ihrer Gesamtheit gekennzeichnet ist, wie z. B.: Zielvorgabe, zeitliche, finanzielle, personelle oder andere Bedingungen, Abgrenzungen gegenüber anderen Vorhaben und projektspezifische Organisation. [DIN09]

Projektmanagement definiert die Norm als die Gesamtheit von Führungsaufgaben, -organisation, -techniken und -mitteln für die Abwicklung eines Projekts [DIN09].

Projekte müssen hinsichtlich der Aufgaben, Kosten, Termine, Personal und Ressourcen geplant werden [Buh04]. Ein zentraler Bestandteil des Projektmanagements ist ein Regelkreis, bei dem die Plandaten mit den Ist-Daten verglichen werden, um Abweichungen frühzeitig zu erkennen und entsprechend zu reagieren. Zum einen kann steuernd auf das Projekt eingegriffen werden, zum anderen muss möglicherweise die Planung angepasst werden (vgl. Abb. 2.4).

Der Regelkreis enthält den Prozess der Projektüberwachung (*Monitor Process*) und den Prozess der Projektsteuerung (*Control Process*)[ABD+04]. Im Rahmen der Projektüberwachung wird die Einhaltung des Plans kontinuierlich zu festgelegten Zeitpunkten überprüft. Bei Abweichungen kann zwischen

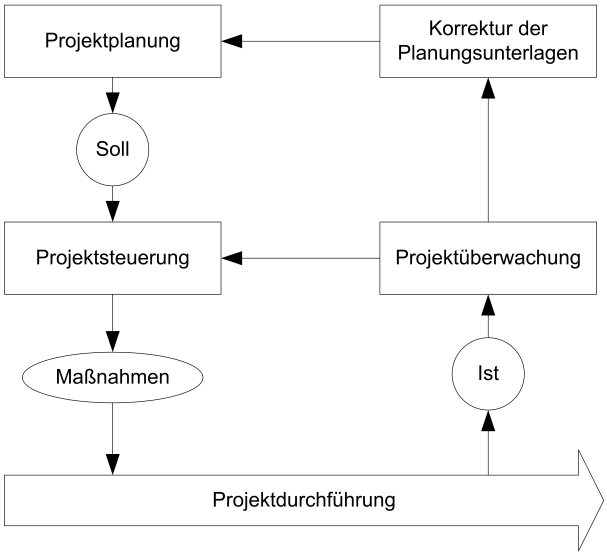


Abbildung 2.4.: Projektmanagement als Regelkreis (nach [Buh04])

Termin-, Kosten-, Funktionalitäts- und Qualitätsproblemen unterschieden werden [HHMS09]. Im Prozess der Projektsteuerung muss über geeignete Maßnahmen entschieden und diese müssen eingeleitet werden. Weiterhin müssen Informationen über die Planeinhaltung an externe Anspruchsgruppen (*Stakeholder*) und innerhalb der Organisation kommuniziert werden.

Im Rahmen eines Messprozesses müssen die für die Projektüberwachung relevanten Informationen erfasst und geeignet analysiert und aufbereitet werden. Zur Analyse stehen eine Reihe von etablierten Methoden zur Verfügung. Beispiele sind die Meilenstein-Trendanalyse, das Kostenvergleichsdiagramm [Hub96] und die Leistungswertanalyse (*Earned Value Analysis*). Viele der benötigten Informationen werden vollständig oder teilweise in Software-Entwicklungsarchiven erfasst, beispielsweise die Größe und die Volatilität des Projekts gemessen in Quellcodezeilen, die Termineinhaltung, der geleistete Aufwand, der Status von Anforderungen oder die Fehlerdichte [ED07]. Bei agilen Projekten basiert die Überwachung auf der Anzahl der fertig gestellten Arbeitsaufträge, oder bei größeren Projekten auf der Anzahl der fertig gestellten Features [OW08, S. 211].

Produktmanagement befasst sich mit der Planung, Überwachung, Messung und Verbesserung der Erhaltungs- und Weiterentwicklungsaktivitäten für ein Software-Produkt [SHT05]. Es umfasst somit sämtliche Projekte im Lebenszyklus des Produktes, also Entwicklungs-, Erhaltungs-, Sanierungs-, Integrations- und Migrationsprojekte. Software-Produktmanagement in einem weiteren Sinne enthält auch die Tätigkeitsbereiche *Portfolio Management* und *Product Roadmapping*, in denen die langfristige Weiterentwicklung eines Produkt-Portfolios bzw. eines einzelnen Produktes geplant wird.

Sneed et al. unterscheiden als primäre Ziele des Produktmanagements die Kontinuität der Dienstleistung und die Erhaltung der Funktionalität, und als se-

kundäre Ziele die Erfüllung zusätzlicher Anforderungen und die Steigerung der Qualität [SHT05]. Davon werden dem GQM-Ansatz folgend eine Reihe von Metriken abgeleitet, beispielsweise die Mängeldichte, die Änderungsrate oder die Systemvolatilität. Die relevanten Metriken ähneln somit denen für die Überwachung eines Entwicklungsprojekts. Allerdings kann beim Produktmanagement die Entwicklung über längere Zeiträume betrachtet und verglichen werden. Außerdem sind potentiell mehr Informationen über im Betrieb des Produkts festgestellte Probleme oder Störungen verfügbar.

2.3.2. Prozessbewertung

Prozessbewertung kann grundsätzlich in einem der folgenden Kontexte eingesetzt werden [ISO04]:

- Im Rahmen der **Prozessverbesserung** wird der Ist-Zustand von Prozessen in einer Organisation erfasst und im Hinblick auf Stärken, Schwächen und Risiken bewertet. Diese Bewertung ist der Ausgangspunkt für die Einleitung von Verbesserungsmaßnahmen.
- Bei der **Bestimmung der Prozessreife** wird der Ist-Zustand von Prozessen in einer Organisation gegen ein definiertes Fähigkeitsprofil verglichen. Das Fähigkeitsprofil orientiert sich dabei an einem Prozess-Referenzmodell. Der für eine Organisation bestimmte Reifegrad kann dann bei der Entscheidung über die Auswahl dieser Organisation als Auftragnehmer herangezogen werden.

Das heute in der Softwareentwicklung am weitesten verbreitete Qualitätsmodell zur Bewertung von Prozessreife ist das bereits im vorherigen Abschnitt eingeführte Modell CMM bzw. sein Nachfolgemodell CMMI (*Capability Maturity Model Integration*) [CMM06]. Das Modell definiert fünf Reifegradstufen. Jeder der Reifegradstufen sind eine Reihe von Prozessgebieten zugeordnet. CMMI ermöglicht neben der Bewertung nach Reifegradstufen (*Staged Representation*) eine sogenannte kontinuierliche Bewertung, in der eine feinere Darstellung von einzelnen Prozessgebieten möglich ist (*Continuous Representation*).

Das Prozessgebiet *Measurement and Analysis*, welches sich mit der Definition, Erfassung und Auswertung von Metriken befasst, ist bereits der Reifegradstufe 2 zugeordnet. Dies zeigt die hohe Bedeutung von Messung für die Prozessverbesserung. Während auf Stufe 2 Metriken noch projektbezogen erfasst werden, sind auf Stufe 3 zusätzlich organisationsübergreifende Standards für Metriken gefordert. Dies ermöglicht, die Metriken auch für projekübergreifende Vergleiche zu verwerten. Die im Prozessgebiet *Measurement and Analysis* beschriebenen Praktiken dienen dazu, die Informationsbedürfnisse in anderen Prozessgebieten zu erfüllen. Im Kontext dieser Arbeit sind hier beispielsweise die Gebiete *Project Monitoring und Control*, welches auf die Fortschrittsüberwachung der Projekte abzielt, sowie *Requirements Management* zu nennen. Im CMMI selbst sind keine Metriken definiert. Zu einzelnen Praktiken werden jedoch Beispiel-Metriken aufgeführt.

Angelehnt an CMMI und das Modell Bootstrap [SE96] wurde der Standard ISO/IEC 15504 entwickelt [ISO04], auch bekannt unter dem Namen SPICE (*Software Process Improvement and Capability Determination*). SPICE beschreibt ein allgemeines Vorgehen zur Bewertung von einzelnen Prozessen. Es werden allerdings keine verbindlichen Prozesse definiert. Stattdessen muss sich die Bewertung auf Prozessreferenzmodelle stützen. Als ein solches Referenzmodell kann beispielsweise der Standard ISO/IEC 12207 [ISO95] (Prozesse im Software-Lebenszyklus) herangezogen werden. Das genaue Vorgehen für die Bewertung von Prozessen entsprechend eines solchen Prozessreferenzmodells muss dann in einem Prozess-Assessment-Modell beschrieben werden.

Der Teil 5 des Standards ISO/IEC 15504 beschreibt ein solches Prozess-Assessment-Modell basierend auf ISO/IEC 12207. Ähnlich wie CMMI werden hier ebenfalls die Tätigkeitsfelder Messung, Problem-Management und Änderungsmanagement abgedeckt. Auch wenn dies nicht expliziet in der Norm gefordert ist, hat es sich als gute Praxis für die Reifegradstufe 2 erwiesen, Daten zu eingehenden Problemberichten und deren Abarbeitung zu erfassen und auszuwerten [HDHM06].

Ebenfalls angelehnt an CMMI wurden auch Reifegradmodelle für die Softwarewartung entwickelt. Das *Corrective Maintenance Maturity Model* (CM^3) konzentriert sich auf die korrektive Wartung [KM02]. Weiter fortgeschritten ist die Entwicklung des *Software Maintenance Maturity Model* (SM^{mm}) [AHAD05, AA08]. Es deckt Wartungsprozesse umfassender ab, und gliedert diese in vier Prozessbereiche:

- **Prozessmanagement** betrachtet die organisatorischen Prozesse bei der Wartung.
- **Maintenance Request Management** enthält Prozesse zum Umgang und Planung von eingehenden Änderungsanträgen.
- **Evolution Engineering** betrachtet Prozesse zur Fehlerkorrektur und Weiterentwicklung der Software.
- **Support to Evolution Engineering** enthält unterstützende Prozesse wie Qualitätssicherung und Konfigurationsmanagement.

Der letztgenannte Prozessbereich enthält auch den Prozess *Measure and Analysis of Maintenance* welcher die Definition, Erhebung und Auswertung von Metriken zum Inhalt hat [AA09]. In diesem Modell werden allerdings keine konkreten Metriken genannt.

Die in den genannten Reifegradmodellen zur Softwarewartung betrachteten Modelle überschneiden sich mit den in Abschnitt 2.3.3.1 beschriebenen Prozessen des IT Service Managements. Die Sichtweise des IT Service Managements legt den Schwerpunkt allerdings auf die durch IT-Systeme erbrachten Dienstleistungen.

2.3.3. IT-Controlling

Neben den genannten Ansätzen zur Prozessverbesserung ist das IT-Controlling ebenfalls ein Antrieb für die Vermessung von Prozessen im Kontext der Entwicklung und des Betriebs von Systemen der Informationstechnik. Controlling im Allgemeinen erfüllt die Aufgaben der zielorientierte Koordination von Planung und Kontrolle sowie die Informationsversorgung des Managements [Hor03]. Diese beiden Aufgaben werden auch in dem Leitbild für Controller der *International Group of Controlling* deutlich, welches unter anderem folgende Tätigkeiten benennt [IGC02]:

- Controller sorgen für Strategie-, Ergebnis-, Finanz-, Prozesstransparenz und tragen somit zu höherer Wirtschaftlichkeit bei.
- Controller koordinieren Teilziele und Teilpläne ganzheitlich und organisieren unternehmensübergreifend das zukunftsorientierte Berichtswesen.
- Controller moderieren und gestalten den Managementprozess der Zielfindung, der Planung und der Steuerung so, dass jeder Entscheidungsträger zielorientiert handeln kann.
- Controller leisten den dazu erforderlichen Service der betriebswirtschaftlichen Daten- und Informationsversorgung.

Controlling kann als zielorientierte Steuerung eines Systems verstanden werden [Küt05]. Das System ist beispielsweise ein Prozess, ein Projekt oder eine Organisation. Zur Erfüllung der Steuerungsaufgabe muss der Ist-Zustand des Systems regelmäßig erfasst und mit dem geforderten Soll-Zustand verglichen werden. Man spricht dabei auch von dem Controlling-Regelkreis. In diesem Sinne ist Controlling Aufgabe des für die Zielerreichung verantwortlichen Managers. Der Controller selbst hat eine unterstützende und beratende Funktion. Ein wesentlicher Teil dieser Unterstützung besteht im Schaffen von Transparenz.

Durch die Bedeutung der Informationstechnologie in heutigen Organisationen wurde ein eigenes Fachcontrolling notwendig, welches als **IT-Controlling** bezeichnet wird. Dabei wird nicht nur Softwareentwicklung und -wartung, sondern vielmehr der Aufbau und Betrieb einer IT-Infrastruktur betrachtet, die es ermöglicht, IT-Dienstleistungen zu erbringen. Daraus ergeben sich die beiden Sichten des IT-Leistungserstellers sowie des IT-Leistungsnehmers, also der Teil der Organisation, der die bereitgestellten Dienstleistungen in Anspruch nimmt.

Controlling war traditionell finanzwirtschaftlich orientiert. Eine reine Kostensicht kann jedoch zu Fehlentscheidungen führen, beispielsweise beim Outsourcing [Küt06]. Dadurch ergibt sich die Notwendigkeit für ein prozessorientiertes Controlling, um Transparenz über Leistungen zu schaffen. Die erbrachten Leistungen müssen dazu klar definiert und messbar gemacht werden. Dies erleichtert es, die Kosten verursachungsgerecht auf die IT-Leistungsnehmer zu verteilen.

Die mit dem IT-Controlling zu steuernden Objekte ergeben sich aus dem Lebenszyklus von IT-Systemen [Küt05]. In einem Projekt werden Systeme realisiert. Um Dienstleistungen auf Basis eines IT-Systems bereitzustellen, muss der Leistungsersteller Prozesse ausführen, die in gleicher oder ähnlicher Weise durchlaufen werden. Die Kernobjekte des IT-Controllings sind somit IT-Dienstleistungen, Prozesse, Systeme und Projekte.

Allgemein kann zwischen operativem und strategischem Controlling unterschieden werden. Operatives Controlling ist eher kurzfristig angelegt, sodass die Ziele klar definiert sind, und nicht mehr hinterfragt werden müssen [Küt05]. Das strategische Controlling ist langfristig ausgerichtet mit dem Schwerpunkt auf Geschäftsfeld- und Kernkompetenzplanung [Wit02]. Dabei müssen bei der Planung alternative Ziele betrachtet, und alternative Strategien ermittelt und ausgewählt werden. Die Umsetzung einer Strategie erstreckt sich dann über mehrere Planungsperioden, wobei es zu einer Anpassung von Teilzielen oder der Strategie kommen kann.

Im Folgenden werden mit *IT Service Management* und *IT Governance* zwei sich teilweise überschneidende Tätigkeitsfelder vorgestellt, in denen Aufgaben des IT-Controllings wahrgenommen werden. Es sollen dabei Informationsbedürfnisse des IT-Controllings beleuchtet werden, zu deren Beantwortung die im Rahmen dieser Arbeit entwickelten Methoden und Werkzeuge beitragen können.

2.3.3.1. IT Service Management

IT Service Management bezeichnet Prozesse zur Bereitstellung von Dienstleistungen aus dem Bereich der Informationstechnologie. Der Fokus liegt dabei auf der Unterstützung von Geschäftsprozessen einer Organisation.

Die *Information Technology Infrastructure Library* (ITIL) ist der bekannteste Ansatz zur Umsetzung des IT Service Managements. Die Entwicklung von ITIL wurde in den Achtzigerjahren im Auftrag der britischen Regierung begonnen. ITIL ist gegliedert in die Tätigkeitsfelder Servicestrategie [INT07], Serviceentwurf [LRT07], Serviceüberführung [LMT07], Servicebetrieb [CWT07] und kontinuierliche Serviceverbesserung [CST07]. Jedem Tätigkeitsfeld ist eine Reihe von relevanten Prozessen zugeordnet, für die bewährte Vorgehensweisen beschrieben sind.

Angelehnt an ITIL wurde 2005 der Standard ISO/IEC 20000 veröffentlicht. Der Standard definiert Anforderungen an die Planung, Durchführung und Überwachung des IT Service Managements und gibt eine Gliederung für die Prozesse des IT Service Managements vor (vgl. Abb. 2.5).

Überwachung des IT Service Managements bedeutet, dass regelmäßig überprüft werden muss, ob die für das Service Management geplanten Ziele erreicht wurden. Dies enthält auch nach Möglichkeit, die Qualität der bereitgestellten

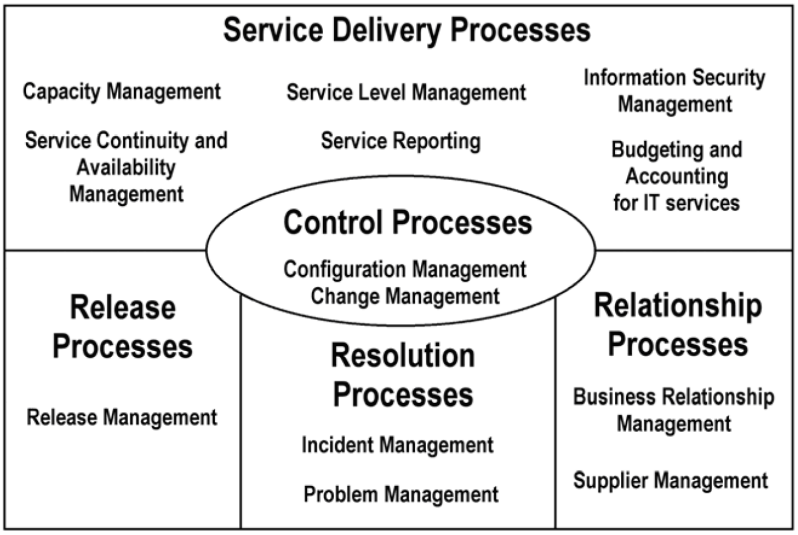


Abbildung 2.5.: Prozesse im IT Service Management [ISO05a]

Dienstleistungen zu messen. Im Kontext dieser Arbeit sind dabei vor allem die folgenden Prozesse relevant:

- Im **Service Level Management** wird die Qualität und Quantität der IT-Dienstleistungen definiert. Entsprechende Vereinbarungen werden in *Service Level Agreements* festgehalten, deren Einhaltung überwacht werden muss.

- Das **Incident Management** nimmt Störungsmeldungen und Anfragen von Anwendern entgegen. Neben dem Ziel einer schnellen Behebung von Störungen, ist auch gefordert, dass eine schnelle Reaktion auf die Anfrage des Anwenders erfolgt, und dieser über den Fortschritt der Bearbeitung informiert wird.

- Ziel des **Problem Managements** ist es, proaktiv Ursachen für bekannte oder potenzielle Störungen zu analysieren und zu beseitigen. Dies erfordert die systematische Erfassung von eingegangenen Anfragen und Störmeldungen, sowie die Dokumentation des Bearbeitungsverlaufs.

- Das **Change Management** muss sicherstellen, dass Änderungsanträge systematisch analysiert, durchgeführt und geprüft werden.

ITIL nennt Metriken für die Vermessung und Steuerung der einzelnen Prozesse. Diese müssen für den praktischen Einsatz allerdings weiter konkretisiert werden [Küt06]. In Tabelle 2.1 sind Beispiele für entsprechende Metriken aufgelistet. Für eine ganze Reihe dieser Metriken werden die benötigten Informationen in einem Issue-Tracking-System erfasst.

Tabelle 2.1.: Beispiele für Metriken aus ITIL

ITIL Prozess	**Metriken**
Service Level Management	• Anzahl und Schwere von SLA-Verletzungen • Wirksame Überprüfung und Verfolgung aller SLA-Verletzungen
Incident Management	• Mittlere Zeit für die Beseitigung oder Umgehung von Störungen - aufgeteilt nach Auswirkungskategorien • Anteil der Störungen, die innerhalb der vereinbarten Lösungszeit beseitigt wurden - aufgeteilt nach Auswirkungskategorien • Anteil der Störungen, die fehlerhaft kategorisiert worden sind • Anteil der nachträglich noch einmal geöffneten Störungsmeldungen • Durchschnittliche Anzahl von Störungen, die von einem Mitarbeiter des 1st-Level-Supports bearbeitet werden
Problem Management	• Mittlere Lösungszeit für Probleme • Zeitdauer, um die Ursachen von Problemen zu diagnostizieren • Mittlere Anzahl offener Probleme und Fehler (Backlog)
Change Management	• Anteil termingerecht umgesetzter Änderungsanträge • Durchschnittliche Dauer zur Durchführung einer Änderung • Anzahl oder Volumen der noch nicht umgesetzten Änderungen (Backlog) • Anzahl der eiligen Änderungen

2.3.3.2. IT Governance

IT Governance kann folgendermaßen definiert werden [IT 03]:

> IT Governance liegt in der Verantwortung des Vorstands und des Managements und ist ein wesentlicher Bestandteil der Unternehmensführung. IT Governance besteht aus Führung, Organisationsstrukturen und Prozessen, die sicherstellen, dass die IT die Unternehmensstrategie und -ziele unterstützt.

Während beim IT Service Management im Wesentlichen ein operatives IT-Controlling durchgeführt wird, steht bei IT Governance ein strategisches IT-Controlling im Vordergrund. Dabei muss ein Gesamtkonzept entworfen und durchgeführt werden, bei dem die Interessen von IT-Leistungsersteller und IT-Leistungsnehmer in Einklang gebracht werden.

Das bekannteste Referenzmodell für IT Governance ist COBIT (*Control Objectives for Information and Related Technology*) [IT 07]. Es definiert insgesamt 34 Prozesse, die alle Tätigkeitsfelder einer IT-Organisation abdecken sollen. Diese Prozesse sind grob gegliedert in vier Bereiche:

- Planung und Organisation,
- Erwerb und Implementierung von Systemen,
- Serviceerstellung und Kundenbetreuung,
- Überwachung.

Der Bereich Serviceerstellung und Kundenbetreuung deckt ähnliche Tätigkeiten ab, wie sie in ITIL beschrieben sind. Der Bereich Überwachung ist als Querschnittsbereich zu betrachten, und beschreibt Aufgaben des IT-Controllings. Dabei liegt der Fokus auf Informationsbeschaffung und der Frage, ob die Prozesse in der definierten Form ablaufen. Controlling im Sinne eines Steuerungsprozesses wird von COBIT nicht abgedeckt [Küt05].

COBIT formuliert Ziele auf vier verschiedenen Ebenen: Geschäftsziele der Organisation, Ziele der IT innerhalb der Organisation, Ziele eines Prozesses und schließlich Ziele einer Aktivität. Zu den Zielen werden Metriken genannt, mit denen gemessen werden soll, wie gut Ziele erreicht werden, und ob die an den Prozess gestellten Anforderungen erfüllt werden. Ähnlich zu den in Tabelle 2.1 genannten Metriken finden sich darunter ebenfalls eine ganze Reihe von Metriken, bei denen Issue-Tracking-Datenbanken als Informationsbasis herangezogen werden können.

Weiterhin beschreibt COBIT Reifegradstufen für die einzelnen Prozesse, ähnlich der *Continuous Representation* des CMMI-Modells.

2.3.4. Bewertung von Open Source Softwareentwicklung

Der Einsatz von Open Source Komponenten in industriell entwickelten Software-Systemen wirft bestimmte Fragestellungen auf. Beispiele sind: Liefert eine Komponente die benötigte Funktionalität? Wird das Open Source Projekt langfristig weiterentwickelt? Sind die Lizenzbedingungen kompatibel zum eigenen Geschäftsmodell? Aus diesen Fragestellungen ist die Bewertung von Open Source Software und deren Entwicklungsprozess motiviert.

Der Vergleich der Produktfunktionalität kann nach gleichen Maßstäben erfolgen, unabhängig, ob es sich um Open oder Closed Source Software handelt. Bei der Betrachtung des Prozesses spielen bei Open Source Software jedoch einige besondere Aspekte eine Rolle, beispielsweise die Aktivität im Projekt, der Umgang mit Support-Anfragen oder die Größe der Anwender- und Entwicklergemeinschaft des Projekts.

Aus dieser Motivation sind verschiedene Qualitätsmodelle zur Bewertung von Open Source Entwicklungsprojekten entstanden. Erste Modelle beschreiben im Wesentlichen ein Vorgehen für eine Expertenbewertung. Darunter fallen die Modelle OSMM [Gol04] (*Open Source Maturity Model*), OpenBRR [Ope05] (*Open Business Readiness Rating*) und QSOS [Ato06] (*Qualification and Selection of Open Source Software*). Alle diese Modelle beschreiben jeweils eine Hierarchie von relevanten Qualitätsmerkmalen sowie ein Vorgehen zur Bewertung von Einzelmerkmalen und deren Aggregation zu einer Indexzahl. Momentan wird von den vorgenannten Modellen nur QSOS aktiv weiterentwickelt. QSOS unterscheidet zwischen Qualitätsmerkmalen, die spezifisch für eine bestimmte Klasse von Produkten sind (z.B. *Groupware*, Datenbanken, etc.), und generischen Qualitätsmerkmalen wie Reife des Produkts, Verbreitung und Aktivität im Projekt.

Das EU-Projekt QualOSS [CS08] (*Quality of Open Source Software*) zielt auf die Entwicklung eines Qualitätsmodells ab, das die Prozessqualität stärker berücksichtigt als die vorgenannten Modelle. Prozesse werden dazu in Anforderungs- und Änderungsmanagement, Release-Management sowie Support- und Community Management unterschieden. Zu jedem Bereich liegt ein umfassender Fragenkatalog vor, der die Begutachtung durch Experten anleitet.

Im Gegensatz zu den bisher genannten Ansätzen konzentriert sich das Modell SQO-OSS [SGSS08] (*Software Quality Observatory for Open Source Software*) auf die Automatisierung der Qualitätsbewertung, um eine kontinuierliche Qualitätsüberwachung zu ermöglichen. Die Funktionalität des Produktes wird dabei explizit nicht betrachtet. Berücksichtigt werden die Qualität des Quellcodes und die Aktivität in der Open Source Community. Für die Bewertung werden dann die öffentlich zugänglichen Software-Entwicklungsarchive herangezogen. Als Basismetriken werden beispielsweise die Anzahl der Nachrichten in Mailinglisten oder die Häufigkeit von Aktualisierungen der Dokumentation verwendet. Dieser Ansatz lässt sich in die im nächsten Abschnitt vorgestellten Verfahren zur Auswertung von Software-Entwicklungsarchiven einordnen.

2.4. Auswertung von Software-Entwicklungsarchiven

Der unter dem Begriff *Mining Software Repositories (MSR)* bekannte Forschungsbereich befasst sich mit der Auswertung von Software-Entwicklungsarchiven, wie Versionskontrollsystemen, *Issue-Tracking-Systemen* und auch Kommunikationsarchiven wie *Newsgroups*. In der Regel werden dabei Änderungen über die Zeit betrachtet, zum Beispiel die Versionshistorie des Quellcodes oder der Bearbeitungsverlauf von Änderungsanträgen.

Erste Arbeiten in diesem Bereich finden sich im Zusammenhang mit Lehmans Untersuchungen zur Software-Evolution [LP76]. Auswertungen von Software-Entwicklungsarchiven waren lange Zeit auf industrielle Software-Projekte beschränkt, deren Software-Entwicklungsarchive nicht öffentlich verfügbar sind. Durch das Aufleben von Open Source Projekten entstand eine breite, öffentlich verfügbare Datenbasis für die Anwendung und Evaluierung von Auswertungsverfahren.

2.4.1. Kategorisierung von Auswertungsansätzen

In den letzten Jahren wurde eine Vielzahl von Ansätzen für sehr unterschiedliche Fragestellungen entwickelt. Diese umfassen Bereiche wie die Evolution von Architektur [BZL06, Hou07, WY08], Kommunikationsstrukturen zwischen Entwicklern und Nutzern einer Software [KG07], thematische Klassifikation der Quellcode-Änderungen von Entwicklern [LRB+07] oder Vorhersagen über die Anzahl von Änderungen [HGBR07], Fehleranzahl [Zha08] oder Bearbeitungszeiten von Problemberichten [Pan07]. Einen umfangreichen Überblick geben Kagdi et al. [KCM07]. Dazu werden die Ansätze nach folgenden Dimensionen kategorisiert:

- **Verwendete Informationsquellen**: Die bei weitem am häufigsten verwendeten Informationsquellen sind Issue-Tracking-Systeme und Versionskontrollsysteme. Andere betrachtete Informationsquellen sind beispielsweise Chat-Protokolle [SJH09], Testüberdeckungsdaten [ZRDvD08], Execution Traces [OAH03], Archive von Mailing-Listen [TMK+06, BGD+06] und Produktivitätsdaten [TMY+06]. Weiterhin gibt es Ansätze, die Verknüpfungen zwischen Entitäten aus unterschiedlichen Quellen betrachten, beispielsweise Änderungen im Quellcode mit später aufgetretenen Fehlerberichten [SZZ05].

- **Einsatzzweck**: Grundlegend für die Unterscheidung zwischen den Auswertungsverfahren ist natürlich die verfolgte Fragestellung. Nach Kagdi et al. können die betrachteten Fragestellungen grob in zwei Kategorien eingeordnet werden [KCM07]. Die erste Kategorie sind Warenkorb-Fragen im Sinne von: Wenn etwas Bestimmtes passiert, was passiert dann in der Regel noch? Die Antwort sind typische Trends oder Abhängigkeiten zwischen bestimmten Ereignissen. Die zweite Kategorie sind Fragen nach der

Häufigkeit des Auftretens von bestimmten Ereignissen (*Prevalence Questions*). Unter die zweite Kategorie fallen die Ermittlung von Metriken (z.B. die Anzahl der geänderten Codezeilen in einem Zeitraum) und boolesche Anfragen (z.B. die Existenz von zyklischen Abhängigkeiten zwischen Paketen).

- **Untersuchungsmethode**: Bei der Untersuchung von Software-Evolution lassen sich prinzipiell zwei Ansätze unterscheiden. Bei dem **indirekten** Ansatz wird der Zustand der betrachteten Artefakte zu früheren Zeitpunkten rekonstruiert. Auf dieser Basis werden jeweils bestimmte Eigenschaften ermittelt, die dann zwischen unterschiedlichen Zeitpunkten verglichen werden. Bei dem **direkten** Ansatz werden die Änderungen der betrachteten Artefakte von einem Zeitpunkt zum nächsten analysiert.

  Im Detail kommen sehr unterschiedliche Techniken zum Einsatz, beispielsweise aus den Bereichen statische Analyse von Quellcode, *Data Mining*, *Supervised Learning* oder *Clone Detection*.

- **Evaluierung**: Die letzte Dimension bei der Kategorisierung ist die Frage, wie ein gegebener Ansatz evaluiert werden kann. Ziel der Verfahren zur Auswertung von Software-Entwicklungsarchiven ist es, aus der Änderungshistorie Erkenntnisse zu gewinnen, die dabei helfen, den Entwicklungsprozess zu verbessern. Dies kann zum einen dadurch geschehen, dass Vorgänge bei der Evolution von Software besser verstanden werden, beispielsweise die Zusammenarbeit von Entwicklern oder das Altern von Software (*Architectural Decay*). Zum anderen gibt es eine Reihe von Ansätzen, die auf Basis der historischen Daten Prognosen erstellen, beispielsweise über die Anzahl von Fehlern in Software-Modulen [OWB05].

  Um solche prognostischen Verfahren zu bewerten, werden üblicherweise die Präzision (*Precision*) und die Vollständigkeit des Ergebnisses (*Recall*) betrachtet. Daneben finden sich Ansätze aus der Informationstheorie zum Vergleich zwischen einer tatsächlichen und einer vorhergesagten Verteilung [AH06].

  Die Evaluierung von Anwendbarkeit und Nutzen in der Praxis ist ein langwieriges Unterfangen, und benötigt Überzeugungsarbeit, um entwickelte Ansätze im industriellen Kontext zum Einsatz zu bringen [OW07]. Während die Skalierbarkeit von vorgeschlagenen Verfahren durch Anwendung auf Daten aus Open-Source Projekten gezeigt werden kann, fehlen oft Erfahrungen oder eine Perspektive für den praktischen Einsatz.

An dieser Stelle konnte nur ein kurzer Überblick über die Bandbreite der Verfahren im Bereich MSR gegeben werden. Im nächsten Abschnitt werden solche Verfahren betrachtet, die verwandt zum Kontext dieser Arbeit sind, also auf die Analyse und Bewertung von Entwicklungsprozessen abzielen.

2.4.2. Auswertungen zur Analyse von Entwicklungsprozessen

Publikationen über Verfahren zur Auswertung von Software-Entwicklungsarchiven mit dem Ziel der Analyse von Entwicklungsprozessen konzentrieren sich auf die Betrachtung von Open Source Entwicklung.

Erste Fallstudien betrachteten das Wachstum des Quellcodes bei Open Source Projekten [GT00, AHK+01]. Die Fallstudie von Mockus et al. betrachtete dann auch Aspekte des Entwicklungsprozesses [MFH02]. Die folgenden Fragestellungen wurden für die Open Source Projekte *Mozilla* und *Apache* auf der Grundlage des Quellcode-Archivs und der Datenbank der Problemberichte untersucht:

- Anzahl der beteiligten Personen an der Entwicklung und beim Berichten von Problemen
- Kumulative Verteilung der beigetragenen Quellcode-Änderungen bzw. Problemberichten zwischen den beteiligten Personen
- Fehlerdichte im Quellcode
- Lösungszeiten für eingestellte Problemberichte

Auf Basis dieser Fallstudien wurde eine Reihe von Hypothesen zur Charakterisierung der Entwicklungsprozesse bei Open Source Projekten formuliert.

Koch und Schneider verwendeten die Größe von Änderungen im Quellcode und die Anzahl von *Postings* in Diskussionsforen am Beispiels des Projekts GNOME, um eine grobe Abschätzung des in einem Open Source Projekt erbrachten Aufwandes zu geben [KS02].

Michlmayr und Senyard untersuchten die Rate der geöffneten und geschlossenen Problemberichte, sowie deren Lösungszeiten für das Projekt *Debian* [MS06].

Francalanci und Merlo untersuchten ebenfalls die Lösungszeit für eingestellte Problemberichte für eine Reihe von Open Source Projekten [FM08]. Sie verwenden dazu eine Visualisierung durch ein Streudiagramm, das den Zeitpunkt der Erzeugung eines Problemberichts gegen die Lösungszeit aufträgt. Eine weitere Visualisierung ist die Entwicklung der Anzahl von offenen und geschlossenen Problemberichten über die Zeit. Auf Basis der Untersuchung wird eine Einteilung in vier Kategorien vorgeschlagen, die unterschiedlichen Qualitätsstufen bei der Bearbeitung von eingehenden Problemberichten entsprechen.

Gasser und Ripoche beschreiben einen Ansatz zur Untersuchung von Abstimmungs- und Entscheidungsprozessen auf Basis von Änderungsanträgen bei Open Source Software [GR03]. Als eine beispielhafte Anwendung wird ein Zustandsdiagramm extrahiert, welches die Häufigkeit von Übergängen zwischen den möglichen Status-Werten eines Änderungsantrags darstellt. Daneben sollen weitere Informationen durch semantische Analyse der textlichen Kommentare gewonnen werden, beispielsweise die Klassifizierung der Kommentare in Kategorien wie „Zustimmung“, „Bewertung“, „Begründung“ oder „Konflikt“. In einer späteren

Untersuchung wird der Grad der Verknüpfungen zwischen Änderungsanträgen betrachtet [SGR04].

Eine Reihe von Ansätzen kombiniert Informationen aus Issue-Tracking-Systemen und Konfigurationsmanagement-Systemen, beispielsweise zum Auffinden von für eine aktuelle Aufgabe relevanten Artefakten wie Quellcode-Dateien oder Problemberichte [CMSB05], zur Klassifikation von Codeänderungen in anpassende, korrigierende oder verbessernde Wartung [MV00], oder zur Vorhersage von Dateien, die auf Grund eines neuen Problemberichts geändert werden müssen [CC05]. Im Ansatz von Sliwerski et al. wird eine Heuristik beschrieben, um auf Basis der Änderungen im Quellcode und der Problemberichte solche Änderungen im Quellcode zu identifizieren, die später bei einer Fehlerbehebung wieder korrigiert wurden [SZZ05]. Somit kann untersucht werden, ob diese fehlerverursachenden Änderungen auffällige Eigenschaften haben, wie beispielsweise der Zeitpunkt der Änderung, oder die Größe der geänderten Datei.

Zusammenfassend lässt sich für die vorgenannten Ansätze feststellen, dass sich diese jeweils auf eine isolierte Fragestellung konzentrieren. Skripte und Werkzeuge zur Extraktion der Informationen aus Software-Entwicklungsarchiven werden jeweils speziell für diese bestimmte Fragestellung entwickelt.

2.4.3. Werkzeuginfrastrukturen zur Metrikauswertung

Neben den oben genannten recht spezialisierten Analysen gibt es Werkzeuge, die für die Analyse der Daten eine Schnittstelle auf einer höheren Abstraktionsebene anbieten. Ein Beispiel ist das Werkzeug *Bloof*, welches Informationen aus CVS in eine eigene Datenbank extrahiert [DP03]. Dazu wird eine JAVA API angeboten, die sowohl eine Reihe von vordefinierte Abfragen als auch SQL-Abfragen unterstützt, um explorative Analysen auf CVS-Daten zu erleichtern.

Eine ähnlicher Ansatz wird bei dem Werkzeug RaSOSS verfolgt [Kop06]. RaSOSS extrahiert Informationen aus Versionskontrollsystemen und Issue-Tracking-Systemen in eine eigene Datenbank. Als *Plug-Ins* für das Werkzeug können dann sogenannte Analyse-Module implementiert werden, beispielsweise zur Ermittlung von durchschnittlichen Lösungszeiten oder von typischen Statuswechseln im Lebenslauf eines Problemberichtes.

Im Kontext des in Abschnitt 2.3.4 vorgestellten Modells SQO-OSS wurde ein Werkzeuginfrastruktur namens *Alithea Core* entwickelt [GS09a]. *Alithea Core* bietet eine einheitliche Schnittstelle für Daten aus heterogenen Software-Entwicklungsarchiven, eine OSGi-basierte Architektur für Metrik-Plug-Ins sowie Unterstützung für die Präsentation der Ergebnisse. Als Beispiel wird ein Metrik-Plug-In vorgestellt, welches darauf abzielt, die Intensität der Diskussionen in Mailinglisten zu vermessen.

Die drei vorgenannten Werkzeuge bieten also jeweils eine Plattform für einen einfacheren Zugriff auf Daten in Software-Entwicklungsarchiven. Dies ist insbesondere relevant für die Auswertung von Versionskontrollsystemen, da deren

historische Informationen üblicherweise nur über eine Log-Schnittstelle abgefragt werden, was relativ unkomfortabel für explorative Analysen ist.

Die Berechnung der Metriken muss weiterhin händisch implementiert werden, beispielsweise in Form der Metrik-Plug-Ins. Die direkte Definition von eigenen Metriken durch den Anwender ist nicht möglich. Zudem ist die detaillierte Definition einer Metrik nicht für den Anwender sichtbar.

3. Ziele

Denn nur aufs Ziel sehen
verdirbt die Lust am Reisen.

(Friedrich Rückert, 1788-1866)

Inhaltsangabe

Dieses Kapitel beschreibt die Ziele der Arbeit. Im folgenden Abschnitt wird zunächst die Ausgangslage dargestellt, und die gestellten Ziele werden motiviert. Die Ziele werden dann in Abschnitt 3.2 in Form von mehreren Fragestellungen präzisiert, die in dieser Arbeit zu beantworten sind. Schließlich folgt in Abschnitt 3.3 eine grobe Skizze des entwickelten Lösungsansatzes, um dem Leser die Orientierung in den folgenden Kapiteln zu erleichtern.

3.1. Motivation

Prozessbewertung und IT-Controlling zielen darauf ab, Transparenz über den Ablauf von Prozessen und den Status von Projekten zu schaffen. Wie in Abschnitt 2.3 dargestellt, gibt es in diesem Kontext eine ganze Reihe von Fragestellungen, die mit Hilfe von Informationen aus Software-Entwicklungsarchiven beantwortet werden können. Im Vergleich zu einer manuellen Erfassung von Statusinformationen zu Projekten und Produkten können bei einer Auswertung von Software-Entwicklungsarchiven zusätzlich auch Informationen aus der Vergangenheit berücksichtigt werden [CVW98]. Zudem wird die manuelle Berichterstattung von den dafür Verantwortlichen oft als aufwändig und lästig empfunden [CVW98]. Die manuell erfassten Daten sind typischerweise unvollständig und fehlerbehaftet [Zel07].

Somit stellt eine automatische Auswertung der ohnehin in den Software-Entwicklungsarchiven gesammelten Daten eine kostengünstige Alternative dar. Diese Daten werden allerdings nur unzureichend genutzt. Gängige Werkzeuge für das Issue-Tracking bieten üblicherweise eine Reihe von fest vorgegebenen Metrik-Auswertungen, bei denen nur einige wenige Parameter vom Benutzer angepasst werden können, wie das betrachtete Produkt oder der Zeitraum [GSL07b]. Um organisationsspezifische Informationsbedürfnisse zu erfüllen, müssen Metriken

eigens implementiert werden [KA08]. Weiterhin unterstützen gängige Werkzeuge nur die Auswertung einer einzelnen Informationsquelle, etwa ein bestimmtes Issue-Tracking-System. Verknüpfungen zwischen Einträgen in Issue-Tracking-Systemen und Konfigurationsmanagement-Systemen werden nicht berücksichtigt.

Ähnlich wie im Bereich der Quellcode-Vermessung liefern die verwendeten Messwerkzeuge abgesehen von vordefinierten Grenzwerten kaum Interpretationsunterstützung [RW05]. Ebenso gilt wiederum, dass die Definition, Implementierung und Validierung von Metriken arbeitsaufwändig ist [EGM+06].

Um die Metrikberechnung korrekt zu implementieren und die Ergebnisse interpretieren zu können, ist eine präzise Definition der Metrik notwendig [HDHM06]. In der Realität sind die Details einer Metrikdefinition für den Anwender der Metrik schwer nachzuvollziehen, da diese letztlich in der Implementierung verborgen sind. Dies erschwert zudem die Vergleichbarkeit von Metriken zwischen unterschiedlichen Messwerkzeugen.

Um organisationsspezifisch angepasste Metriken auf Software-Entwicklungsarchiven auszuwerten, müssen eigene Werkzeuge entwickelt bzw. organisationsspezifisch angepasst werden. Zudem müssen entsprechend spezialisierte Mitarbeiter verfügbar sein. Dieser Aufwand kann besonders in kleinen und mittelgroßen Organisationen oft nicht erbracht werden. Wenn überhaupt, dann werden nur solche Auswertungen betrachtet, die in den verwendeten Werkzeugen schon vorgegeben sind. Die Interpretation von Messwerten im Hinblick auf Qualitätsmerkmale des Prozesses auf einer höheren Ebene ist dann oft unklar.

Der Ist-Zustand im Bereich der Auswertung von Software-Entwicklungsarchiven ist also zusammenfassend charakterisiert durch Defizite im methodischen Vorgehen:

- Für die detaillierte Definition von Metriken und deren Erhebungsmethoden gibt es wenig Unterstützung.
- Es ist unklar, wie erhobene Messwerte zu interpretieren sind.

Weiterhin bestehen Defizite im Bereich der Werkzeugunterstützung:

- Die in einem Werkzeug vorgegebenen Metriken können vom Benutzer nur wenig angepasst werden.
- Die implementierten Metrikdefinitionen sind schwer nachvollziehbar.
- Werkzeuge sind auf ein isoliertes Software-Entwicklungsarchiv beschränkt.

Viele Daten im Entwicklungsprozess werden zwar routinemäßig automatisch erfasst. Allerdings können diese nicht geeignet ausgewertet werden. Mit wachsender Größe einer Organisation besteht dadurch die Gefahr einer unzureichenden Transparenz über die laufende Entwicklung. Es fehlen somit Informationen zur Steuerung und Verbesserung der Prozesse.

3.2. Fragestellungen

Aus den bereits genannten Schwierigkeiten ergeben sich die für diese Arbeit relevanten Fragestellungen:

A. **Wie werden geforderte Qualitätsmerkmale des Prozesses in Bezug gesetzt zu den Metriken?** Diese Frage motiviert sich aus der fehlenden Interpretationsunterstützung für die Messergebnisse. Wie in Kapitel 2 bereits eingeführt, sind die für eine Auswertung interessanten Qualitätsmerkmale von den Zielen der Organisation abhängig. Bei der Definition von entsprechenden Metriken muss zum einen der gelebte Prozess und zum anderen die in Software-Entwicklungsarchiven vorhandenen Daten berücksichtigt werden. Es muss geklärt werden, wie Messergebnisse im Hinblick auf ein Qualitätsmerkmal interpretiert werden. Weiterhin müssen die Messergebnisse zwischen verschiedenen Messobjekten vergleichbar sein. Es ist also eine konzeptuelle Basis nötig, um die Beziehung zwischen Qualitätsmerkmalen und zugeordneten Metriken zu beschreiben.

B. **Wie können geeignete Metriken kostengünstig entwickelt und validiert werden?** Ansätze wie GQM [BCR94] beschreiben eine allgemeine Vorgehensweise zur Entwicklung von Metriken. Für die detaillierte Definition von Metriken gibt es allerdings wenig Unterstützung [Sch07]. Zudem gibt es im Bereich der Auswertung von Software-Entwicklungsarchiven nach unserer Erfahrung eine Reihe von typischen Problemen bei der Definition von Metriken [GSL07a]. Aus diesem Grund ist ein systematisches Vorgehen zur Entwicklung von Metriken notwendig, welches auf eine präzise Definition der Metrik und deren Validierung abzielt. Dieses muss es auch für kleine und mittelgroße Organisationen ermöglichen, mit vertretbarem Aufwand spezifisch angepasste Metriken zu entwickeln.

C. **Wie können Software-Entwicklungsarchive auf flexible Weise ausgewertet werden?** Abgesehen von den bereits genannten methodischen Fragen muss auch geklärt werden, wie eine flexibel anwendbare Werkzeugunterstützung zur Auswertung von Software-Entwicklungsarchiven aussieht. Wie in Abschnitt 3.1 beschrieben, haben gängige Werkzeuge Schwächen hinsichtlich der Anpassbarkeit der Metriken, bei der Anbindung unterschiedlicher Datenquellen und bei der Nachvollziehbarkeit der Metrikdefinitionen. Um einen breit anwendbaren Ansatz zu bieten, muss die Werkzeugunterstützung unabhängig von dem konkret auszuwertenden System sein und eine Auswertung von gängigen Issue-Tracking- und Konfigurationsmanagement-Systemen unterstützen. Um organisationsspezifische Informationsbedürfnisse zu erfüllen, muss eine weites Spektrum von Metriken ausgewertet werden können. Zudem soll die Werkzeugunterstützung so flexibel sein, dass neue Metriken direkt durch den Anwender definiert werden können, ohne dass dazu die Implementierung des Werkzeugs verändert werden muss.

Die Betrachtung von Software-Entwicklungsarchiven möchten wir auf Issue-Tracking-Systeme und Konfigurationsmanagement-Systeme konzentrieren, da diese Systeme am häufigsten in der Praxis eingesetzt werden. Die verwalteten Änderungsanträge sind bei der Auswertung als der zentrale Ausgangspunkt zu sehen. Mit diesen wird der Fortschritt von konkreten Arbeitsaufträgen dokumentiert, sodass ein unmittelbarer Bezug zum Prozess besteht. Dies schließt nicht aus, dass die vorgestellten Ansätze auf andere Software-Entwicklungsarchive erweitert werden können, beispielsweise durch die Verknüpfung von Änderungsanträgen zu Testfällen und archivierten Testresultaten.

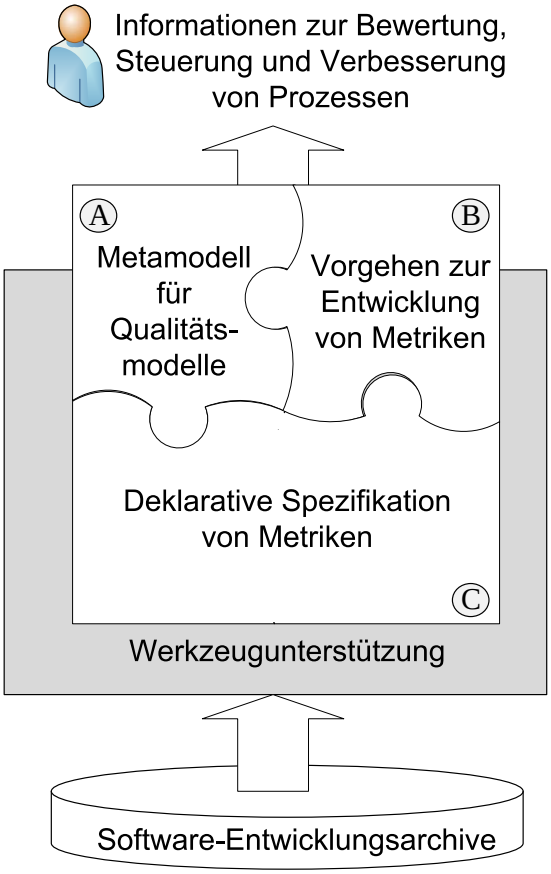


Abbildung 3.1.: Überblick über den entwickelten Lösungsansatz

3.3. Überblick über den entwickelten Lösungsansatz

Um eine Orientierung für die weiteren Kapitel zu geben, wird in diesem Abschnitt der entwickelte Lösungsansatz skizziert. Einen entsprechenden Überblick zeigt Abbildung 3.1. Ausgangspunkt sind die in Software-Entwicklungsarchiven gesammelten Daten über den Entwicklungsprozess. Die Abbildung zeigt die wesentlichen Elemente des Lösungsansatzes: Eine Sprache zur deklarativen Spezifikation von Metriken (Abschnitte 4.3 bis 4.6), ein entsprechendes Vorgehen zur

Entwicklung von Metriken (Kapitel 5), sowie ein Metamodell für Qualitätsmodelle (Kapitel 6). Alle diese Elemente des Lösungsansatzes werden durch Werkzeuge unterstützt (Kapitel 7). Die Buchstaben im Diagramm zeigen den Zusammenhang mit den im vorherigen Abschnitt aufgelisteten Fragestellungen.

Mit der im nächsten Kapitel eingeführten deklarative Sprache können Metriken auf Software-Entwicklungsarchiven in kompakter und präziser Form beschrieben werden. Die zugehörige Werkzeugunterstützung ermöglicht eine einfache Anbindung an unterschiedliche Datenquellen. Die deklarative Sprache zusammen mit der Werkzeugunterstützung zielt auf die flexible Auswertung von Software-Entwicklungsarchiven ab.

Weiterhin kann mit diesen Hilfsmitteln eine Metrik schnell angepasst werden, und die Ergebnisse sind direkt verfügbar. Dies ermöglicht ein schnelles, iteratives Vorgehen zur Entwicklung und Validierung von Metriken.

Um schließlich die Beziehung zwischen Qualitätsmerkmalen des Prozesses und den Metriken zu modellieren, werden benutzerdefinierte Qualitätsmodelle eingesetzt. Das eingeführte Metamodell für Qualitätsmodelle gibt die Struktur und die Elemente dieser Qualitätsmodelle vor. Dabei sind Flexibilität und Verständlichkeit der eingeführten Modellelemente wichtig. Weiterhin wird vorgestellt, wie in diesem Zusammenhang empirische Vergleichsdaten herangezogen werden, um eine realistische Einordnung und Bewertung von Messergebnissen vorzunehmen.

In Abschnitt 8.1 wird schließlich gezeigt, wie der vorgestellte Lösungsansatz für die Bewertung von Prozessen eingesetzt wird. Die gezeigten Fallbeispiele beziehen sich zum einen auf die industrielle Softwareentwicklung, zum anderen auf den *Open Source* Bereich. Eine Evaluierung der entwickelten Lösungen im Hinblick auf die in diesem Abschnitt formulierten Ziele folgt dann in den Abschnitten 8.2 bis 8.5.

4. Deklarative Spezifikation von Metriken

> Everything is vague to a degree you do not realize till you have tried to make it precise.
>
> *(Betrand Russell, 1872-1970)*

Inhaltsangabe

In diesem Kapitel wird eine deklarative Sprache zur Spezifikation von Metriken auf Software-Entwicklungsarchiven vorgestellt. Zunächst werden existierende Möglichkeiten zur präzisen Beschreibung von Metriken diskutiert. Dabei betrachten wir auch Ansätze in anderen Anwendungsbereichen, wie zum Beispiel die Vermessung von Quellcode.

4.1. Motivation

Roche nennt als ein Grundprinzip für die Verwendung von Metriken, dass jede Metrik klar und eindeutig definiert werden muss [Roc94]. Typischerweise werden Metriken definiert mit Hilfe von strukturierten natürlichsprachlichen Beschreibungen und mathematischen Formeln zur Darstellung von Berechnungsschritten. Beispiele dazu finden sich etwa in den Teilen 2-4 von ISO/IEC 9126 und in den von Kütz beschriebenen Kennzahlenkatalogen [Küt06].

In der Dokumentation zu Code-Metrik-Werkzeugen werden ebenfalls strukturierte natürlichsprachliche Beschreibungen verwendet (z.B. [IBM08]). Die eigentliche Definition der Metriken steckt in der Implementierung des Werkzeugs. Die genaue Metrikdefinition ist somit entweder nicht verfügbar oder sie befindet sich auf einem niedrigen Abstraktionsniveau.

In der Praxis ergeben sich allerdings Probleme in Bezug auf die Methodik und die Werkzeugunterstützung. So wird in den Erläuterungen zum Modell CMMI darauf hingewiesen, dass viele Organisationen Messwerte nicht interpretieren

können auf Grund von Inkonsistenzen bei der Definition der Metriken und deren Erhebung [CMM06].

Die Vergleichbarkeit von Metriken ist selbst im Bereich der „klassischen“ Code-Metriken auf Grund von ungenauen Definitionen und Schwächen der Werkzeugunterstützung nicht gegeben. Lincke et al. stellten in einer vergleichenden Untersuchung von verschiedenen Code-Metrik Werkzeugen fest, dass es beispielsweise bei den Metriken *Coupling between object classes (CBO)* und *Lack of cohesion of methods (LCOM)* [CK94] deutliche Abweichungen bei den ermittelten Ergebnissen gab [LLL08]. Unterschiede waren beispielsweise zurückzuführen auf das Einschließen oder Ausschließen von vererbten Elementen, die unterschiedliche Berücksichtigung von Elementen aus Bibliotheksklassen, oder die unterschiedliche Behandlung von sprachspezifischen Konstrukten. Diese Ergebnisse wurden in einer Studie von Breuker et al. bestätigt [BBDA09]. Breuker führt dies darauf zurück, dass die Definitionen der Metriken in der Literatur zu abstrakt oder zu vage sind. Zudem ist bei vielen Werkzeugen nicht ersichtlich, welche Definition einer Metrik implementiert wird.

Es ist also notwendig, Metriken präziser zu beschreiben. Ein Mittel dazu ist eine formale Sprache. Eine formale Sprache allein ist allerdings noch nicht ausreichend. Messerschmidt und Schweinsberg schildern in einer Fallstudie Probleme beim Einsatz von SQL-Abfragen zur Ermittlung von Metriken auf einer Datenbank mit Problemberichten [MS03]. Unter anderem ist ein Verständnis eines relativ komplexen Datenbankschemas nötig. Daraus ergeben sich ein hoher Arbeitsaufwand und die Gefahr von Fehlern in den erzeugten SQL-Abfragen. Zudem fehlen einheitliche, leicht verständliche Definitionen der relevanten Kennzahlen.

Daraus ergibt sich, dass die Sprache zur Beschreibung der Metriken ein höheres Abstraktionsniveau besitzen muss, um eine vollständige, präzise Definition einer Metrik in kompakter Form zu ermöglichen. Die Metrik-Spezifikation selbst muss lesbar und leicht verständlich sein, im Sinne einer deklarativen Beschreibung.

Im folgenden Abschnitt werden zunächst bestehende Ansätze für formale Beschreibungssprachen zur Spezifikation von Metriken vorgestellt. In Abschnitt 4.3 wird dann die Sprache ITMS zur Spezifikation von Metriken auf Issue-Tracking-Systemen eingeführt. Die Abschnitte 4.4 und 4.5 beschreiben die an ITMS gestellten Anforderungen, sowie Aufbau und Syntax der Sprache. Um die Semantik der Sprache unabhängig von einer Referenzimplementierung zu halten, wird in Abschnitt 4.6 eine formale Semantik von ITMS vorgestellt.

Ein Vorgehen für die Entwicklung und Validierung von Metrik-Spezifikationen in ITMS wird in Kapitel 5 beschrieben. Eine Referenzimplementierung für die Auswertung von ITMS Metrik-Spezifikationen wird dann in Kapitel 7 vorgestellt.

4.2. Verwandte Arbeiten

Bei der Verwendung einer formalen Sprache zur Beschreibung von Metriken, muss betrachtet werden, wie das Messobjekt selbst formalisiert ist.

Im Bereich der Vermessung von Quellcode oder Architekturen ist es naheliegend das Messobjekt auf Graphstrukturen zurückzuführen. Reißing stellt beispielsweise ein sprachunabhängiges Metamodell für objektorientierte Programme vor, welches auf Graphen basiert [Rei01]. Metriken können dann als mathematische Berechnungen auf dem Graphen formalisiert werden. Somit steht eine einfache und klare Beschreibung von Metriken zur Verfügung. Mens und Lanza verfolgen einen ähnlichen Ansatz [ML02]. Bei beiden Ansätzen erfolgt die eigentliche Implementierung von Metriken allerdings in einer Standard-Programmiersprache.

Im Folgenden werden Ansätze für formale Sprachen zur Spezifikation von Metriken vorgestellt, bei denen die Metrik-Spezifikation selbst ausführbar ist. Die Ansätze sind nach der technischen Herangehensweise gegliedert.

4.2.1. Attributgrammatiken

Cogan und Hunter beschreiben den Einsatz von Attributgrammatiken [Knu68] zur Ermittlung von Quellcode-Metriken [CH96]. Falls mehrere Durchläufe über den Quellcode zur Ermittlung der Metrik erforderlich sind, stößt dieser Ansatz wegen Länge und Komplexität der dazu nötigen Attributgrammatiken an seine Grenzen. Im Kontext von Prozessmetriken ist ein solcher Ansatz nicht geeignet.

4.2.2. Abfragesprachen für relationale Datenbanken

Ein häufig verwendeter Ansatz ist, das Messobjekt in eine andere Formalisierung zu überführen. Diese bildet dann eine Abstraktionsschicht, auf der die Metriken beschrieben werden können.

Im Ansatz von Scotto et al. wird eine relationale Datenbank als Abstraktionsschicht zur Berechnung von Metriken auf Quellcode verwendet [SSSV04]. Der Quellcode wird geparst und Beziehungen zwischen Quellcode-Elementen in eine relationalen Datenbank importiert. Eine Metrik wird dann in Form einer SQL-Abfrage spezifiziert. Die Metrik-Spezifikation hat dadurch ein höheres Abstraktionsniveau. Die Überführung in die relationale Datenbank als Abstraktionsschicht ist allerdings mit einem Informationsverlust verbunden und begrenzt somit die Anzahl der abfragbaren Metriken.

4.2.3. Abfragesprachen für OLAP-Datenbanken

OLAP (*Online Analytical Processing*) ist ein Ansatz zur Analyse und Auswertung von multidimensional aufbereiteten Daten. Dieser Ansatz lässt sich ebenfalls für Metrik-Abfragen einsetzen [MS03]. Im Folgenden werden kurz einige Grundbegriffe aus dem Bereich des *Data Mining* eingeführt, in den sich OLAP einordnen lässt. Dann wird der Einsatz von OLAP für Auswertungen auf einem Issue-Tracking-System vorgestellt.

Vereinfacht gesagt, bezeichnet der Begriff **Data Mining** die Gewinnung von Wissen aus großen Datenmengen [HK06]. Unter Wissen sind hier für Entscheidungen oder Aufgaben eines Anwenders relevante Informationen zu verstehen. Ein **Data Warehouse** enthält nach einem einheitlichen Schema strukturierte Daten, die durch Aufbereitung von Daten aus mehreren, möglicherweise heterogenen Datenbanken gewonnen wurden. Das Data Warehouse unterstützt Aufgaben des Berichtswesens und der Entscheidungsunterstützung. Dazu werden die Daten aus operativen Datenbeständen aufbereitet und in ein einheitliches Schema überführt. Dieser Vorgang wird auch als ETL-Prozess bezeichnet (*Extract, Transform, Load*). Typischerweise werden die Daten für die späteren Auswertungen **multidimensional** aufbereitet. Beispielsweise können die Verkäufe einer Handelskette aufbereitet werden nach den Dimensionen Produkt, Standort und Verkaufsdatum. Eine Dimension kann auch durch Kombination mehrerer Attribute gebildet werden. Diese Aufbereitung ermöglicht bei Abfragen jeweils genau solche Datensätze zu ermitteln, die Bedingungen an die Belegung der Attribute der verschiedenen Dimensionen entsprechen. Eine OLAP-Anwendung ermöglicht es dem Anwender, auf flexible Weise Ad-hoc-Abfragen auf einem *Data Warehouse* durchzuführen. Wichtig ist dabei eine schnelle Antwortzeit, um die Anfrage schrittweise verfeinern zu können. In OLAP-Systemen werden eigene Abfragesprachen eingesetzt. Eine von Microsoft eingeführte und inzwischen auch von anderen Anbietern eingesetzte Sprache ist MDX (*Multi-Dimensional Expressions*) [WZP05].

Messerschmidt und Schweinsberg beschreiben in einer Fallstudie eine OLAP-Anwendung für Metrik-Auswertungen auf einem Issue-Tracking-System [MS03]. Dieses System speichert den aktuellen Zustand aller bisher eingegangen Problemberichte in einer Tabelle, und hält ein Protokoll der Zustandsänderungen von Problemberichten in einer weiteren Tabelle vor. Diese Daten müssen kopiert und geeignet aufbereitet werden. So werden etwa die historisierten Zustandsänderungen in eine Tabelle überführt, die beschreibt, von welchem Start- bis zu welchem Enddatum ein Problembericht in einem bestimmten Status verweilte. Als Ergebnis sind die Problemberichte beispielsweise aufbereitet in Dimensionen wie:

- das System oder Systemteil, welches von dem berichteten Problem betroffen ist
- Datum, an dem der Problembericht erstellt wurde (Erstellungsdatum)

- Datum, an dem der Problembericht geschlossen wurde (Schließungsdatum)
- Schweregrad des Problemberichts

Als weiterer Schritt der Aufbereitung werden vorab für die verschiedenen Dimensionen Kennzahlen ermittelt, wie die Anzahl der Problemberichte, die Differenz zwischen Erstellungszeitpunkt und Lösungszeitpunkt oder die durchschnittliche Bearbeitungsdauer.

Auf Basis der aufbereiteten Daten können dann Metriken bestimmt werden, wie etwa die Anzahl der geschlossenen Probleme pro Kalenderwoche und Komponente des Systems. Die Ergebnisse lassen sich beispielsweise weiter aufgliedern nach der Woche, in der ein Problembericht geöffnet wurde.

Mit Hilfe der bereits im ETL-Prozess erfolgten Datenaufbereitung können Abfragen interaktiv durchgeführt werden. Die Menge der auswertbaren Metriken ist allerdings auch durch die Form der Aufbereitung beschränkt. Komplexere Abfragen können es erforderlich machen, die Skripte für den ETL-Prozess zu ergänzen. Insbesondere wenn Zusammenhänge zwischen Ereignissen im Lebenslauf eines Problemberichts oder Rückbezüge auf frühere Ereignisse betrachtet werden, kann dies nur durch die Definition und Berechnung geeigneter Kennzahlen im Rahmen des ETL-Prozesses erreicht werden. Somit ist dieser Ansatz zugeschnitten für Ad-hoc-Abfragen, bei denen vorab eingegrenzt werden kann, auf welche Zustandswechsel oder Attribute eines Problemberichts sie sich typischerweise beziehen.

Der gleiche technischen Ansatz wird bei dem Open Source Projekt *Software Quality Reports for Jira/Bugzilla* verfolgt [SQR], welches auf der Business Intelligence Software *Pentaho* basiert.

4.2.4. Weitere Abfragesprachen

Neben SQL und OLAP-Abfragesprachen finden sich noch eine Reihe von weiteren Abfragesprachen. Simon und Lewerentz stellen einen Ansatz zur Ermittlung von Metriken auf objektorientiertem Code vor, bei dem eine SQL-ähnliche Abfragesprache zur Definition von Metriken auf einem E/R-Schema verwendet wird, in dem Eigenschaften und Beziehungen zwischen Klassen, Methoden und Attributen erfasst werden [SL97].

McQuillan und Power beschreiben ein MOF-konformes Metamodell [MP08], welches ebenfalls dazu dient, eine sprachunabhängige Abstraktionsschicht zur Definition von Metriken zu schaffen. Quellcode-Metriken werden durch OCL-Abfragen auf dem Metamodell beschrieben [OMG06]. Auch Baroni et al. verwenden OCL zur Spezifikation von Metriken für Datenbank-Schemata. [BCBP06].

Weitere Ansätze verwenden die Sprache *XQuery* zur Spezifikation von Metriken [EEBF05, EGM+06]. *XQuery* ist eine Abfrage-Sprache für XML-Daten, die auch Elemente einer funktionalen Sprache besitzt [W3C07].

4.2.5. Modell-zu-Modell Transformationsprachen

Im Rahmen des Eclipse Projekts *MoDisco* [MoD], welches eine Infrastruktur für die Extraktion von Modellen aus *Legacy*-Anwendungen bereitstellen soll (*Model Driven Reverse Engineering*), wird eine Anwendung zur Ermittlung von Metriken aus dem Issue-Tracking-System Bugzilla vorgestellt [Bru09]. Metriken werden dabei mit der Modell-zu-Modell Transformationssprache ATL beschrieben [ATL]. Das Ziel-Modell entspricht einer tabellarischen Darstellung von Metrik-Ergebnissen. Dieser Anwendungsfall wird jedoch nur anhand der einzigen Metrik „Anzahl Problemberichte nach Schweregrad“ demonstriert. Auch für diese einfache Metrik ergibt sich bereits ein relativ langes ATL-Skript. Offensichtlich dient dies nur als ein Anwendungsbeispiel für den Einsatz von *MoDisco*.

4.2.6. Domänenspezifische Sprachen

In den im Folgenden genannten Ansätzen wird nicht eine bekannte Programmiersprache verwendet, sondern eine eigene Sprache für die Metrik-Spezifikationen vorgegeben.

Marinescu et al. stellen die domänenspezifische Sprache SAIL vor, um Design-Metriken für objektorientierte Entwürfe zu spezifizieren [MMG05]. SAIL bietet Elemente einer prozeduralen Sprache, in die SQL-ähnliche Abfragen eingebettet werden können. Dies ermöglicht eine prägnantere Beschreibung von Metriken, als in einer reinen Abfragesprache oder prozeduralen Sprache.

García et al. stellen ein MOF-konformes Metamodell zur Beschreibung von sogenannten *Software Measurement Models* vor [GSCL+07]. Ein solches *Software Measurement Model* kann dann spezifisch für eine Domäne definiert werden und spezifiziert die Berechnung von Metriken. Als Beispiele werden die Vermessung von E/R-Modellen und die Vermessung von relationalen Datenbank-Schemata beschrieben. In dem Beitrag wird ebenfalls eine Werkzeugunterstützung zur Spezifikation von Metriken mit Hilfe einer graphischen Benutzeroberfläche und deren automatische Berechnung vorgestellt. Sowohl die Messobjekte als auch die Metriken müssen dabei im Format XMI vorliegen, einer XML-basierten Sprache zur Beschreibung von MOF-konformen Modellen und Metamodellen [OMG07].

Monperrus et al. beschreiben einen Ansatz zur deklarativen Spezifikation von Metriken für die Vermessung von Modellen in der modellgetriebenen Entwicklung [MJCH08]. Das vorgeschlagene Metamodell für die Metrik-Spezifikationen

ist domänenunabhängig. Mögliche Elemente einer Metrik-Spezifikation sind beispielsweise das Zählen von Modellelementen, das Zählen von Links zwischen Modellelementen oder das Bestimmen der Länge eines Pfades im Modell. Die Metrikberechnungen lassen sich also wiederum auf Graphen zurückführen. In dem Ansatz wird eine textliche deklarative Sprache zur Beschreibung von konkreten Metriken verwendet. Das Messwerkzeug wird auf Basis der Metrik-Spezifikation generiert.

Die beiden vorgenannten Ansätze sind ähnlich. Beide führen die Metrikberechnungen auf Graphen zurück. Im Ansatz von García et al. wird nur das Zählen von Modellelementen und die Bestimmung von Pfadlängen in einem Graph als Messmethoden diskutiert. Monperrus et al. liefern eine genauere Analyse der typischen Elemente einer Metrik-Spezifikation.

Allgemein ist die Ausdruckstärke durch die zu Grunde liegende Abstraktion beschränkt. So ist es beispielsweise mit der von Monperrus et al. vorgestellten deklarativen Sprache nicht möglich, die Metrik *Lack of Cohesion in Methods* zu definieren [CK94]. Diese müsste wiederum händisch auf Basis des Java-Metamodells implementiert werden.

4.2.7. Diskussion

Zusammenfassend lässt sich feststellen, dass die meisten der vorgestellten Ansätze auf die Vermessung von Quellcode oder Entwurfsmodellen abzielen. Das Messobjekt wird für die Vermessung oft erst in ein sprachunabhängiges Format überführt, um so eine Abstraktionsschicht zu schaffen, auf der die Metriken formalisiert werden.

Es lassen sich dann zwei unterschiedliche Vorgehen unterscheiden. Zum einen werden gängige Abfragesprachen zur Spezifikation von Metriken verwendet (z.B. SQL, OCL, MDX, XQuery). Zum anderen werden eigene Sprachen bzw. Metamodelle zur Spezifikation von Metriken eingeführt.

Eine eigene domänenspezifische Sprache ermöglicht eine kompaktere und prägnante Beschreibung der Metrik. Bei einer existierenden Sprache besteht eine bessere Werkzeugunterstützung und der Einarbeitungsaufwand ist möglicherweise geringer. In den vorherigen Abschnitten wurde deutlich, dass die Ausdrucksstärke der Metrik-Spezifikationen grundsätzlich durch die zu Grunde liegende Abstraktion beschränkt ist.

Bei nur wenigen der vorgestellten Ansätze wird über Erfahrungen aus einen Einsatz in der Praxis berichtet [MJCH08, MS03].

4.3. ITMS: Eine Sprache zur Spezifikation von Metriken

Wie im vorherigen Abschnitt erkennbar wurde, lassen sich Strukturen wie der Kontrollfluss im Quellcode oder Abhängigkeiten in einer Architektur gut als Graph formalisieren. Die in einem Issue-Tracking-System gesammelten Daten sind dazu weniger geeignet, da hier auch die Zeiträume zwischen den verschiedenen Ereignissen eine Rolle spielen.

Viele der im vorherigen Abschnitt beschriebenen Ansätze basieren auf einer Transformation des Messobjekts in ein anderes Format. Beispiele dafür sind der Import in eine Datenbank, oder die Transformation in ein Modell, auf welchem die Metriken spezifiziert sind. Eine solche Transformation kann zu einem Informationsverlust führen und schränkt somit ein, welche Metriken berechnet werden können. In unserem Ansatz wird auf eine solche Transformation verzichtet, sodass potentiell alle im Issue-Tracking-System verfügbaren Daten für die Auswertung bereit stehen.

Des Weiteren ermöglicht eine domänenspezifische Sprache ein höheres Abstraktionsniveau der Metrik-Spezifikationen, als es mit einer allgemeinen Programmiersprache oder Abfragesprache möglich ist (vgl. Abschnitt 4.2.6).

Diese Überlegungen bilden den Rahmen für die Entwicklung einer domänenspezifischen Sprache zur Spezifikation von Metriken auf Issue-Tracking-Systemen. Diese Sprache wird im Folgenden als ITMS (*Issue Tracking Metric Specification Language*) bezeichnet.

Die Anforderungen an die Sprache ITMS sind im nächsten Abschnitt dargestellt. Abschnitt 4.5 beschreibt den Aufbau und die Syntax von ITMS. Abschnitt 4.6 enthält eine formale Beschreibung der Semantik von ITMS.

4.4. Anforderungen an die Sprache ITMS

Ausgangspunkte der Anforderungsanalyse für ITMS waren zum einen die Betrachtung von relevanten Metriken im Bereich der Prozessbewertung und zum anderen die Analyse der Möglichkeiten zur Spezifikation von Metrikauswertungen in existierenden Werkzeugen [Gra07]. Auf diesem Weg wurden zunächst typische Elemente bei der Spezifikation von Metriken identifiziert. Dazu gehören beispielsweise Filter für die Auswahl der betrachteten Änderungsanträge, als auch verschiedene Operationen bei der Berechnung von Metriken. Auf Basis dieser typischen Elemente konnte schließlich eine gemeinsame Grundstruktur für Metrik-Spezifikationen definiert werden.

Im folgenden Abschnitt werden zunächst grundlegende Begriffe eingeführt. In Abschnitt 4.4.2 werden dann die einzelnen Bestandteile einer Metrik-Spezifikation in ITMS beschrieben. Ein Überblick über den für alle Metriken in ITMS gemeinsamen Ablauf von Berechnungen folgt in Abschnitt 4.4.4. Schließlich werden in

Abschnitt 4.4.5 die Details für die sogenannten Bewertungsverfahren beschrieben, welche den Hauptteil einer Metrik-Spezifikation bilden.

4.4.1. Grundbegriffe

4.4.1.1. Attribute

Grundlage für die Berechnung einer Metrik ist die Datenbank der Änderungsanträge. Änderungsanträge haben **Attribute** wie Status, Beschreibung und Priorität. Die Werte aller Attribute bestimmen den **Zustand** eines Änderungsantrags. Für die meisten Attribute wird die Änderungshistorie im Issue-Tracking-System gespeichert.

Attribute können in direkte und abgeleitete Attribute unterschieden werden. Als **direkte Attribute** bezeichnen wir solche Attribute, die unmittelbar in der Datenbank verfügbar sind. **Abgeleitete Attribute** können aus direkten Attributen berechnet werden. Der Fertigstellungsgrad eines Änderungsantrags wird zum Beispiel aus dem aktuell geschätzten Gesamtaufwand und den bisher geleisteten Arbeitsstunden berechnet.

Des Weiteren lässt sich zwischen **einwertigen** und **mehrwertigen** Attributen unterscheiden. Normalerweise hat ein Attribut zu einem Zeitpunkt genau einen Wert. Bestimmte Attribute können mehrere Werte zu einem Zeitpunkt besitzen. Ein Beispiel ist das Attribut „Kommentar“, da für einen Änderungsantrag alle bisher eingetragenen Kommentare gültig sind. Bei solchen Attributen kann in einer Metrik-Spezifikation unterschieden werden, ob jeweils nur der letzte Wert, oder auch alle bisherigen Werte des Attributs betrachtet werden sollen.

4.4.1.2. Zustandsfilter

Ein generelles Konzept bei der Spezifikation von Metriken sind Filter. Mit einem Filter werden Bedingungen an die betrachteten Entitäten beschrieben. In der Sprache ITMS werden zwei Arten von Filtern verwendet - Zustandsfilter und Ereignisfilter.

Ein **Zustandsfilter** spezifiziert Bedingungen an den Zustand eines Änderungsantrags. Der Zustand ist durch die Belegung der Attribute des Änderungsantrags gegeben. Beispiele für solche Zustandsfilter sind das Filtern nach Änderungsanträgen, die einem bestimmten Produkt zugeordnet sind, oder das Filtern nach Änderungsanträgen, die sich in einem bestimmten Bearbeitungszustand befinden.

In ITMS können zur Beschreibung von Bedingungen an die Attributbelegung auch reguläre Ausdrücke verwendet werden. Weiterhin können Zustandsfilter mit den logischen Operatoren *nicht*, *und* und *oder* verknüpft werden.

4.4.1.3. Ereignisfilter

Mit einem **Ereignisfilter** wird spezifiziert, welche Ereignisse im Lebenslauf eines Änderungsantrags bei der Berechnung einer Metrik berücksichtigt werden. Typische Ereignisse sind:

- **Erzeugung des Änderungsantrags**: Dieses Ereignis tritt zu dem Zeitpunkt auf, an dem der Änderungsantrag erstellt wird.
- **Änderung eines Attributwerts**: Dieses Ereignis beschreibt die Änderung der Belegung eines Attributs. Entsprechende Ereignisfilter können für alle Attribute formuliert werden, für die historische Werte verfügbar sind. Zusätzlich kann eingeschränkt werden, welche Werte das Attribut vor oder nach der Änderung hatte. Ein Beispiel ist ein Ereignisfilter für den Übergang des Attributs „Status“ vom Attributwert „Neu“ in einen der Attributwerte „Bestätigt“ oder „Abgelehnt“. Es können wiederum reguläre Ausdrücke verwendet werden, um die zulässigen Attributwerte zu beschreiben.
- **Ende einer Zeitperiode**: Als ein Ereignis kann ebenfalls das Ende einer Zeitperiode angesehen werden. Die Zeitperioden ergeben sich aus der in der Metrik-Spezifikation spezifizierten Aufteilung des Berechnungszeitraums, zum Beispiel in Monate (siehe Abschnitt 4.4.2.1). Mit Hilfe eines entsprechenden Ereignisfilters lassen sich also regelmäßig am Ende jeder Zeitperiode Zählungen vornehmen (vgl. Abschnitt 4.4.2.4).

Es ist nicht möglich, an dieser Stelle eine erschöpfende Liste aller in Frage kommenden Ereignisse zu geben. Je nach Art des zu Grunde liegenden Software-Entwicklungsarchivs können weitere Ereignisse in Auswertungen relevant sein (z.B. das Überschreiten einer Deadline für die Fertigstellung eines Änderungsantrags). Deshalb muss ITMS um zusätzliche Ereignisfilter erweiterbar sein. Weiterhin lassen sich Ereignisfilter mit den logischen Operatoren *und* und *oder* verknüpfen.

4.4.1.4. Zeitreihen

Im Fokus der Sprache ITMS stehen zeitliche Auswertungen über die Vergangenheit. Das Ergebnis einer Auswertung ist eine Zeitreihe. Der Begriff Zeitreihe sei hier folgendermaßen definiert [SS01]:

> Eine (zeitlich) geordnete Folge von Beobachtungen einer Größe wird als **Zeitreihe** bezeichnet. Für jeden Beobachtungszeitpunktliegt dabei genau eine Beobachtung vor.

Die Ergebnisse der bei ITMS betrachteten Metriken sind entweder absolute Zahlen, Verhältnisse oder auch Verweildauern. Somit sind die Beobachtungen in einer Zeitreihe immer reele Zahlen.

4.4.2. Elemente einer Metrik-Spezifikation

In den folgenden Abschnitten sind die Elemente einer Metrik-Spezifikation dargestellt. Diese müssen in der Sprache ITMS beschrieben werden können.

4.4.2.1. Berechnungszeitraum und Zeitgranularität

Bei einer Metrikberechnung wird jeweils ein bestimmter Zeitraum betrachtet, der in Zeitperioden untergliedert wird. Dazu müssen die folgenden Informationen in einer Metrik-Spezifikation enthalten sein.

Der **Berechnungszeitraum** gibt an, zwischen welchem Start- und Endzeitpunkt eine Zeitreihe berechnet wird. Die **Zeitgranularität** gibt an, wie der Berechnungszeitraum untergliedert wird. Als Granularität können sowohl Zeitperioden wie Woche, Monat oder Jahr, als auch benutzerdefinierte Zeitperioden verwendet werden. Letztere können sich beispielsweise an den Release-Zyklen eines Produkts orientieren. Abbildung 4.1 verdeutlicht die Verwendung der Begriffe.

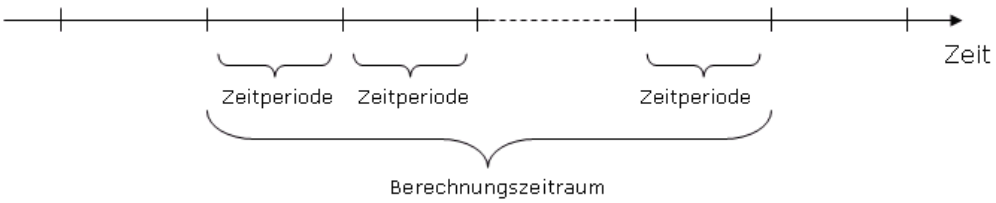


Abbildung 4.1.: Die Begriffe Zeitperiode und Berechnungszeitraum

4.4.2.2. Basisfilter

In einer Metrik-Spezifikation kann eingegrenzt werden, welche Änderungsanträge aus der Datenbank betrachtet werden. Beispielsweise können die Änderungsanträge für ein bestimmtes Produkt betrachtet werden, oder die Änderungsanträge, die einem bestimmten Meilenstein zugeordnet sind. Es werden also nur solche Änderungsanträge betrachtet, die bestimmte Bedingungen erfüllen. Diese Bedingungen beziehen sich auf die Belegung der Attribute. Wir bezeichnen diese Bedingungen als **Basisfilter**. Der Basisfilter wird durch Angabe eines Zustandsfilters spezifiziert (siehe Abschnitt 4.4.1.2).

4.4.2.3. Gruppierung

Das Ergebnis einer Metrikberechnung wird oft untergliedert, z.B. nach den verschiedenen Produkten in einem Produktportfolio, nach dem Status der Änderungsanträge oder nach den einzelnen Komponenten eines Produkts. Eine solche

Gruppierung muss in der Metrik-Spezifikation festgelegt werden können. Es kann jeweils nach einem oder mehreren Attributen der Änderungsanträge gruppiert werden. Das Ergebnis sind mehrere Zeitreihen, jeweils für jede mögliche Belegung der Attribute, nach denen gruppiert wird.

4.4.2.4. Bewertungsverfahren

Eine Hauptfrage bei der Entwicklung der Sprache ITMS ist, auf welche grundlegenden Berechnungen die Ermittlung von Metriken auf Issue-Tracking-Systemen zurückgeführt werden kann.

Ausgangspunkt bei der Identifikation dieser Berechnungen war eine Liste von Metriken, die für Fragestellungen in einem konkreten Kontext entwickelt wurden. Beispiele für diese Metriken werden in Abschnitt 4.4.5 beschrieben.

Das Ergebnis der betrachteten Metriken ist jeweils immer eine Zeitreihe. Daher muss betrachtet werden, wie eine einzelne Beobachtung in der Zeitreihe ermittelt wird. Eine solche Beobachtung lässt sich auf Berechnungen für einzelne Änderungsanträge zurückführen.

Auf Grund dieser Überlegung wurden vier verschiedene Kategorien von grundlegenden Berechnungen identifiziert, die wir als **Bewertungsverfahren** bezeichnen. Bei einem Bewertungsverfahren wird jeweils der Lebenslauf eines einzelnen Änderungsantrags betrachtet, und zu bestimmten Zeitpunkten werden sogenannte Bewertungszahlen ermittelt. Diese Bewertungszahlen fließen schließlich in eine Beobachtung innerhalb der Zeitreihe ein (siehe Abschnitt 4.4.2.5).

Die folgenden Kategorien von Bewertungsverfahren stehen zur Verfügung:

- **Zählen von Ereignissen in einer Zeitperiode**: Immer dann, wenn im Lebenslauf eines Änderungsantrags ein bestimmtes Ereignis auftritt, wird eine entsprechende Bewertung erzeugt.
- **Zählen von Ereignissen bis zu einem Ereignis**: Das Auftreten eines Ereignisses A wird über Zeitperioden hinweg gezählt. Eine Bewertung wird erst zu dem Zeitpunkt des Auftretens eines anderen definierten Ereignisses B vorgenommen.
- **Intervalllänge**: Die Zeitdauer zwischen einem Startereignis und einem Endereignis im Lebenslauf eines Änderungsantrags wird bestimmt. Eine Bewertung wird beim Auftreten des Endereignisses erstellt.
- **Verweilzeit**: Die Verweilzeiten in einem bestimmten Zustand werden über den Lebenslauf des Änderungsantrags hinweg aufaddiert. Eine Bewertung wird beim Auftreten eines weiteren spezifizierten Ereignisses erstellt.

Über die hier gegebene Kurzbeschreibung hinaus kann jedes dieser Bewertungsverfahren durch bestimmte Parameter angepasst werden, wie beispielsweise Ereignisfilter.

Details zu den vier Bewertungsverfahren folgen in Abschnitt 4.4.5. Zunächst werden die weiteren Elemente einer Metrik-Spezifikation vorgestellt, um dann in Abschnitt 4.4.4 einen groben Überblick über den Ablauf der Metrik-Berechnungen zu geben.

4.4.2.5. Gruppenwertberechnung

Durch Anwendung eines der vorgenannten Bewertungsverfahren werden in jeder Zeitperiode des Berechnungszeitraums Änderungsanträgen keine, eine oder mehrere Bewertungen zugeordnet. Aus diesen Bewertungen muss ein einzelnes Resultat ermittelt werden. Dieses Resultat ist dann eine einzelne Beobachtung in der Ergebniszeitreihe. Dazu wird eine **Gruppenwertberechnung** verwendet. Für jede Zeitperiode wird die Menge der erstellten Bewertungen betrachtet, und aus dieser Menge ein einzelnes Resultat bestimmt. Beispiele für die dazu verwendeten Operationen sind:

- **Anzahl** der Bewertungszahlen
- **Summe** der Bewertungszahlen
- **Maximum/Minimum** der Bewertungszahlen
- **Median** aus der Menge der Bewertungszahlen

Die Sprache ITMS muss um weitere Gruppenwertberechnungen erweiterbar sein.

Das Resultat einer Gruppenwertberechnung ist eine Zeitreihe, die für jede Zeitperiode im Berechnungszeitraum eine reelle Zahl enthält. Zeitreihen aus unterschiedlichen Gruppenwertberechnungen können mit den arithmetischen Operationen Addition, Subtraktion, Division, Multiplikation verknüpft werden, und auch unter Verwendung von Konstanten umgerechnet werden. Somit entspricht die Gruppenwertberechnung dem Konzept einer abgeleiteten Metrik [ISO07b].

4.4.2.6. Fixierte Attribute

Als **Fixierte Attribute** können solche Attribute bestimmt werden, bei denen die früheren Belegungen des Attributs nicht berücksichtigt werden sollen, sondern nur der aktuelle Wert betrachtet wird. Der Grund dafür ergibt sich daraus, dass bei der initialen Klassifikation eines Änderungsantrags teilweise falsche Attributwerte gesetzt werden, und diese dann später korrigiert werden. Deshalb kann es sinnvoll sein, immer die aktuelle Belegung eines Attributs zu betrachten.

Als Beispiel für den Einsatz eines fixierten Attributs betrachten wir eine Metrik, bei der eine Gewichtung nach dem Schweregrad der Änderungsanträge erfolgt. Wenn gewünscht ist, dass bei der Gewichtung immer die aktuelle Belegung des Attributs verwendet wird, und nicht die Belegung zum jeweiligen früheren

Tabelle 4.1.: Metrik-Spezifikation als Pseudocode

Basisfilter:	„Produkt 1“
Gruppierung:	Keine
Gruppenwertberechnung:	„Durchschnittliche Bearbeiterwechsel pro Änderungsantrag“ $:= \frac{\text{Summe der Bewertungszahlen für „Bearbeiterwechsel“}}{\text{Anzahl der Bewertungszahlen für „Bearbeiterwechsel“}}$
Bewertungsverfahren:	„Bearbeiterwechsel“ Zähle das Ereignis „Wechsel des zuständigen Bearbeiters“ und erstelle eine Bewertungszahl beim „Übergang in Status CLOSED“
Berechnungszeitraum:	Von „2007-01-01“ bis „2009-12-31“
Zeitgranularität:	Monat
Fixierte Attribute:	Keine

Zeitpunkt, so kann das Attribut „Schweregrad“ als fixiertes Attribut spezifiziert werden.

4.4.3. Beispiel

Der Ablauf einer Metrikberechnung soll zunächst anhand eines Beispiels verdeutlicht werden. Tabelle 4.1 zeigt dazu eine Metrik-Spezifikation als Pseudocode.

Die beschriebene Metrik bestimmt die durchschnittliche Anzahl der Bearbeiterwechsel von Änderungsanträgen, die in einer Zeitperiode geschlossen wurden. Der Basisfilter legt fest, dass nur Änderungsanträge zu „Produkt 1“ betrachtet werden. Eine Gruppierung in mehrere Ergebniszeitreihen wird nicht verwendet. In der Gruppenwertberechnung wird das Bewertungsverfahren über den Bezeichner „Bearbeiterwechsel“ referenziert. Dieses Bewertungsverfahren ist vom Typ „Zählen von Ereignissen bis zu einem Ereignis“. Eine Bewertungszahl wird also genau dann erstellt, wenn der Status eines Änderungsantrags zum ersten Mal den Wert „CLOSED“ annimmt. Auf Basis der erstellten Bewertungszahlen in einer Zeitperiode wird dann mit der Gruppenwertberechnung die durchschnittliche Anzahl der Bearbeiterwechsel ermittelt. Der Ablauf der Berechnung dieser Metrik ist in Abbildung 4.2 skizziert.

Die entsprechende Metrik-Spezifikation in ITMS findet sich in Abbildung 4.8 auf Seite 68. Eine formale Beschreibung der Semantik dieser Metrik findet sich in Abschnitt 4.6.7 auf Seite 81.

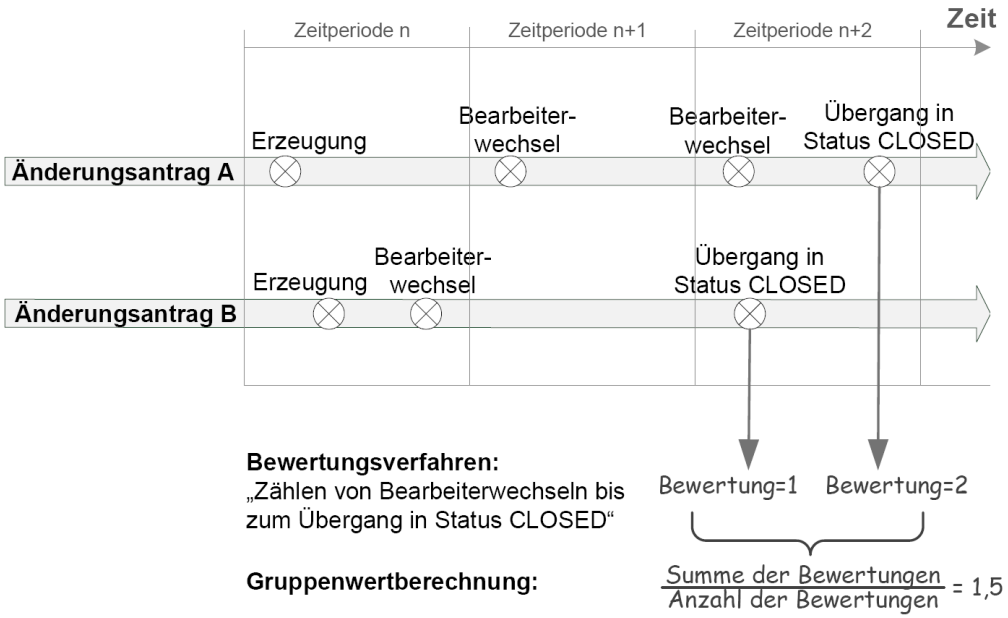


Abbildung 4.2.: Beispiel einer Metrikberechnung

4.4.4. Überblick über den Ablauf einer Metrikberechnung

Nachdem die einzelnen Elemente einer Metrik-Spezifikation genannt sind, wird in diesem Abschnitt ein Überblick über den Ablauf von Metrikberechnungen gegeben.

Grundlage für die Berechnungen ist die Historie von Änderungsanträgen. Jeder Änderungsantrag wird zu einem bestimmen Zeitpunkt erzeugt und hat einen durch die Belegung von Attributen beschriebenen Zustand. Im Lebenslauf eines Änderungsantrags passieren bestimmte Ereignisse, bei denen sich der Zustand des Änderungsantrags möglicherweise verändert.

Grundsätzlich läuft eine Metrikberechnung in den folgenden Schritten ab. Es wird dabei immer eine einzelne Zeitperiode aus dem Berechnungszeitraum betrachtet.

1. Zunächst wird die Menge der in einer Zeitperiode zu betrachtenden Änderungsanträge bestimmt. Diese Menge hängt von dem Basisfilter und den in der Metrik verwendeten Ereignisfiltern ab.

2. Für die Änderungsanträge in der Menge werden dann entsprechend dem Bewertungsverfahren eine oder mehrere Bewertungen ermittelt. Jedem Änderungsantrag wird zu bestimmten Zeitpunkten eine reelle Bewertungszahl zugeordnet. Diese Zahl hängt von dem verwendeten Bewertungsverfahren ab. Es kann der Standardwert 1 sein, um z.B. Ereignisse zu zählen, oder auch eine Gewichtung des Änderungsantrags, z.B. nach Alter oder Schweregrad. Einem Änderungsantrag können eine oder mehrere solcher Bewertungen zugeordnet werden. Das genaue Vorgehen wird durch die

Kategorie der sogenannten Wertberechnung bestimmt (siehe Abschnitt 4.4.2.4).

3. Auf die Menge aller Bewertungen von Änderungsanträgen in einem Zeitintervall wird eine Gruppenwertberechnung angewendet, die einen einzelnen Wert für dieses Zeitintervall zurückliefert. Typische Gruppenwertberechnungen sind z.B. das Aufsummieren aller Bewertungen oder die Bestimmung des Minimums.

4. Optional können die Ergebnisse aus mehreren Gruppenwertberechnungen entsprechend den Schritten 1-3 mit arithmetischen Operationen verknüpft werden. Damit lassen sich auf flexible Weise abgeleitete Metriken definieren.

4.4.5. Bewertungsverfahren im Detail

Nachdem im vorherigen Abschnitt der gemeinsame Ablauf der Berechnungen für in ITMS gegebene Metrik-Spezifikationen erläutert wurde, folgt nun die genaue Beschreibung der bereits in Abschnitt 4.4.2.4 eingeführten Verfahren zur Ermittlung von Bewertungen für einzelne Änderungsanträge. Grundsätzlich gilt für alle Bewertungsverfahren, dass nur solche Änderungsanträge betrachtet werden, die dem in der Metrik-Spezifikation gegebenen Basisfilter entsprechen.

4.4.5.1. Das Bewertungsverfahren „Zählen von Ereignissen in einer Zeitperiode“

Zur Spezifikation dieses Bewertungsverfahrens muss ein Ereignisfilter E und eine Gewichtung W angegeben werden. Immer dann, wenn für einen Änderungsantrag ein Ereignis auftritt, das dem Ereignisfilter entspricht, wird eine Bewertung erzeugt. Zur Bestimmung der Bewertungszahl wird die **Gewichtung** W verwendet. Gewichtungen können auf der Belegung eines Attributs oder mehrerer Attribute basieren. Beispiele für solche Gewichtungen sind der Schweregrad oder das Alter des Änderungsantrags. Weiterhin ist eine Gewichtung mit 1 als Standardwert möglich. ITMS muss um zusätzliche Gewichtungen erweiterbar sein.

Ein Beispiel für den Einsatz des Bewertungsverfahrens „Zählen von Ereignissen in einer Zeitperiode“ ist die Frage, wie viele Änderungsanträge in einer Zeitperiode erzeugt wurden. Als Ereignisfilter E wird dazu das Ereignis „Erzeugung eines Änderungsantrags“ spezifiziert. Als Gewichtung W wird die Gewichtung mit dem Wert 1 verwendet.

Ein weiteres Beispiel ist der geschätzte Restaufwand für alle offenen Änderungsanträge. Dazu wird im Ereignisfilter E spezifiziert, dass jeweils alle offenen Änderungsanträge am Ende einer Zeitperiode betrachtet werden. Als Gewichtung W wird der geschätzte Restaufwand des Änderungsantrags verwendet.

Zusammenfassend lassen sich mit diesem Bewertungsverfahren solche Metriken beschreiben, bei denen nicht in irgendeiner Form über die Zeitperioden der Metrik-Spezifikation hinweg gezählt wird. Somit kann dieses Bewertungsverfahren für ein breites Spektrum von Metriken eingesetzt werden. Metriken, die Ereignisse über mehrere Zeitperioden hinweg zählen, oder die die Zeitdauer zwischen Ereignissen im Lebenslauf eines Änderungsantrags betrachten, können nicht mit diesem Bewertungsverfahren beschrieben werden. In diesem Fall kommt eines der drei folgenden Verfahren in Frage.

4.4.5.2. Das Bewertungsverfahren „Zählen von Ereignissen bis zu einem Ereignis“

Bei diesem Verfahren wird das Auftreten eines bestimmtes Ereignisses A über Zeitperioden hinweg gezählt. Eine Bewertung wird beim **ersten** Auftreten eines Ereignisses B ermittelt. Als Bewertungszahl wird dann die Anzahl des Auftretens von Ereignis A zugewiesen.

Ein Beispiel für den Einsatz dieses Bewertungsverfahrens ist die Frage, wie oft der zuständige Bearbeiter gewechselt hat, bis ein Änderungsantrag geschlossen wurde. Als Ereignis A wird in diesem Fall der Zustandswechsel beim Attribut „Bearbeiter“ spezifiziert, als Ereignis B der Zustandswechsel des Attributs „Status“ nach „Geschlossen“.

4.4.5.3. Das Bewertungsverfahren „Intervalllänge“

Mit diesem Bewertungsverfahren wird die Länge eines Zeitintervalls zwischen einem Ereignis A und einem Ereignis B im Lebenslauf eines Änderungsantrags ermittelt. Tritt das Ereignis A mehrmals auf, wird standardmäßig bei jedem Auftreten von B die Intervalllänge jeweils bis zum nächsten zeitlich davor liegenden Auftreten von A bestimmt. Zur Spezifikation des Verfahrens müssen Ereignisfilter für A und B angegeben werden.

Ein Beispiel für den Einsatz dieses Bewertungsverfahrens ist die Frage, wie viel Zeit vergangen ist, wenn mit der Bearbeitung eines Änderungsantrags begonnen wird. In diesem Fall ist der Ereignisfilter A die Erzeugung eines Änderungsantrags. Der Ereignisfilter B spezifiziert die Änderung des Attributs „Status“ auf den Wert „In Bearbeitung“.

Es gibt zwei optionale Parameter, mit denen das Bewertungsverfahren angepasst werden kann. Erstens kann eine **obere Schranke** (*Threshold*) für die Intervalllänge angegeben werden. Folgt das Ereignis B nicht innerhalb der durch die Schranke gegebenen maximalen Intervalllänge auf ein Ereignis A, so wird bereits zu diesem Zeitpunkt eine Bewertungszahl bestimmt. Hier gibt es dann zwei Alternativen:

- Als Bewertungszahl wird die bisher vergangene Zeit seit dem Ereignis A verwendet. Somit ist die Bewertungszahl immer kleiner oder gleich der oberen Schranke.
- Als Bewertungszahl wird 1 vergeben, genau dann, wenn die obere Schranke überschritten wurde, andernfalls wird der Wert 0 vergeben.

Die obere Schranke dient beispielsweise dazu, Ausreißer nicht bzw. nicht so stark zu berücksichtigen, oder Verletzungen von geforderten Antwortzeiten oder Lösungszeiten zu zählen, wie sie etwa in einem *Service Level Agreement* definiert werden.

Der zweite optionale Parameter ist der **Bewertungszeitpunkt**, zu dem das Auftreten des Ereignisses B berücksichtigt werden soll. Folgende Einstellungen sind möglich:

- **Erstes Auftreten**: Eine Bewertungszahl wird nur beim erstmaligen Auftreten von Ereignis B nach dem Auftreten von Ereignis A erstellt.
- **Jedes Auftreten**: Eine Bewertungszahl wird bei jedem Auftreten des Ereignisses B nach einem Ereignis A erzeugt. Die Ermittlung der Intervalllänge beginnt also immer wieder neu mit dem Auftreten von Ereignis A. Dies ist das Standardverhalten, wenn der Parameter nicht angegeben ist.
- **Letztes Auftreten**: Eine Bewertungszahl wird nur bei dem zeitlich letzten Auftreten von Ereignis B erzeugt.

Bei allen diesen Einstellungen wird die Länge des Zeitintervalls immer von dem nächsten vor B liegenden Auftreten des Ereignisses A gezählt. Andere Möglichkeiten wären hier prinzipiell denkbar, ähnlich dem oben beschriebenen Parameter für den Bewertungszeitpunkt. Nach unseren bisherigen Erfahrungen ergab sich allerdings noch nicht die Notwendigkeit, Metriken auf diese Weise zu spezifizieren. Deswegen muss in ITMS kein weiterer optionaler Parameter für dieses Bewertungsverfahren eingeführt werden.

4.4.5.4. Das Bewertungsverfahren „Verweilzeit“

Mit diesem Bewertungsverfahren wird bestimmt, wie lange ein Änderungsantrag insgesamt in einem bestimmten Zustand verweilt hat. Zur Spezifikation des Verfahrens muss ein Zustandsfilter F angegeben werden, sowie ein Ereignisfilter E, der angibt, wann eine Bewertungszahl erzeugt werden soll. Die Bewertungszahl enthält dann die gesamte Verweilzeit in dem durch F spezifizierten Zustand. Ähnlich wie bei der Berechnung einer Intervalllänge kann optional spezifiziert werden, ob die Bewertungszahl bei jedem Auftreten des Ereignisses E, nur beim ersten, oder nur beim zeitlich letzten Auftreten erzeugt werden soll.

Ein Beispiel für den Einsatz dieses Bewertungsverfahrens ist die Ermittlung der Verweilzeit eines Änderungsantrags im Zustand „In Bearbeitung“ bis zum Schließen des Änderungsantrags. Dazu wird im Zustandsfilter F spezifiziert, dass

das Attribut „Status“ den Wert „In Bearbeitung“ haben muss. Der Ereignisfilter E spezifiziert die Änderung des Attributs „Status“ zu dem Wert „Geschlossen“.

Der Unterschied zwischen den Bewertungsverfahren für die Verweilzeit und dem für die Intervalllänge besteht darin, dass bei der Verweilzeit auch mehrere Zeitintervalle aufaddiert werden, während denen der Zustandsfilter für einen Änderungsantrag erfüllt war. Stattdessen wird bei der Intervalllänge nur das jeweils letzte Intervall bei der Ermittlung der Bewertungszahl berücksichtigt.

4.4.6. Zusammenfassung der Anforderungen

Kern der Sprache ITMS sind die vier hier dargestellten Bewertungsverfahren. Diese werden mit Zustands- und Ereignisfiltern sowie mit Gewichtungen parametrisiert. Weiterhin wird mit dem Basisfilter die Menge der betrachteten Änderungsanträge eingegrenzt. Die Filter lassen sich durch logischen Operationen flexibel kombinieren. Mit Hilfe von Gruppenwertberechnungen wird dann aus den Ergebnissen der Bewertungsverfahren eine Zeitreihe ermittelt. Die betrachteten Zeitperioden und der Berechnungszeitraum werden ebenfalls in der Metrik-Spezifikation bestimmt.

Die Auswertung von unterschiedlichen Software-Entwicklungsarchiven erfordert Variabilität beim Umfang der Sprache. Die Menge der Attribute, als auch die möglichen Gewichtungen und Ereignisse hängen also von den konkret auszuwertenden Entwicklungsarchiven ab.

Eine weitere wichtige Anforderung ist die Erweiterbarkeit der Sprache, beispielsweise um zusätzliche Gewichtungen, Filter und Gruppenwertberechnungen.

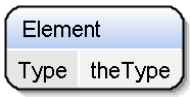
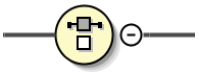
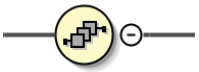
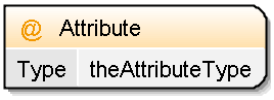
4.5. Aufbau von ITMS

Nachdem die Anforderungen an die Sprache vorgestellt sind, werden in diesem Abschnitt detailliert die Sprachelemente und die Syntax von ITMS vorgestellt. Die Syntax der Sprache ITMS basiert auf XML.

Details der Syntax sind je nach dem auszuwertenden Software-Entwicklungsarchiv unterschiedlich ausgeprägt. Der hier vorgestellte grundsätzliche Aufbau einer ITMS Metrik-Spezifikation ist unabhängig von den auszuwertenden Software-Entwicklungsarchiven. Auf Erweiterungsmöglichkeiten der Sprache und gegebenenfalls unterschiedliche Ausprägungen wird an den jeweiligen Stellen hingewiesen.

Eine vollständige Syntaxbeschreibung für ein konkretes Software-Entwicklungsarchiv ist beispielsweise für Auswertungen auf dem Issue-Tracking-System Bugzilla verfügbar [Bug09b]. Außerdem ist hier auch die Syntax für die optionale Anbindung der Konfigurationsmanagement-Systeme CVS und Subversion beschrieben.

Tabelle 4.2.: Diagrammelemente zur Darstellung eines XML Schemas

XML-Schema-Element	Diagrammdarstellung
Element-Deklaration	
Element-Deklaration mit Angabe des Typs	
Definition eines komplexen Typs	
Definition eines einfachen Typs	
Elementgruppen-Definition	
Auswahlgruppe	
Sequenzgruppe	
Attribut mit Angabe des Typs	

In diesem Abschnitt wird zur Beschreibung der Syntax eine Diagrammdarstellung des XML-Schemas verwendet. Diese Darstellung ist für eine XML-basierte Sprache übersichtlicher als eine Darstellung in EBNF oder als Syntaxdiagramm. Tabelle 4.2 zeigt eine Legende der verwendeten Diagrammelemente. Die gleiche oder ähnliche Diagrammdarstellungen finden sich in gängigen XML-Schema-Editoren [Oxy, Alt, Sty]. Konkrete Ausschnitte aus ITMS Metrik-Spezifikationen werden zusätzlich angegeben, um die Syntax zu veranschaulichen.

Im Folgenden wird zunächst der grundlegende Aufbau einer ITMS Metrik-Spezifikation beschrieben. Es folgen detaillierte Beschreibungen für die einzelnen Elemente. Ein zusammenhängendes Beispiel einer Metrik-Spezifikation in ITMS wird schließlich in Abschnitt 4.5.4 vorgestellt.

Den Aufbau einer ITMS Metrik-Spezifikation zeigt Abbildung 4.3. Die Wurzel des XML-Dokuments ist das Element *metric*. Die direkt untergeordneten Elemente haben eine feste Reihenfolge. Sie entsprechen den in Abschnitt 4.4.2 beschriebenen Bestandteilen einer Metrik-Spezifikation. Die Syntax dieser Elemente wird im Folgenden vorgestellt.

- **baseFilter** Das Element *baseFilter* entspricht dem in Abschnitt 4.4.2.2 vorgestellten Basisfilter. Es hat daher den Typ eines Zustandsfilters (*stateFilter*). Zustandsfilter werden in Abschnitt 4.5.1 vorgestellt.
- **groupingParameter** Das Element *groupingParameter* beschreibt die Gruppierung der Ergebnisse in mehrere Zeitreihen (vgl. Abschnitt 4.4.2.3).

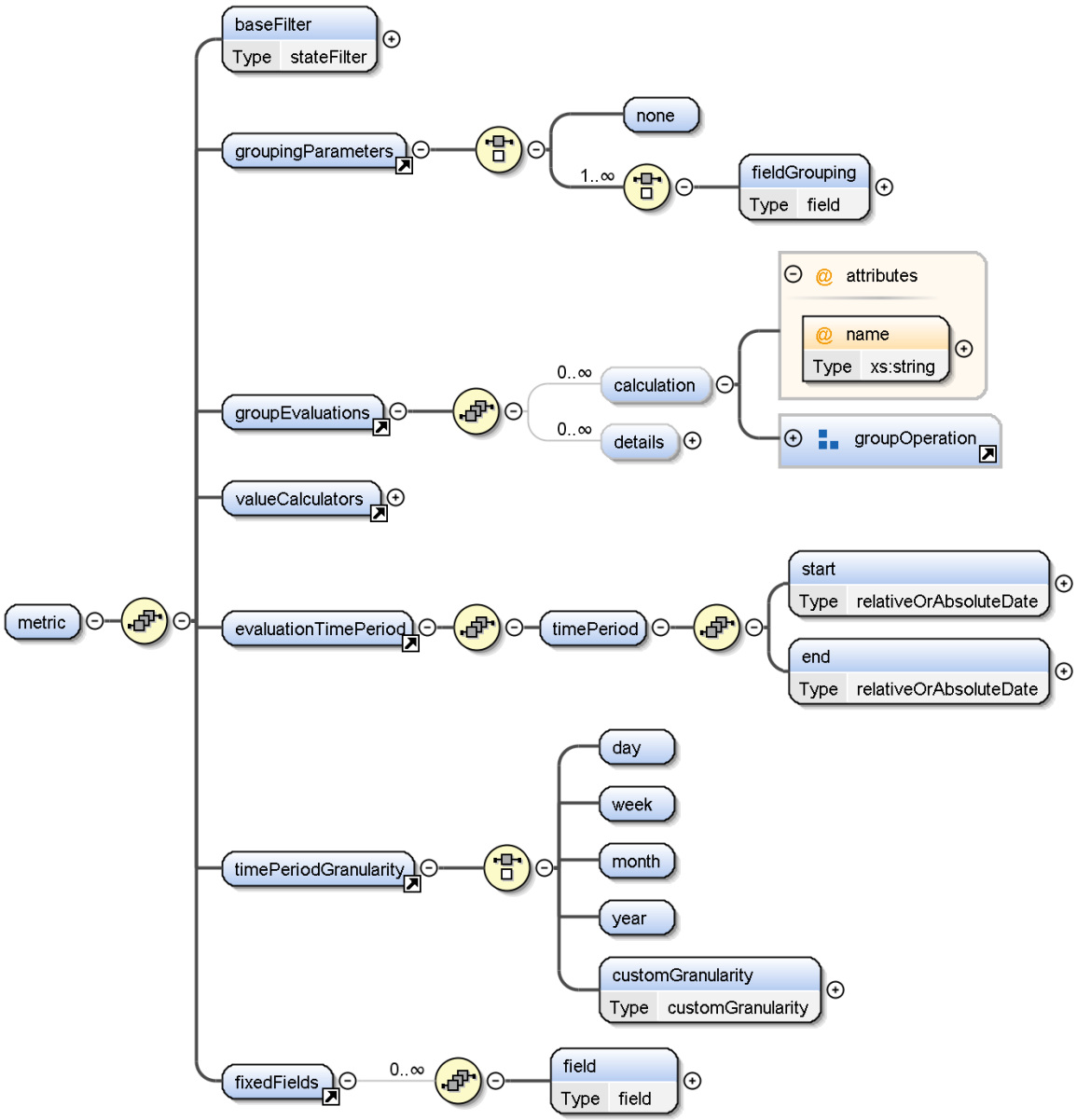


Abbildung 4.3.: Aufbau einer Metrik-Spezifikation (Legende siehe Tabelle 4.2)

Es können beliebig viele Attribute aufgelistet werden, nach denen gruppiert wird. Ein Attribut selbst ist vom Typ *field*, einem Auflistungstyp der zulässigen Attribute. Dieser Typ muss spezifisch für die auszuwertenden Software-Entwicklungsarchive ausgeprägt werden. Bei Angabe des XML-Tags *none* wird keine Gruppierung verwendet. Das folgende Beispiel zeigt den XML-Code für das Element *groupingParameters* und eine Gruppierung nach den Attributen Produkt und Version.

```
<groupingParameters>
  <fieldGrouping>product</fieldGrouping>
  <fieldGrouping>version</fieldGrouping>
</groupingParameters>
```

- **groupEvaluations**: Dieses Element enthält beliebig oft die untergeordneten Elemente *calculation* und *details*. Im Element *calculation* wird eine Gruppenwertberechnung spezifiziert (vgl. Abschnitt 4.4.2.5). Eine Gruppenwertberechnung hat einen eindeutigen Bezeichner im Attribut *name* des XML-Elements. Dieser wird in der Ergebniszeitreihe ausgegeben. Das Element *calculation* ist vom Typ *groupOperation*, dessen Syntax genauer in Abschnitt 4.5.2 beschrieben wird. Das Element *details* wird verwendet, um für ein in der Metrik spezifiziertes Bewertungsverfahren die ermittelten Bewertungszahlen aller betrachteten Änderungsanträge als zusätzliche Details auszugeben.
- **valueCalculators**: In diesem Element können beliebig viele Bewertungsverfahren spezifiziert werden (vgl. Abschnitt 4.4.2.4). Die Details zur Syntax werden in Abschnitt 4.5.3 erläutert.
- **evaluationTimePeriod** spezifiziert den Berechnungszeitraum (vgl. Abschnitt 4.4.2.1). Dazu wird in den Elementen *start* und *end* das Anfangs- und Enddatum angegeben, entweder als absolutes oder relatives Datum (z.B. *today*).
- **timePeriodGranularity** gibt an, in welche Zeitperioden der Berechnungszeitraum unterteilt wird (vgl. Abschnitt 4.4.2.1). Zum einen können vorgegebene Zeitperioden wie Monat oder Jahr verwendet werden. Zum anderen können mit dem Element *customGranularity* auch benutzerdefinierte Zeitperioden spezifiziert werden.
- **fixedFields**: Mit diesem Element werden fixierte Attribute spezifiziert (vgl. Abschnitt 4.4.2.6). Es können beliebig viele Attribute aufgelistet werden.

4.5.1. Zustandsfilter

Abbildung 4.4 zeigt die Syntax des Elements Zustandsfilter. Zunächst möchten wir atomare Zustandsfilter betrachten (*atomicStateFilter*). Ein atomarer Zustandsfilter spezifiziert Bedingungen an die Belegung genau eines Attributs

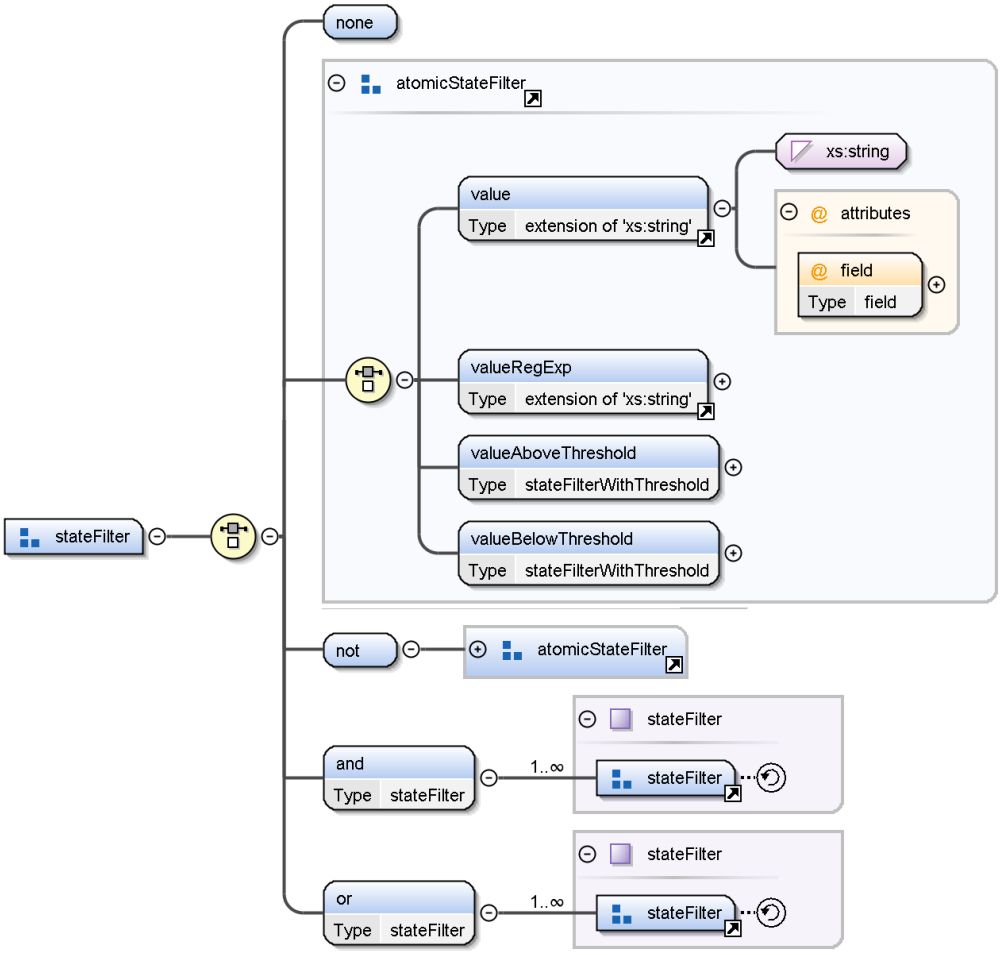


Abbildung 4.4.: Aufbau eines Zustandsfilters in einer Metrik-Spezifikation (Legende siehe Tabelle 4.2)

eines Änderungsantrags. Der *value*-Zustandsfilter akzeptiert alle Änderungsanträge, bei denen ein angegebenes Attribut mit dem angegebenen Wert belegt ist. Der Zustandsfilter in dem folgenden Beispiel akzeptiert nur solche Änderungsanträge, die dem Produkt mit der *Id* 4 zugeordnet sind. *Field* referenziert dabei den Namen des Attributs in der Datenbank. Bei Attributen bei denen eine *Id* vergeben ist, wird im Filter ebenfalls die *Id* verwendet, ansonsten wird der geforderte Wert des Attributs angegeben.

```
<value field="product">4</product>
```

Weiterhin gibt es atomare Zustandsfilter für das Filtern mit Hilfe von regulären Ausdrücken (*valueRegExp*), oder das Filtern von Werten, die unterhalb oder oberhalb einer angegebenen Schranke liegen (*valueAboveThreshold* bzw. *valueBelowThreshold*).

Ein atomarer Zustandsfilter kann mit *not* negiert werden. Schließlich können Zustandsfilter mit den logischen Operatoren *and* und *or* verknüpft werden.

4.5.2. Gruppenwertberechnung

Abbildung 4.5 zeigt die Syntax zur Beschreibung von Gruppenwertberechnungen (vgl. Abschnitt 4.4.2.5). Mit den gezeigten Operationen vom Typ *simpleOperation* werden die mit Hilfe eines Bewertungsverfahrens ermittelten Bewertungen zu einem einzigen Resultat je Zeitperiode verrechnet. Dazu wird die Spezifikation des Bewertungsverfahrens durch den im Attribut *valueCalculator* gegebenen Bezeichner referenziert. Mögliche Operationen sind Anzahl, Summe, Minimum, Maximum und Median der erstellten Bewertungszahlen in einer Zeitperiode. Weiterhin kann mit der Operation *countUnique* die Anzahl der Änderungsanträge ermittelt werden, für die mindestens eine Bewertungszahl erstellt wurde.

Bei Operationen vom Typ *thresholdOperation* wird mit dem Attribut *threshold* zusätzlich noch eine Schranke für die Bewertungszahlen angegeben. Es werden dann nur solche Bewertungszahlen gezählt oder aufsummiert, die unter- bzw. oberhalb der angegebenen Schranke liegen. Hier wird deutlich, dass noch weitere Gruppenwertberechnungen denkbar sind, beispielsweise die Berechnung anderer Mittelwerte. Daher kann der Sprachumfang von ITMS um zusätzliche Gruppenwertberechnungen erweitert werden.

Mit den Operationen vom Typ *binaryOperation* werden Gruppenwertberechnungen durch arithmetische Operationen verknüpft. Das Element *constant* dient dazu, bei solchen Bewertungen einen konstanten Zahlenwert statt einer Zeitreihe zu verwenden.

4.5.3. Bewertungsverfahren

Der Aufbau der Spezifikation der vier verschiedenen Bewertungsverfahren ist in Abbildung 4.6 zu sehen (vgl. auch Abschnitt 4.4.5). Es können beliebig viele

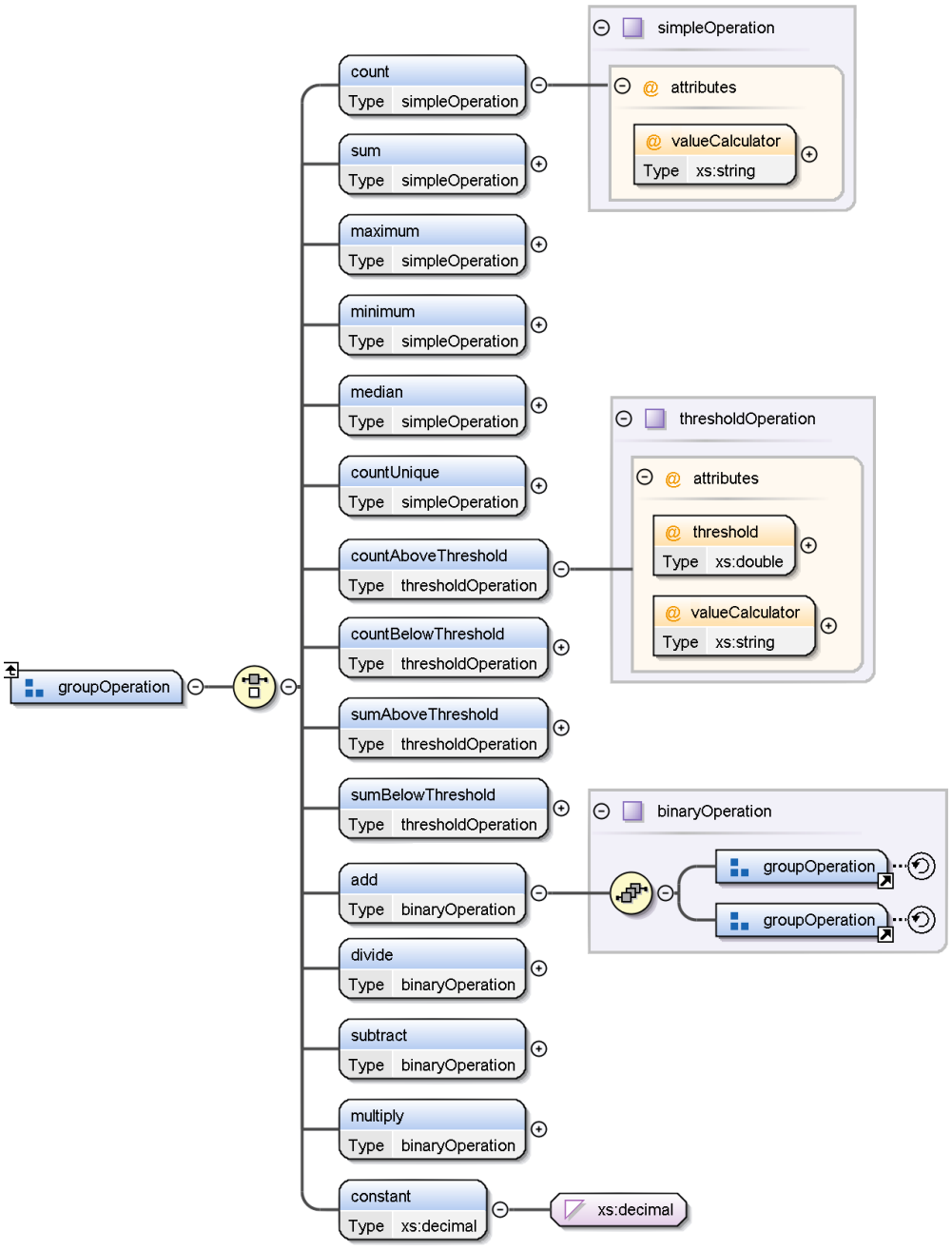


Abbildung 4.5.: Aufbau einer Gruppenwertberechnung (Legende siehe Tabelle 4.2)

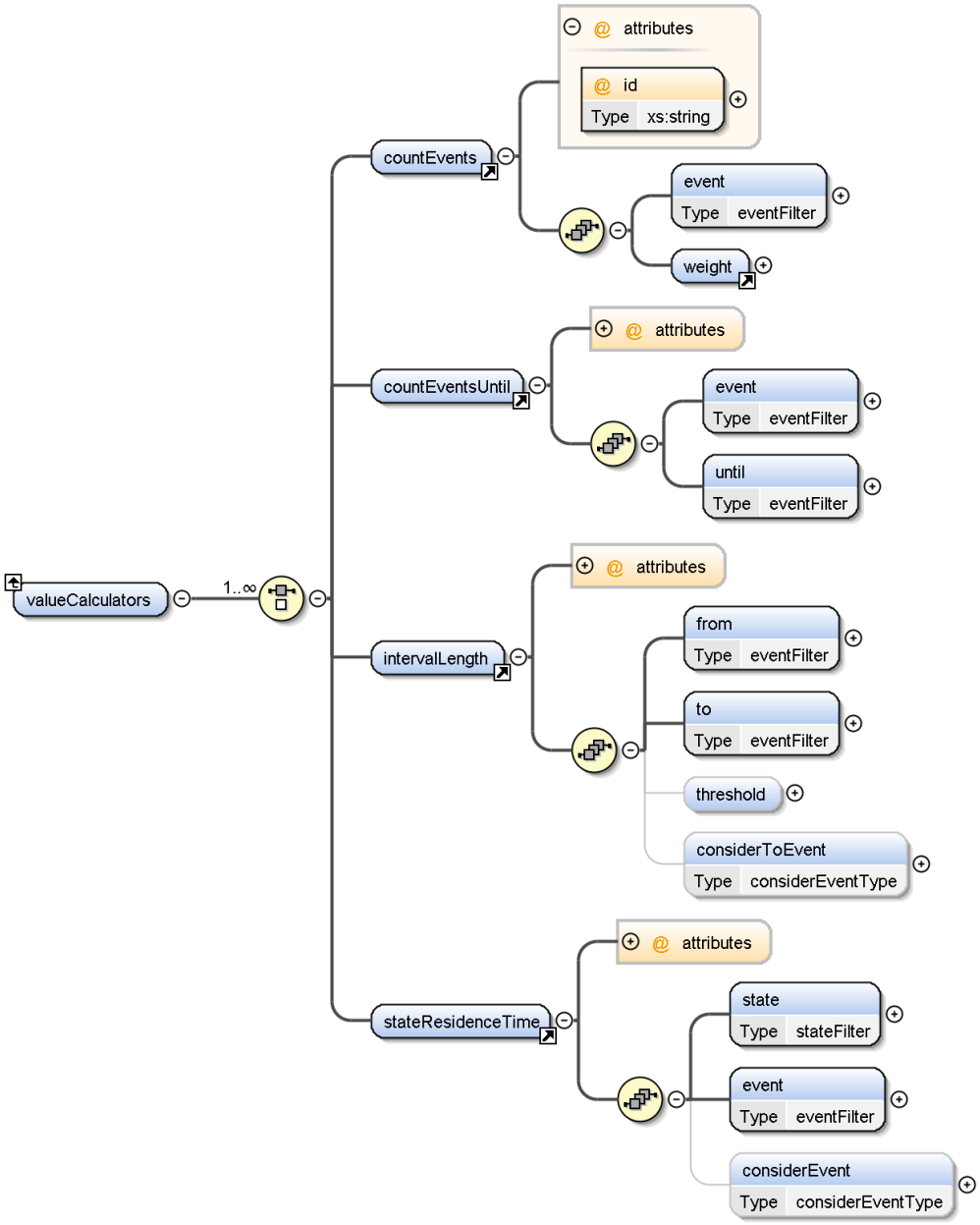


Abbildung 4.6.: Aufbau der Bewertungsverfahren (Legende siehe Tabelle 4.2)

unterschiedliche Bewertungsverfahren in einer Metrik-Spezifikation angegeben werden. Jedes eingesetzte Bewertungsverfahren muss mit einem eindeutigen Bezeichner gekennzeichnet werden, der im Attribut *id* angegeben ist. Dieser Bezeichner wird in den Gruppenwertberechnungen referenziert.

Zu jedem Bewertungsverfahren sind in den untergeordneten Elementen bestimmte Parameter anzugeben, die das jeweilige Verfahren genauer spezifizieren. Dazu werden typischerweise Ereignisfilter angegeben. Die Spezifikation von Ereignisfiltern in ITMS wird in Abschnitt 4.5.3.2 vorgestellt. Im folgenden werden zunächst die einzelnen Parameter für die Bewertungsverfahren erläutert.

Zählen von Ereignissen in einer Zeitperiode (countEvents)

- Mit dem Ereignisfilter *event* wird das zu zählende Ereignis spezifiziert.
- Der Parameter *weight* gibt an, wie Änderungsanträge bei der Zählung zu gewichten sind (siehe Abschnitt 4.5.3.1).

Zählen von Ereignissen bis zu einem Ereignis (countEventsUntil)

- Mit dem Ereignisfilter *event* wird das zu zählende Ereignis spezifiziert.
- Der Ereignisfilter *until* gibt an, bei welchem Ereignis im Lebenslauf eines Änderungsantrags die Zählung beendet wird, und eine entsprechende Bewertungszahl erstellt wird.

Intervalllänge (intervalLength)

- Mit den Ereignisfiltern *from* und *to* wird das Start- und Endereignis für die Bestimmung einer Intervalllänge spezifiziert.
- Optional kann in dem Parameter *treshold* eine obere Schranke für die Intervalllänge angegeben werden (vgl. Abschnitt 4.4.5.3).
- Mit dem Parameter *considerToEvent* wird angegeben, zu welchen Zeitpunkten eine Bewertungszahl erstellt werden soll. Dies kann bei jedem Auftreten des Endereignisses, sowie nur beim ersten oder nur beim letzten Auftreten dieses Ereignisses erfolgen. Als Parameter wird dazu ein Literal aus dem Aufzählungstyp *eachTime, firstTime, lastTime* angegeben. Fehlt diese Angabe, so wird *eachTime* als Standardwert genommen.

Verweilzeit (stateResidenceTime)

- Der Zustandsfilter *state* spezifiziert, welcher Zustand eines Änderungsantrags bei der Verweilzeit betrachtet wird.
- Der Ereignisfilter *event* gibt an, zu welchen Zeitpunkt(en) eine Bewertungszahl erstellt wird.

- Mit dem Parameter *considerEventType* wird wiederum genauer spezifiziert, ob eine Bewertungszahl bei jedem Auftreten, nur beim ersten oder nur beim letzten Auftreten dieses Ereignisses erstellt wird.

Das folgende Code-Beispiel zeigt die Spezifikation eines Bewertungsverfahrens der Kategorie *countEvents*, also dem Zählen von Ereignissen. Als Ereignisse werden sowohl das Erstellen eines Änderungsantrags betrachtet (Ereignisfilter *create*), als auch die Änderung des Attributs *status* von einem der Werte *RESOLVED* oder *VERIFIED* auf den Wert *REOPENED*. Es erfolgt eine Gewichtung jedes dieser Ereignisse auf Basis des für den Änderungsantrag gesetzten Schweregrades.

```
<countEvents id="Incoming_Requests_Weighted_By_Severity">
  <event>
    <or>
      <create />
      <transition field="status">
        <from>RESOLVED</from>
        <from>VERIFIED</from>
        <to>REOPENED</to>
      </transition>
    </or>
  </event>
  <weight>
    <mapping field="severity">
      <map from="blocker"  to="4" />
      <map from="critical" to="3" />
      <map from="major"    to="2" />
      <map from="normal"   to="1" />
    </mapping>
  </weight>
</countEvents>
```

4.5.3.1. Gewichtung

Wie in Abschnitt 4.4.5.1 bereits beschrieben und auch in dem vorherigen Beispiel zu sehen, wird bei dem Bewertungsverfahren „Zählen von Ereignissen“ eine sogenannte Gewichtung verwendet. Diese legt fest, welche Bewertungszahl beim Auftreten eines Ereignisses im Lebenslauf eines Änderungsantrags erzeugt wird. Im Folgenden wird eine Reihe von Gewichtungen genannt. Das Schlüsselwort für das XML-Element in der ITMS Metrik-Spezifikation steht jeweils in Klammern.

- **Standardgewichtung**: Bei dieser Einstellung werden alle Änderungsanträge mit dem Wert 1 gewichtet (*default*).

- **Attribut-basierte Gewichtung**: Zur Gewichtung eines Änderungsantrags können mit Zahlwerten belegte Attribute herangezogen werden, wie beispielsweise der geschätzte Aufwand (*originalEstimatedEffort*), die Anzahl der Kommentare (*commentCount*) oder die Anzahl der abhängigen Änderungsanträge (*Blocks*).

- **Benutzerdefinierte attribut-basierte Gewichtung**: Der Benutzer kann eine Abbildung von Attributwerten auf Zahlwerte definieren. Beispielsweise kann von Schweregraden wie *blocker* oder *trivial* auf Zahlwerte abgebildet werden. Wie im obigen Beispiel zu sehen, werden dazu in dem Element *mapping* alle Abbildungen aufgelistet.

- **Zeitbasierte Gewichtung**: Die Gewichtung kann bestimmt werden durch die Zeit, die seit einem bestimmten Stichtag im Lebenslauf eines Änderungsantrags vergangen ist. Typische Gewichtungen sind das Alter des Änderungsantrags (*ageInDays*) oder die abgelaufene Zeit seit einem geforderten Fertigstellungstermin für den Änderungsantrag (*daysBeyondDeadline*).

- **Gewichtung auf Basis mehrerer Attributwerte**: Es können Gewichtungen verwendet werden, die unter Verwendung mehrerer Attributwerte berechnet werden. Diese müssen für ein konkretes Issue-Tracking-System implementiert werden. Ein Beispiel für diese Art der Gewichtung ist eine Indexzahl, die bewertet, wie weit der tatsächliche Aufwand vom geschätzten Aufwand abweicht (*estimationAccuracy*).

Die Sprache ITMS kann um weitere Arten von Gewichtungen erweitert werden. Insbesondere können die Gewichtungen spezifisch für die auszuwertenden Software-Entwicklungsarchive ausgeprägt werden.

4.5.3.2. Ereignisfilter

Die Syntax zur Spezifikation von Ereignisfiltern ist in Abbildung 4.7 zu sehen. Folgende Ereignisfilter sind möglich:

- **Erzeugung des Änderungsantrags (create)**: Mit diesem Ereignisfilter werden Änderungsanträge zum Zeitpunkt ihrer Erzeugung betrachtet.

- **Ende einer Zeitperiode (endOfTimeInterval)**: Dieser Ereignisfilter bezieht sich auf die in der Metrik spezifizierten Zeitperioden des Berechnungszeitraums. Es lassen sich so regelmäßig am Ende jeder Zeitperiode Zählungen vornehmen.

- **Eintreten in den Basisfilter (enterBaseFilter)**: Mit diesem Ereignisfilter werden Ereignisse betrachtet, bei denen sich die Belegung der Attributwerte eines Änderungsantrags ändert, sodass er die im Basisfilter der Metrik spezifizierten Bedingungen erfüllt (vgl. Abschnitt 4.4.2.2).

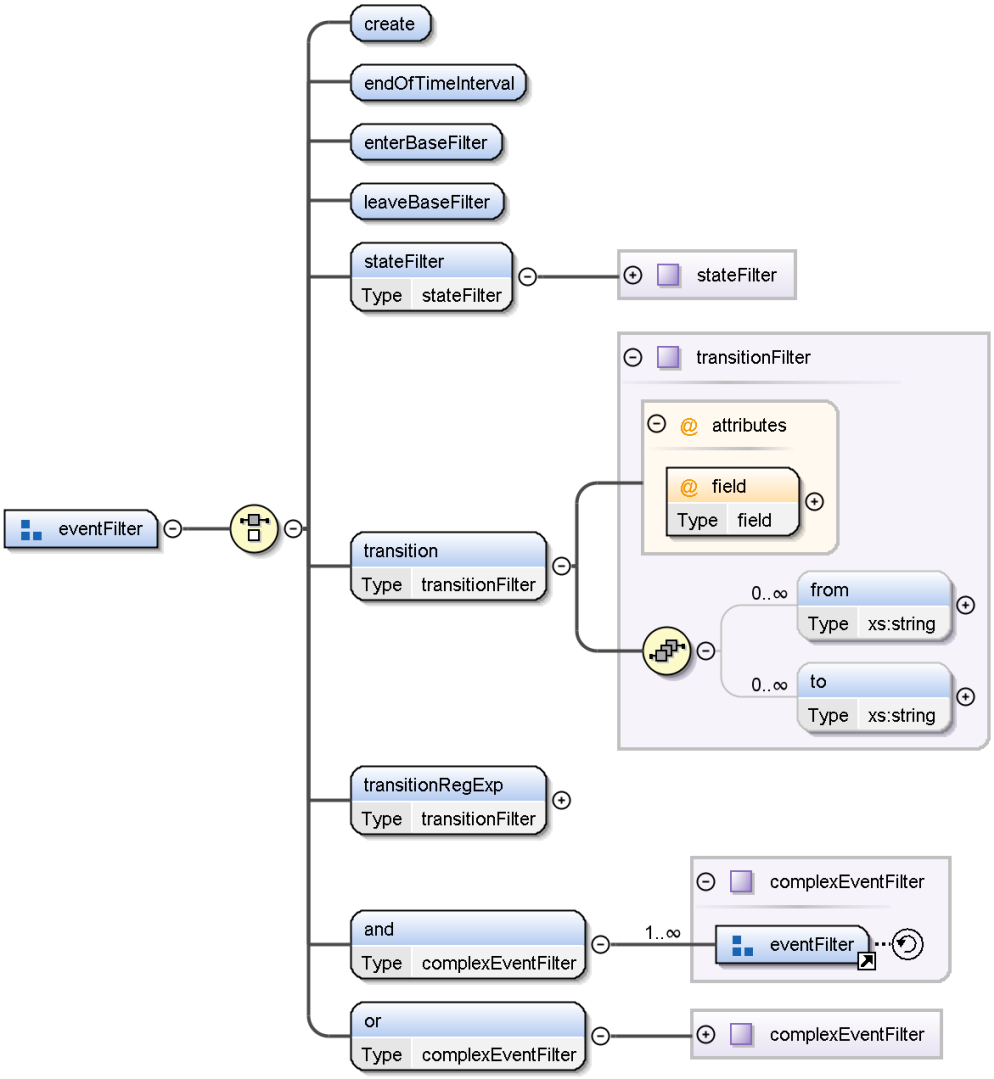


Abbildung 4.7.: Aufbau eines Ereignisfilters (Legende siehe Tabelle 4.2)

- **Verlassen des Basisfilters (leaveBaseFilter)**: Analog zum Eintreten in den Basisfilter kann sich die Belegung von Attributen auch so ändern, dass ein Änderungsantrag die Bedingungen im Basisfilter nicht mehr erfüllt.

- **Zustandsfilter als Ereignisfilter (stateFilter)**: Es können auch Zustandsfilter als Ereignisfilter eingesetzt werden. Diese akzeptieren alle Ereignisse, bei denen ein Änderungsantrag in dem beschriebenen Zustand ist. Dementsprechend wird ein Zustandsfilter für gewöhnlich durch *and* mit einem Ereignisfilter verknüpft, um die Bedingungen an das Ereignis zu verfeinern. Das folgende Beispiel zeigt einen Ereignisfilter für neu eingestellte Änderungsanträge mit der Priorität 1.

```
<event>
  <and>
    <create />
    <stateFilter>
      <value field="priority">P1</value>
    </stateFilter>
  </and>
</event>
```

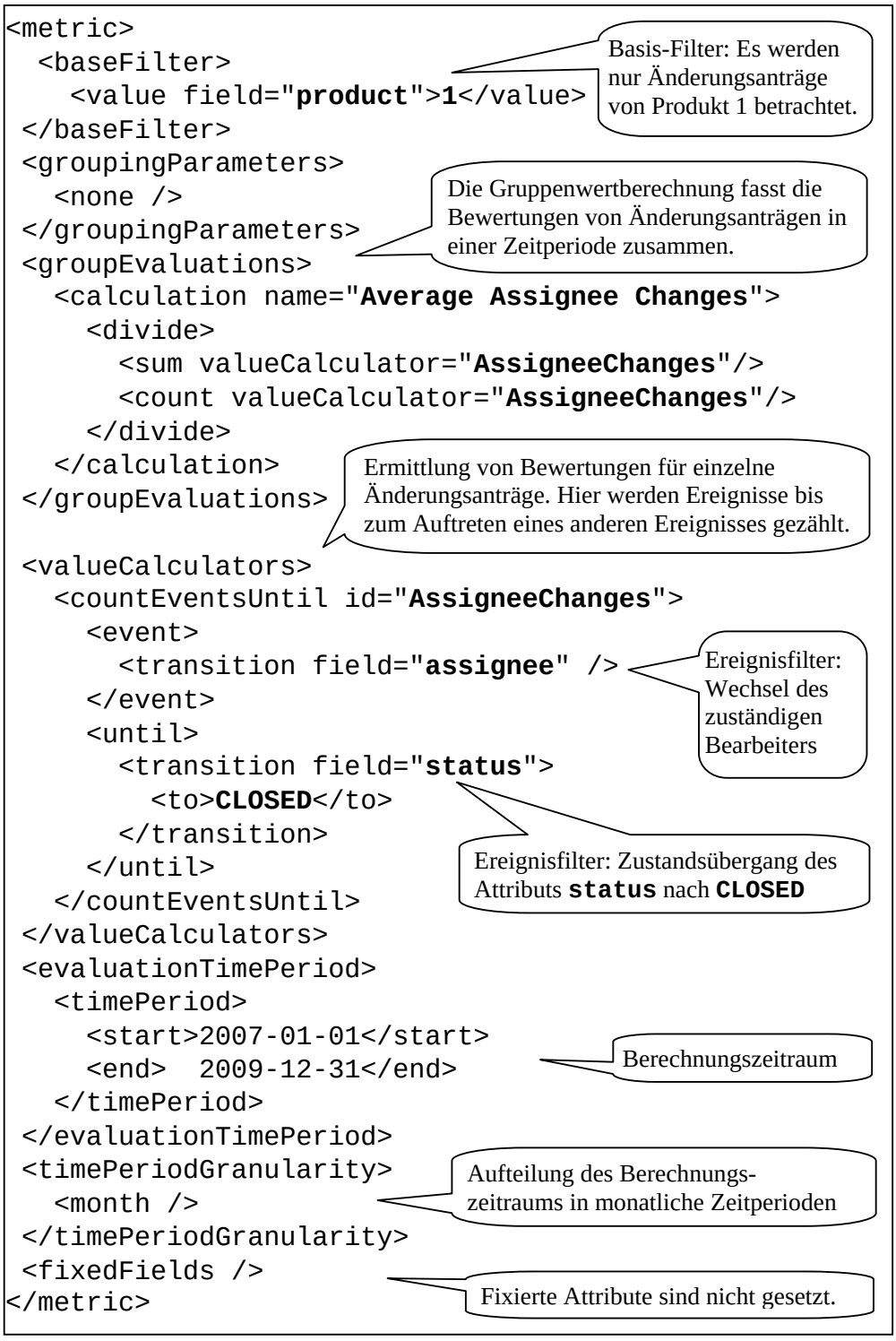
- **Zustandsänderung (transition)**: Mit diesem Ereignisfilter werden solche Ereignisse betrachtet, bei denen sich die Belegung eines bestimmten Attributs verändert. Dazu muss im Attribut *field* der Name des Attributs genannt werden. Mit den Elementen *from* und *to* können optional Bedingungen an die Belegung des Attributs vor oder nach der Zustandsänderung spezifiziert werden.

- **Zustandsänderung spezifiziert durch reguläre Ausdrücke (transitionRegExp)**: Analog zum vorgenannten Ereignisfilter können auch reguläre Ausdrücke verwendet werden, um auf flexible Weise Bedingungen an die Belegung der Attribute zu spezifizieren.

Alle vorgenannten Filter können mit den logischen Operatoren *and* und *or* verknüpft werden.

4.5.4. Beispiel

Um die oben geschilderten Konzepte zu verdeutlichen, soll hier als Beispiel eine vollständige Metrik-Spezifikation vorgestellt werden (siehe Abbildung 4.8). Die Metrik berechnet die durchschnittliche Anzahl der Bearbeiterwechsel eines Änderungsantrags zu dem Zeitpunkt, an dem dieser geschlossen wird. Diese Metrik wurde bereits in Abschnitt 4.4.3 auf Seite 50 in Form von Pseudocode eingeführt. Die Bedeutung der wesentlichen Elemente der Metrik-Spezifikation ist jeweils mit einer Legende erläutert. Die genaue Semantik dieser Spezifikation ist in Abschnitt 4.6.7 auf Seite 81 angegeben.




```
<metric>
  <baseFilter>
    <value field="product">1</value>
  </baseFilter>
  <groupingParameters>
    <none />
  </groupingParameters>
  <groupEvaluations>
    <calculation name="Average Assignee Changes">
      <divide>
        <sum valueCalculator="AssigneeChanges"/>
        <count valueCalculator="AssigneeChanges"/>
      </divide>
    </calculation>
  </groupEvaluations>

  <valueCalculators>
    <countEventsUntil id="AssigneeChanges">
      <event>
        <transition field="assignee" />
      </event>
      <until>
        <transition field="status">
          <to>CLOSED</to>
        </transition>
      </until>
    </countEventsUntil>
  </valueCalculators>
  <evaluationTimePeriod>
    <timePeriod>
      <start>2007-01-01</start>
      <end>  2009-12-31</end>
    </timePeriod>
  </evaluationTimePeriod>
  <timePeriodGranularity>
    <month />
  </timePeriodGranularity>
  <fixedFields />
</metric>
```

Abbildung 4.8.: Beispiel einer ITMS Metrik-Spezifikation: Durchschnittliche Anzahl der Bearbeiterwechsel eines Änderungsantrags

4.6. Formale Semantik von ITMS

Um die Semantik einer Sprache, also die Bedeutung von Programmen in dieser Sprache, festzulegen, gibt es die folgenden Möglichkeiten [Kle08]:

- **Denotationell** durch Angabe mathematischer Konstrukte, welche die Bedeutung eines Programms in der Sprache repräsentieren,
- **Operationell** durch Angabe, wie ein Programm als eine Folge von Berechnungsschritten interpretiert wird, beispielsweise gegeben durch einen Zustandsautomaten,
- **Translational** durch Überführung des Programms in ein Programm einer anderen Sprache, welche gut verstanden ist, das heißt für die eine Semantik spezifiziert ist,
- **Pragmatisch** durch Bereitstellung eines Werkzeugs, welches das Programm ausführt, also einer Referenzimplementierung.

Durch die QMetric Tool-Suite ist eine Referenzimplementierung für die Sprache ITMS gegeben. Für den Anwender bietet zudem die Benutzerdokumentation eine Einführung in die Semantik der Sprache. Eine solche natürlichsprachliche Beschreibung kann natürlich nicht die Präzision einer formalen Beschreibung der Semantik erreichen. Das alleinige Vorliegen einer Referenzimplementierung ist ebenfalls unbefriedigend, da sich Fragen zur Semantik oft nur durch Nachstellen von Testfällen klären lassen. Daher sollte es unabhängig von der Referenzimplementierung eine präzise Definition der Semantik geben.

Bei einer translationalen Semantik muss für jede ITMS Metrik-Spezifikation ein äquivalentes Programm in einer anderen Zielsprache angegeben werden. Die Metrik-Spezifikationen sind vom Abstraktionsniveau relativ weit von den detaillierten Berechnungsschritten entfernt. Eine Transformation in eine Abfragesprache wie SQL oder in eine Programmiersprache würde in einem wesentlich längeren und komplexeren Programm resultieren, was wenig hilfreich zur Klärung von Fragestellungen zur Semantik ist. Daher wird dieser Ansatz hier nicht weiter verfolgt.

Eine operationelle Semantik beschreibt, wie eine abstrakte Maschine sich bei Ausführung des Programms verhält. Allerdings gibt es keine nahe liegende abstrakte Maschine zur Beschreibung der in ITMS spezifizierten Metrikberechnungen.

Der denotatielle Ansatz bietet sich an, da sich die zu Grunde liegende Datenbank der Änderungsanträge, als auch die Ergebnisse der Metrikberechnungen gut als mathematische Konstrukte formalisieren lassen. Im Folgenden wird also eine denotationelle Semantik für ITMS vorgestellt.

4.6.1. Überblick

Die denotationelle Semantik einer Metrik-Spezifikation in der Sprache ITMS wird im Folgenden beschrieben als eine Reihe von Funktionen, die den Inhalt einer Datenbank mit Änderungsanträgen D auf eine Zeitreihe als Ergebnis der Metrikberechnung abbilden. Einen Überblick über die verwendeten Funktionen gibt Abbildung 4.9. Eine präzise Beschreibung der verwendeten Notation wird in den nächsten Abschnitten gegeben.

Eine Metrik wird berechnet in Bezug auf eine Zeitpartition T, welche die Aufteilung eines Zeitraums in Zeitperioden vorgibt. Zunächst wird eine Bewertungsfunktion v angewendet. Diese Funktion entspricht einer der vier Kategorien von Bewertungsverfahren aus Abschnitt 4.4.2.4. Mit Hilfe dieser Funktion werden für jede Zeitperiode Bewertungen von Änderungsanträgen ermittelt. Es ergibt sich somit eine Zeitreihe mit Mengen von Bewertungen für Änderungsanträge. Eine Gruppenwertberechnung g fasst dann die für eine Zeitperiode erstellten Bewertungen zusammen, beispielsweise durch Berechnung der Summe oder des Maximums aller Bewertungszahlen. Das Ergebnis ist eine Zeitreihe mit reellen Zahlen. In einer Metrik-Spezifikation kann die Berechnung mehrerer solcher Zeitreihen beschrieben werden. Diese können dann weiter mit arithmetischen Operationen verknüpft werden, um schließlich wieder eine Zeitreihe als Endergebnis der Metrikberechnung zu erhalten.

4.6.2. Grundlegende Definitionen

4.6.2.1. Zeit und Zeitreihen

Für die Berechnung werden ausschließlich diskrete Zeitpunkte betrachtet. Somit ist die **Menge der Zeitpunkte** gegeben als $\mathfrak{T} = \mathbb{N}$, und ein einzelner Zeitpunkt als $t \in \mathfrak{T}$. Ein **Zeitintervall** wird angegeben als $\tau = [t_1, t_2) = \{t | t_1 \leq t < t_2\}$.

Die Menge aller Zeitintervalle aus $\mathfrak{T}$ sei bezeichnet mit $\Theta(\mathfrak{T})$.

Definition 4.1 $T = (\tau_1, ...\tau_n)$ ist eine **vollständige Partition** des Zeitintervalls $\tau = [t_1, t'_n)$ genau dann, wenn gilt $\tau_1 = [t_1, t_2)$, $\tau_2 = [t_2, t_3)$, ..., $\tau_n = [t_n, t'_n)$, und $t_1 < t_2 < \ldots < t_n < t'_n$.

Die Menge aller möglichen vollständigen Partitionen von Zeitintervallen aus $\mathfrak{T}$ sei bezeichnet mit $\Phi(\mathfrak{T})$.

Definition 4.2 Sei V eine Menge von Werten und $T = \{\tau_1, \tau_2, ..., \tau_n\}$ eine vollständige Partition eines Zeitintervalls $\tau = [t_1, t'_n)$. Dann nennen wir einen Vektor $\zeta = ((v_1, \tau_1), \ldots, (v_n, \tau_n))$ mit $v_i \in V$ eine **Zeitreihe** von V bezüglich T.

Im Folgenden wird für Zeitreihen eine kürzere Schreibweise mit der Zeitpartition als Index verwendet werden: $\zeta = (v_1, ..., v_n)_T$.

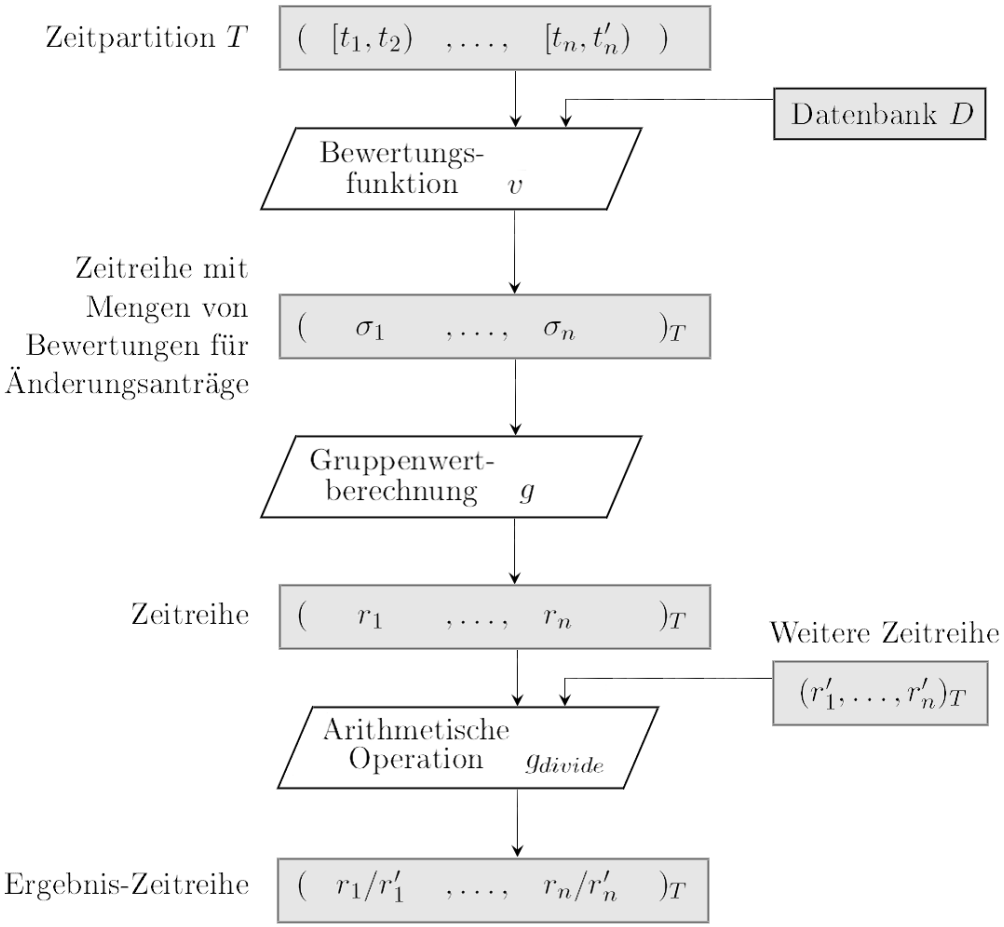


Abbildung 4.9.: Überblick der verwendeten Funktionen zur Beschreibung der Semantik

Die **Menge aller Zeitreihen von V bezüglich T** bezeichnen wir mit $Z(V,T)$.

Die **Menge aller Zeitreihen von V bezüglich vollständigen Partitionen von Zeitintervallen aus** $\mathfrak{T}$ bezeichnen wir mit $\mathfrak{Z}(V,\mathfrak{T}) = \bigcup_{T\in\Phi(\mathfrak{T})} Z(V,T)$.

4.6.2.2. Änderungsanträge

Ausgangspunkt für die Metrikauswertungen mit ITMS sind die Änderungsanträge als zu vermessende Entitäten. In diesem Abschnitt soll die Struktur der betrachteten Änderungsanträge formalisiert werden. Die **Menge der Änderungsanträge** sei gegeben als $C = \{c_1, ..., c_n\}$.

Alle Änderungsanträge haben bestimmte Attribute, wie beispielsweise Status oder Priorität. Für alle Änderungsanträge werden dieselben Attribute erfasst. Die **Menge aller Attribute** sei gegeben als $A = \{a_1, ..., a_n\}$. Die Menge der zulässigen Werte eines Attributs a sei gegeben als W_a.

Definition 4.3 Das Attribut, welches den **Erstellungszeitpunkt** eines Änderungsantrags repräsentiert, wird bezeichnet mit $a_{creation} \in A$, und es gilt $W_{a_{creation}} = \mathfrak{T}$. Jeder Änderungsantrag c hat einen eindeutigen Erstellungszeitpunkt, gegeben durch die Funktion $t_{creation}(c)$.

Definition 4.4 Der Zustand eines Änderungsantrags zu einem Zeitpunkt t ist durch die Belegung seiner Attribute gegeben. Die Belegung eines Attributs für einen gegebenen Änderungsantrag und einen gegebenen Zeitpunkt sei definiert durch eine **Zustandsfunktion** $s : C \times \mathfrak{T} \times A \to \bigcup_{a\in A} W_a \cup \{undefined\}$.

Die Zustandsfunktion bildet in die Menge der zulässigen Werte des Attributs oder auf das Symbol $undefined$ ab. Das Symbol $undefined$ wird verwendet, um zu kennzeichnen, dass ein Änderungsantrag zu einem gegebenen Zeitpunkt für ein Attribut a keinen definierten Wert besitzt. Dies ist der Fall, wenn der Änderungsantrag zu diesem Zeitpunkt noch nicht erstellt ist. Es gilt: $s(c,t,a) \in W_a \cup \{undefined\}$ für alle $a \in A, c \in C, t \in \mathfrak{T}$.

Definition 4.5 $D = (C, A, s)$ mit einer Menge von Änderungsanträgen C, einer Menge von Attributwerten A und einer Zustandsfunktion s, ist eine **Datenbank von Änderungsanträgen**, wenn gilt:

$$\begin{aligned} a_{creation} &\in A \\ s(c,t,a_{creation}) &= \begin{cases} t_{creation}(c) & t \geq t_{creation}(c) \\ undefined & \text{sonst} \end{cases} \\ s(c,t,a) &= undefined \;\; \forall t \in \mathfrak{T}, t < t_{creation}(c) \end{aligned}$$

Es gibt also ein Attribut $a_{creation}$ für den Erstellungszeitpunkt der Änderungsanträge. Jeder Änderungsantrag c hat einen definierten Erstellungszeitpunkt, gegeben durch die Funktion $t_{creation}(c)$. Die Zustandsfunktion s liefert für jeden Zeitpunkt vor dem Erstellungszeitpunkt für alle Attribute den Wert $undefined$.

Die Menge aller möglichen Datenbanken von Änderungsanträgen sei bezeichnet mit $\mathfrak{D}$.

Definition 4.6 Änderungsanträgen wird zu einem oder mehreren Zeitpunkten eine reelle Zahl zugeordnet, die ein Zwischenergebnis der Metrikberechnung enthält. Eine einzelne **Bewertung** u ist ein Tripel aus einem Änderungsantrag c, einer Bewertungszahl r und dem Zeitpunkt t. Es gilt also:
$u = (c, r, t) \in C \times \mathbb{R} \times \mathfrak{T}$.

4.6.3. Filter

Ein zentrales Konzept von ITMS sind Filter. Ein Filter spezifiziert Bedingungen, denen eine Entität genügen muss. Im Folgenden werden Filter formalisiert als Funktionen, die in den Wertebereich $\{0, 1\}$ abbilden. Dabei bedeutet 1, dass die abgebildete Entität den im Filter spezifizierten Bedingungen entspricht. Es gibt zwei Arten von Filtern: Filter für den Zustand eines Änderungsantrags, sowie Filter für Ereignisse im Lebenszyklus eines Änderungsantrags.

4.6.3.1. Zustandsfilter

Ein Zustandsfilter spezifiziert, ob ein Änderungsantrag aus einer Datenbank zu einem gegebenen Zeitpunkt eine bestimmte Bedingung erfüllt.

Definition 4.7 Ein **Zustandsfilter** ist eine Funktion der Form $f : \mathfrak{D} \times C \times \mathfrak{T} \rightarrow \{0, 1\}$.

Die Menge aller Zustandsfilter sei bezeichnet mit $\mathfrak{F} = \mathit{Abb}(\mathfrak{D} \times C \times \mathfrak{T}, \{0, 1\})$.

Übliche Zustandsfilter sind der Null-Filter und der Attributwert-Filter.

Der **Null-Filter** akzeptiert jeden Änderungsantrag, der zu dem gegebenen Zeitpunkt bereits existiert.

$$f_{null}(D, c, t) \quad = \quad \begin{cases} 1 & t \geq t_{creation}(c) \\ 0 & sonst \end{cases}$$

Der **Attributwert-Filter** dient dazu, nur solche Änderungsanträge auszuwählen, bei denen ein Attribut a zum gegebenen Zeitpunkt einen festgelegten Wert x annimmt.

$$f_{a,x}(D, c, t) = \begin{cases} 1 & s(c, t, a) = x \\ 0 & sonst \end{cases}$$

Die vorher genannten Filter werden als atomare Zustandsfilter bezeichnet. Ein atomarer Zustandsfilter kann mit dem Operator $\neg$ negiert werden. Es gilt:

$$\neg f(D,c,t) = \begin{cases} 1 & f(D,c,t) = 0 \\ 0 & f(D,c,t) = 1 \end{cases}$$

Weiterhin können Filter mit den logischen Operatoren $\wedge$ und $\vee$ verknüpft werden. Es gilt:

$$\begin{aligned} (f_1 \wedge f_2)(D,c,t) &= min(f_1(D,c,t), f_2(D,c,t)) \\ (f_1 \vee f_2)(D,c,t) &= max(f_1(D,c,t), f_2(D,c,t)) \end{aligned}$$

4.6.3.2. Ereignisfilter

Der Ereignisfilter spezifiziert, ob zu einem gegebenen Zeitpunkt t ein bestimmtes Ereignis im Lebenszyklus eines Änderungsantrags eingetreten ist.

Definition 4.8 Ein **Ereignisfilter** ist eine Funktion der Form $e : \mathfrak{D} \times C \times \mathfrak{T} \rightarrow \{0,1\}$

Wie zu sehen ist, haben Ereignisfilter die gleiche Signatur wie Zustandsfilter. Ereignisfilter haben allerdings konzeptionell eine andere Rolle, wie in Abschnitt 4.6.5 beschrieben. Folgende Ereignisfilter werden häufig eingesetzt:

Der **Erzeugung-Filter** gibt an, ob ein Änderungsantrag zu dem gegebenen Zeitpunkt erstellt wurde:

$$e_{creation}(D,c,t) = \begin{cases} 1 & t = t_{creation}(c) \\ 0 & \text{sonst} \end{cases}$$

Der **Zeitperioden-Ende-Filter** liefert genau dann den Wert 1, wenn der betrachtete Zeitpunkt t am Ende einer in der gegebenen Zeitpartition T enthaltenen Zeitperiode liegt. Für eine gegebene Zeitpartition $T = ([t_1,t_2),[t_2,t_3),\ldots,[t_n,t'_n))$ gilt also:

$$e_{endOfTimePeriod,T}(D,c,t) = \begin{cases} 1 & t \in \{t_2,t_3,\ldots,t_n,t'_n\} \\ 0 & \text{sonst} \end{cases}$$

Der **Zustandsänderung-Filter** liefert genau dann den Wert 1, wenn sich für ein gegebenes Attribut a zum Zeitpunkt $t+1$ der Wert geändert hat:

$$e_{transition,a}(D,c,t) = \begin{cases} 1 & s(c,t,a) \neq s(c,t+1,a) \text{ und } t > t_{creation}(c) \\ 0 & \text{sonst} \end{cases}$$

Analog dazu kann auch ein Zustandsänderung-Filter definiert werden, der zusätzlich nach dem für das Attribut gesetzten Wert vor oder nach der Zustandsänderung filtert.

4.6.4. Metrik

Im Folgenden wird die Semantik einer Metrik formal beschrieben. Die Beschreibung beschränkt sich hier auf die Hauptelemente der Metrik-Spezifikation. In Abschnitt 4.6.6 wird erläutert, wie die Definition der Semantik für weitere Elemente der Metrik-Spezifikation, wie etwa die Gruppierung, durchgeführt werden kann.

Definition 4.9 Eine **Metrik** $M = (f, T, b)$ auf einer Datenbank von Änderungsanträgen $D \in \mathfrak{D}$ besteht aus

- einem als Basisfilter bezeichneten Zustandsfilter $f \in \mathfrak{F}$,
- einer vollständigen Zeitpartition T,
- und einer Auswertungsfunktion $b : \mathfrak{D} \times \mathfrak{F} \times \Phi(\mathfrak{T}) \to \mathfrak{Z}(\mathbb{R}, \mathfrak{T})$.

Die Auswertungsfunktion b bildet von der Datenbank der Änderungsanträge, dem Basisfilter und einer gegebenen vollständigen Zeitpartition T auf eine Zeitreihe von reellen Zahlen bezüglich T ab. Es gilt $b(D, f, T) \in Z(\mathbb{R}, T)$.

Die **Auswertungsfunktion** b wird mit Hilfe der folgenden Funktionen formal beschrieben:

Die **Bewertungsfunktion** v bildet von der Datenbank der Änderungsanträge, dem Basisfilter und einem gegebenen Zeitintervall aus $\mathfrak{T}$ auf eine Zeitreihe ab. Diese Zeitreihe enthält genau einen Eintrag für eine Zeitperiode. Der Wert dieses Eintrags ist eine Menge von Bewertungen für Änderungsanträge:

$$v : \mathfrak{D} \times \mathfrak{F} \times \Theta(\mathfrak{T}) \to \underbrace{\mathfrak{Z}(\underbrace{\underbrace{\mathcal{P}(C \times \mathbb{R} \times \mathfrak{T})}_{\text{Menge von Bewertungen für Änderungsanträge}}}_{\text{Zeitreihe mit Mengen von Bewertungen für Änderungsanträge}}, \mathfrak{T})}$$

Die resultierende Zeitreihe wird mit Hilfe einer atomaren **Gruppenwertberechnung** g in eine Zeitreihe mit reellen Zahlen transformiert.

$$g : \mathfrak{Z}(\mathcal{P}(C \times \mathbb{R} \times \mathfrak{T}), \mathfrak{T}) \to \mathfrak{Z}(\mathbb{R}, \mathfrak{T})$$

Die Ergebnisse von Gruppenwertberechnungen können wiederum mit Hilfe arithmetischer Operationen kombiniert werden.

Im Folgenden sollen nun zunächst die Gruppenwertberechnungen und dann die möglichen Bewertungsfunktionen formal beschrieben werden.

4.6.4.1. Gruppenwertberechnungen

Es gibt zwei Arten von Gruppenwertberechnungen. Zunächst betrachten wir atomare Gruppenwertberechnungen. Diese wandeln eine Zeitreihe mit Bewertungen für Änderungsanträge in eine Zeitreihe von reellen Zahlen um. Im Folgenden seien beispielhaft einige der zur Verfügung stehenden **atomaren Gruppenwertberechnungen** beschrieben. Die **Menge aller Bewertungen in einer Zeitperiode** i bei einer Metrikberechnung sei bezeichnet mit $\sigma_i \in \mathcal{P}(C \times \mathbb{R} \times \mathfrak{T})$.

$$\begin{aligned} g_{count,T}((\sigma_1,\ldots,\sigma_n)_T) &= (|\sigma_1|,\ldots,|\sigma_n|)_T \\ g_{sum,T}((\sigma_1,\ldots,\sigma_n)_T) &= (\sum_{(c,r,t)\in\sigma_1} a,\ldots,\sum_{(c,r,t)\in\sigma_n} a)_T \\ g_{max,T}((\sigma_1,\ldots,\sigma_n)_T) &= (\max_{(c,r,t)\in\sigma_1} a,\ldots,\max_{(c,r,t)\in\sigma_n} a)_T \end{aligned}$$

Analog dazu können weitere atomare Gruppenwertberechnungen wie beispielsweise Minimum, Maximum oder Median definiert werden.

Die aus Gruppenwertberechnungen resultierenden Zeitreihen können mit arithmetischen Operationen in der folgenden Form verknüpft werden:

$$g : \mathfrak{Z}(\mathbb{R},\mathfrak{T}) \times \mathfrak{Z}(\mathbb{R},\mathfrak{T}) \rightarrow \mathfrak{Z}(\mathbb{R},\mathfrak{T})$$

$$\begin{aligned} g_{add}((r_1,\ldots,r_n)_T,(r'_1,\ldots,r'_n)_T) &= (r_1+r'_1,\ldots,r_n+r'_n)_T \\ g_{subtract}((r_1,\ldots,r_n)_T,(r'_1,\ldots,r'_n)_T) &= (r_1-r'_1,\ldots,r_n-r'_n)_T \\ g_{multiply}((r_1,\ldots,r_n)_T,(r'_1,\ldots,r'_n)_T) &= (r_1*r'_1,\ldots,r_n*r'_n)_T \\ g_{divide}((r_1,\ldots,r_n)_T,(r'_1,\ldots,r'_n)_T) &= (r_1/r'_1,\ldots,r_n/r'_n)_T \end{aligned}$$

Des Weiteren können Zeitreihen mit reellen Zahlen addiert, subtrahiert, multipliziert und dividiert werden. Es gilt für alle Operationen jeweils analog:

$$r \odot (r_1,\ldots,r_n)_T = (r \odot r_1,\ldots,r \odot r_n)_T \ , \odot \in \{+,-,*,/\}$$

4.6.4.2. Gewichtungsfunktion

Für das Bewertungsverfahren „Zählen von Ereignissen in einer Zeitperiode“ wird eine Gewichtungsfunktion verwendet. Eine Gewichtungsfunktion w liefert für einen gegebenen Änderungsantrag c aus einer Datenbank D und einen gegebenen Zeitpunkt t eine reelle Zahl zurück:

$$w : \mathfrak{D} \times C \times \mathfrak{T} \rightarrow \mathbb{R}$$

Häufig verwendete Gewichtungsfunktionen sind die Standardgewichtung mit dem Wert 1, die Gewichtung nach dem Alter des Änderungsantrags, oder die

Gewichtung auf Basis eines Attributs des Änderungsantrags. Dazu seien hier beispielhaft die Definitionen angegeben:

$$
\begin{aligned}
w_{Default}(D,c,t) &= 1 \\
w_{Mapping,a}(D,c,t) &= map(s(c,t,a)) \\
&\quad \text{mit einer benutzerdefinierten Abbildung} \\
&\quad map : W_a \to \mathbb{R} \\
w_{Age}(D,c,t) &= \begin{cases} timeBetween(t_{creation}(c),t) & t \geq t_{creation}(c) \\ 0 & \text{sonst} \end{cases}
\end{aligned}
$$

Die folgende Hilfsfunktion liefert die Länge des Zeitintervalls zwischen zwei gegebenen Zeitpunkten $t_1 \in \mathfrak{T}$ und $t_2 \in \mathfrak{T}$:

$$
\begin{aligned}
timeBetween : \mathfrak{T} \times \mathfrak{T} &\to \mathbb{N} \\
timeBetween(t_1,t_2) &= |t_1 - t_2|
\end{aligned}
$$

4.6.5. Bewertungsverfahren

Im Folgenden werden die in Abschnitt 4.4.2.4 beschriebenen vier Kategorien von Bewertungsverfahren formal beschrieben. Dazu wird das in Abschnitt 4.6.4 eingeführte Konzept der Bewertungsfunktion verwendet. Allgemein hat eine Bewertungsfunktion die folgende Form:

$$
v_{Kategorie,Parameter} : \mathfrak{D} \times \mathfrak{F} \times \Theta(\mathfrak{T}) \to \underbrace{\mathcal{P}(C \times \mathbb{R} \times \mathfrak{T})}_{\text{Menge von Bewertungen für Änderungsanträge}}
$$

Jede Bewertungsfunktion wird mit verschiedenen Parametern genauer bestimmt. Diese werden im Folgenden als Index an der entsprechenden Funktion notiert.

Des Weiteren tritt bei den Definitionen häufiger die *und*-Verknüpfung von Filtern auf, da typischerweise die als Parameter des Bewertungsverfahrens gegebenen Filter mit dem Basisfilter aus der Metrik verknüpft werden.

4.6.5.1. Zählen von Ereignissen in einer Zeitperiode

Definition 4.10 Für einen Ereignisfilter e und eine Gewichtungsfunktion w sei die Bewertungsfunktion zum Zählen von Ereignissen in einer Zeitperiode definiert als:

$$
v_{CountEvents,e,w}(D,f,\tau) = \left\{ (c,r,t) \;\middle|\; \underbrace{t \in \tau \wedge (f \wedge e)(D,c,t) = 1}_{\text{Ein zu } e \text{ konformes Ereignis passierte zum Zeitpunkt } t \in \tau} \wedge r = w(D,c,t) \right\}
$$

Es wird also jeweils eine Bewertung vorgenommen, wenn für ein Änderungsantrag in dem gegebenen Zeitintervall dem Basisfilter f entspricht, und das im Ereignisfilter e gegebene Ereignis auftrat. Die Bewertungszahl wird durch die Gewichtungsfunktion w bestimmt.

4.6.5.2. Zählen von Ereignissen bis zu einem Ereignis

Definition 4.11 Die Bewertungsfunktion zum Zählen von Ereignissen entsprechend eines Ereignisfilters e_{Count} bis zu dem Auftreten eines Ereignisses, das dem Ereignisfilter e_{Until} entspricht, sei definiert als:

$$v_{CountUntil,e_{Count},e_{Until}}(D,f,\tau) = \Big\{ \quad (c,r,t) \quad \Big|$$

$$\underbrace{t \in \tau \ \wedge \ (f \wedge e_{Until})(D,c,t) = 1}_{\text{Ein } e_{Until} \text{ Ereignis passierte zum Zeitpunkt } t \in \tau} \qquad \wedge$$

$$\underbrace{\Big[\nexists t' \in \mathfrak{T} \text{ mit } t' < t \ \wedge \ (f \wedge e_{Until})(D,c,t') = 1\Big]}_{t \text{ ist der früheste Zeitpunkt an dem ein } e_{Until} \text{ Ereignis auftrat}} \qquad \wedge$$

$$r = \underbrace{|\{t^* \in \mathbb{N} \mid t^* < t \ \wedge \ (f \wedge e_{Count})(D,c,t^*) = 1\}|}_{\text{Anzahl des Auftretens des } e_{Count} \text{ Ereignisses}} \Big\}$$

Es wird also jeweils eine Bewertung vorgenommen, wenn das e_{Until} Ereignis zum ersten Mal im Lebenslauf eines Änderungsantrags auftritt. In der Bewertungszahl wird zurückgegeben, wie oft das e_{Count} Ereignis im Lebenslauf des Änderungsantrags bisher auftrat.

4.6.5.3. Intervalllänge

Für die Bewertungsfunktionen „Intervalllänge“ und „Verweilzeit“ wird ein Parameter z verwendet, der angibt zu welchem Zeitpunkt eine Bewertungszahl erzeugt wird. Mögliche Werte sind $z \in \{firstTime, eachTime, lastTime\}$.

Eine weitere Hilfsfunktion, die in der folgenden Definition verwendet wird, liefert für einen gegebenen Änderungsantrag c aus einer Datenbank D und einem Ereignisfilter e die Menge der Zeitpunkte, an denen das Ereignis auftrat. Es gilt:

$$\begin{aligned} m : \mathfrak{D} \times C \times \mathfrak{F} &\rightarrow \mathcal{P}(\mathfrak{T}) \\ m(D,c,e) &= \{t \in \mathfrak{T} \mid e(D,c,t) = 1\} \end{aligned}$$

Definition 4.12 Die Bewertungsfunktion zur Bestimmung der Intervalllänge von einem Ereignis, das dem Ereignisfilters e_{From} entspricht, bis zu dem Auftreten eines Ereignisses, das dem Ereignisfilter e_{To} entspricht, mit Zählung zu den durch $z \in \{firstTime, eachTime, lastTime\}$ definierten Zeitpunkten, sei definiert als:

$$v_{IntervalLength,e_{From},e_{To},z}(D,f,\tau) = \Big\{ \quad (c,r,t) \quad \Big|$$

$$\underbrace{t \in \tau \ \wedge \ (f \wedge e_{To})(D,c,t) = 1}_{\text{Ein } e_{To} \text{ Ereignis passierte zum Zeitpunkt } t \in \tau} \quad \wedge$$

$$\underbrace{\{t' \mid t' < t \ \wedge \ t' \in m(D,c,f \wedge e_{From})\} \neq \emptyset}_{\text{Ein } e_{From} \text{ Ereignis ist vor } e_{To} \text{ aufgetreten.}} \quad \wedge$$

$$\Big[\quad \underbrace{(z = eachTime)}_{\text{Jedes Auftreten von } e_{To} \text{berücksichtigen}}$$

$$\vee \quad \underbrace{(z = firstTime \ \wedge \ \nexists t^* \in \mathfrak{T} \text{ mit } t^* < t \ \wedge \ (f \wedge e_{To})(D,c,t^*) = 1)}_{\text{Nur das erste Auftreten von } e_{To} \text{ berücksichtigen}}$$

$$\vee \quad \underbrace{(z = lastTime \ \wedge \ \nexists t^* \in \mathfrak{T} \text{ mit } t^* > t \ \wedge \ (f \wedge e_{To})(D,c,t^*) = 1)}_{\text{Nur das letzte Auftreten von } e_{To} \text{ berücksichtigen}} \quad \Big] \quad \wedge$$

$$r = \underbrace{timeBetween(max\{t' | t' < t \ \wedge \ t' \in f \wedge m(D,c,e_{From})\}, t)}_{\text{Intervalllänge zwischen dem Auftreten von } e_{From} \text{ und } e_{To}} \Big\}$$

Es muss also zunächst das Ereignis e_{From} im Lebenslauf eines Änderungsantrags auftreten. Eine Bewertung wird dann beim Auftreten des Ereignisses e_{To} vorgenommen. Von dem Parameter z hängt ab, ob jedes Auftreten, nur das erste oder nur das letzte zeitliche Auftreten berücksichtigt wird. Die Bewertungszahl ist durch den Zeitabstand zwischen dem Auftreten der beiden Ereignisse bestimmt.

4.6.5.4. Verweilzeit

Die folgende Funktion j dient dazu, für einen gegebenen Änderungsantrag c aus einer Datenbank D, einem Zeitpunkt t und einem Zustandsfilter f die Gesamtzeit zu bestimmen, in der sich der Änderungsantrag c vor dem Zeitpunkt t in einem f entsprechenden Zustand befand. Es gilt:

$$\begin{aligned} j : \mathfrak{D} \times c \times \mathfrak{T} \times \mathfrak{F} &\rightarrow \mathbb{N} \\ j(D,c,t,f) &= |\{t' \mid t' < t \ \wedge \ f(D,c,t) = 1\}| \end{aligned}$$

In der folgenden Definition wird weiterhin der in Abschnitt 4.6.5.3 eingeführte Parameter z, sowie die Funktion m verwendet.

Definition 4.13 Die Bewertungsfunktion zur Bestimmung der Verweilzeit in einem dem Zustandsfilter f_{State} entsprechenden Zustand, bis zu dem Auftreten eines Ereignisses, das dem Ereignisfilter e_{Until} entspricht, mit Zählung zu den durch $z \in \{firstTime, eachTime, lastTime\}$ definierten Zeitpunkten, sei definiert als:

$$v_{ResidenceTime,f_{State},e_{Until},z}(D,f,\tau) = \Big\{ \quad (c,r,t) \quad \Big|$$

$$\underbrace{t \in \tau \ \wedge \ (f \wedge e_{Until})(D,c,t) = 1}_{\text{Ein } e_{Until} \text{ Ereignis passierte zum Zeitpunkt } t \in \tau} \qquad \wedge$$

$$\Big[\qquad \underbrace{(z = eachTime)}_{\text{Jedes Auftreten von } e_{Until} \text{berücksichtigen}}$$

$$\vee \quad \underbrace{(z = firstTime \ \wedge \ \nexists t^* \in \mathfrak{T} \text{ mit } t^* < t \ \wedge \ (f \wedge e_{To})(D,c,t^*) = 1)}_{\text{Nur das erste Auftreten von } e_{Until} \text{ berücksichtigen}}$$

$$\vee \quad \underbrace{(z = lastTime \ \wedge \ \nexists t^* \in \mathfrak{T} \text{ mit } t^* > t \ \wedge \ (f \wedge e_{To})(D,c,t^*) = 1)}_{\text{Nur das letzte Auftreten von } e_{Until} \text{ berücksichtigen}} \quad \Big] \quad \wedge$$

$$r = \qquad \underbrace{j(D,c,t,f \wedge f_{State})}_{\text{Verweilzeit von } c \text{ in dem gefragten Zustand}} \quad \Big\}$$

Es wird also eine Bewertung vorgenommen, wenn das Ereignis e_{Until} im Lebenslauf eines Änderungsantrags auftritt. Von dem Parameter z hängt wiederum ab, ob jedes Auftreten, nur das erste oder nur das letzte zeitliche Auftreten berücksichtigt wird. Die Bewertungszahl wird schließlich dadurch bestimmt, wie lange sich der Änderungsantrag vor dem Auftreten des Ereignisses e_{Until} insgesamt in einem dem Zustandsfilter f_{State} entsprechenden Zustand befand.

4.6.6. Weitere Elemente der Metrik-Spezifikation

Zwei der in Abschnitt 4.5 eingeführten Elemente einer Metrik-Spezifikation wurden bei der Definition der Semantik bisher noch nicht berücksichtigt. Diese sind die Gruppierung nach einem Attribut und die fixierten Attribute. Die Semantik dieser Elemente kann auf einfache Weise auf die bisher gegebenen Definitionen zurückgeführt werden.

Die Berechnung einer Metrik $M = (f, T, b)$ mit Gruppierung nach einem Attribut a liefert für jede mögliche Belegung von a eine Zeitreihe, also insgesamt $|W_a|$ Zeitreihen. Die einzelne Zeitreihe für einen möglichen Attributwert $x \in W_f$ kann als Ergebnis der Metrik $M_x = (f \wedge f_{a,x}, T, b)$ ermittelt werden.

Die Berechnung einer Metrik $M = (f, t, b)$ auf einer Datenbank $D = (C, A, s)$ mit einem fixierten Attribut a^* zu einem Zeitpunkt t^* kann dargestellt werden durch Anpassung der Zustandsfunktion von D. Es gelte:

$$s_{a^*}(c,t,a) = \begin{cases} s(c,t,a) & a \neq a^* \\ s(c,t^*,a) & a = a^* \end{cases}$$

Das Ergebnis der Metrikberechnung ergibt sich dann durch Anwendung der Metrik M auf die Datenbank $D^* = (C, A, s_{a^*})$.

4.6.7. Beispiel

Als Beispiel ist hier die Definition der Auswertungsfunktion für die in Abschnitt 4.5.4 auf Seite 68 vorgestellte Metrik-Spezifikation angegeben. Die Metrik berechnet die durchschnittliche Anzahl der Bearbeiterwechsel eines Änderungsantrags zu dem Zeitpunkt, an dem dieser geschlossen wird.

Hilfsfunktionen werden mit dem Kürzel *ac* benannt (*Assignee Changes*). Es gibt eine Datenbank $D = (C, A, s)$ mit den Attributen $status, assigne \in A$ und dem Attributwert $CLOSED \in W_{status}$.

T sei eine vollständige Partition des Zeitraums vom 1.1.2007 bis zum 1.1.2009 in monatliche Zeitperioden. Die Metrik $M_{average\_ac}$ kann definiert werden als:

$$M_{average\_ac} = (f_{\text{product},1}, T, b_{average\_ac})$$

Als Basisfilter wird also der in Abschnitt 4.6.3.1 eingeführte Attributwert-Filter verwendet. In diesem Beispiel werden nur solche Änderungsanträge betrachtet, bei denen das Attribut *product* mit dem Wert 1 belegt ist. Die Auswertungsfunktion $b_{average\_ac}$ ist gegeben als:

$$b_{average\_ac}(D, f, T) = g_{divide}\Big(g_{sum,T}\big(ac(D, f, T)\big), g_{count,T}\big(ac(D, f, T)\big)\Big)$$

Die Hilfsfunktion *ac* liefert für eine Datenbank D, einen Basisfilter f und eine vollständige Zeitpartition T eine Zeitreihe. Diese Zeitreihe enthält für jede Zeitperiode eine Menge von Bewertungen für Änderungsanträge.

$$ac : \mathfrak{D} \times \mathfrak{F} \times \Theta(\mathfrak{T}) \quad \rightarrow \quad \underbrace{\mathfrak{Z}(\underbrace{\mathcal{P}(C \times \mathbb{R} \times \mathfrak{T})}_{\text{Menge von Bewertungen für Änderungsanträge}}, \mathfrak{T})}_{\text{Zeitreihe mit Mengen von Bewertungen für Änderungsanträge}}$$

$$ac(D, f, T) = \Big(\; v_{CountUntil_{e_{Count}, e_{Until}}}(D, f, \tau_1), \quad \ldots \quad , \; v_{CountUntil_{e_{Count}, e_{Until}}}(D, f, \tau_n) \; \Big)_T$$

Die beiden Ereignisfilter als Parameter der Bewertungsfunktion $v_{CountUntil}$ sind schließlich gegeben durch:

$$e_{Count} = e_{transition,assignee}$$

$$e_{Until}(D, c, t) = \begin{cases} 1 & e_{transition,status}(D, c, t) = 1 \quad \wedge \\ & s(c, t+1, status) = CLOSED \\ 0 & sonst \end{cases}$$

Die Ergebnis-Zeitreihe für die gegebene Metrik $M_{average\_ac} = (f_{\text{product},1}, T, b_{average\_ac})$ und eine Datenbank D berechnet sich dann als:

$$\zeta_{average\_ac} = b_{average\_ac}(D, f_{\text{product},1}, T)$$

4.6.8. Zusammenfassung

Die hier vorgestellte denotationelle Semantik von ITMS basiert auf der Formalisierung der Datenbank von Änderungsanträgen mit Hilfe der Zustandsfunktion. Die Semantik einer ITMS-Metrik wird durch verschiedene Arten von Funktionen beschrieben. Dazu gehören die Auswertungsfunktion, die Gruppenwertberechnung, die Bewertungsfunktion, die Gewichtungsfunktion sowie Zustands- und Ereignisfilter. Wie in Abschnitt 4.6.6 dargestellt, können alle Elemente der Sprache ITMS auf diese Funktionen zurückgeführt werden.

5. Entwicklung von Metriken

> During the process of stepwise refinement,
> a notation which is natural to the problem
> in hand should be used as long as possible.
>
> *(Niklaus Wirth, 1934-)*

Inhaltsangabe

In diesem Kapitel wird das Vorgehen zur Entwicklung von Metriken auf Software-Entwicklungsarchiven betrachtet.

Ansätze wie GQM bieten eine allgemeine Vorgehensweise zur zielgerichteten Definition von Metriken. Wie in Abschnitt 2.1.3 bereits geschildert, gibt es für die detaillierte Definition von Metriken und deren Erhebungsmethoden allerdings nur wenig Unterstützung [Sch07]. Die Sprache ITMS erleichtert es, das Vorgehen der Definition einer Metrik auf Software-Entwicklungsarchiven zu konkretisieren.

Zunächst ermöglicht ITMS eine kompakte und präzise Definition einer Metrik. Bei der genauen Festlegung der Details einer solchen Metrik bestehen nach unserer Erfahrung eine Reihe von Fallstricken [GSL07b, SL08]. Typische Beispiele sind:

- Relevante Ereignisse in Bezug auf die beabsichtigte Metrik werden nicht berücksichtigt.
- Es bleibt unberücksichtigt, dass ein Änderungsantrag durch Änderung von Attributen nicht mehr mit dem Basisfilter übereinstimmt, oder umgekehrt ab einem bestimmten Zeitpunkt dem Basisfilter entspricht.
- Unpassende Annahmen über die Verwendung des Status-Workflows werden zu Grunde gelegt.

Des Weiteren kommt es vor, dass in einem Issue-Tracking-System bestimmte Attribute oder Statuswerte inkonsistent verwendet werden. Auch über die Zeit kann sich die Verwendung von Attributen und Zuständen verändert haben. Zudem können neue Attribute eingeführt worden sein oder bestimmte Attribute und Statuswerte nicht mehr verwendet werden. Eine Metrik, die sich auf solche Attribute oder Statuswerte bezieht, liefert dann keine brauchbaren Ergebnisse.

Die vorgenannten Schwierigkeiten sind die Motivation für das in diesem Kapitel vorgestellte Verfahren zur Entwicklung von Metriken in ITMS. Das Verfahren ermöglicht eine schrittweise Validierung, um möglichst früh Probleme zu erkennen und die Metrik entsprechend anzupassen.

5.1. Validierung von Metriken

Der Begriff der Validierung einer Metrik wird in der Literatur unterschiedlich verwendet. Daher soll zunächst eingegrenzt werden, in welchem Sinne die Validierung in unserem Kontext zu verstehen ist.

Ganz allgemein zielt **Verifikation** auf die Bereitstellung eines objektiven Nachweises, dass ein Gegenstand gestellte Anforderungen erfüllt [ISO07c]. **Validierung** zielt auf die Bereitstellung eines objektiven Nachweises, dass ein Gegenstand die Anforderungen für einen spezifischen beabsichtigten Gebrauch erfüllt [ISO07c].

Bei der Entwicklung von Metriken liegt der Schwerpunkt auf der Validierung. Der Begriff der Validierung kann wiederum unterschiedlich verstanden werden. Jacquet und Abran unterscheiden hier zwischen [JA98]:

- **Validierung einer Metrik**: Hier wird überprüft, ob die Metrik das misst, was bei der Entwicklung der Metrik angenommen wird. Anders ausgedrückt handelt es sich um die Frage, ob die Metrik eine geeignete Beschreibung des zu messenden Attributs liefert [FP97]. Zuse bezeichnet dies als **interne Validierung** einer Metrik [Zus97].
- **Validierung der Anwendung einer Metrik**: Hier wird überprüft, ob ein Messergebnis für eine Metrik richtig ermittelt wurde. Dies ist insbesondere relevant für geschätzte Metriken, wie beispielsweise die *Function Point Schätzung* [MD96].
- **Validierung von Modellen zur Vorhersage**: Hier wird überprüft, ob ein Vorhersagemodell, welches auf Messwerte einer Metrik aufbaut, brauchbare Vorhersagen liefert. Zuse bezeichnet dies als **externe Validierung** einer Metrik [Zus97].

In dem im folgenden Abschnitt vorgestellten Verfahren wird die interne Validierung von Metriken betrachtet. Bei einer internen Validierung können wiederum verschiedene Kriterien geprüft werden. In der Regel steht die **Repräsentationsbedingung** im Vordergrund, also die Bedingung, dass die Messwerte die herrschenden empirischen Relationen adäquat wiedergeben [Fen94]. Daneben können aber auch weitere Kriterien herangezogen werden, um Metriken zu klassifizieren und zu vergleichen.

Ein erster Ansatz dazu wurde von Weyuker beschrieben, bezogen auf Metriken zur Bewertung von Komplexität von Software [Wey88]. Zu den von ihr benannten Kriterien gehören beispielsweise die Existenz von Äquivalenzklassen, oder die schwache Ordnung bezüglich der Verkettung von Messobjekten.

Schneidewind beschreibt ebenfalls eine Reihe von allgemeinen Kriterien sowie dazugehörige statistische Ansätze zu deren Validierung. Neben der Korrelation zwischen einem Messwert und dem damit zu messenden Merkmal (*Quality Factor*) sind dies Kriterien wie Konsistenz, Unterschiedlichkeit der Messwerte bei unterschiedlicher Merkmalsausprägung (*Discriminative Power*) und Zuverlässigkeit. Das Kriterium Konsistenz wird beispielsweise mathematisch ausgedrückt durch eine obere Schranke für den Korrelationskoeffizient der Messwerte zu den Ausprägungen des Qualitätsmerkmals.

Einige der von Weyuker und von Schneidewind beschriebenen Kriterien fanden Eingang in den IEEE Standard 1061 [IEE98]. Dieser Standard beschreibt hauptsächlich die externe Validierung. Es wird angenommen, dass die Erfüllung eines Qualitätsmerkmals durch eine sogenannte direkte Metrik gemessen werden kann, die nicht validiert werden muss. Als Beispiel einer direkten Metrik für das Qualitätsmerkmal „Zuverlässigkeit“ wird die mittlere Zeit bis zu einer Störung (*Mean Time To Failure*) genannt. Eine solche direkte Metrik kann für gewöhnlich nicht während der Softwareentwicklung gemessen werden. Bei der externen Validierung wird dann anhand von Stichproben die Korrelation von Vorhersagewerten zu den Ergebnissen der direkten Metrik durchgeführt.

Dieser Ansatz wurde von verschiedenen Autoren kritisiert. Fenton weist darauf hin, dass ein betrachtetes Qualitätsmerkmal, auf dessen Messung abgezielt wird, oft in sehr unterschiedliche Submerkmale zerfällt [Fen94]. Deswegen kann in einem solchen Fall keine geeignete direkte Metrik angegeben werden, da diese in Konflikt stehende Ziele erfüllen müsste. Kitchenham et al. beanstanden, dass bei einem solchen Vorgehen nur das Vorhersagemodell, nicht aber die Metrik selbst validiert wird [KPF95]. An gleicher Stelle werden Kriterien für gültige Metriken beschrieben. Diese Kriterien wurden wiederum als zu restriktiv kritisiert [MBW+97].

Kitchenham et al. unterscheiden folgende weitere Aspekte der Validierung [KPF95]:

- Eignung der Maßeinheit (*Unit Validity*)
- Eignung des Messwerkzeugs (*Instrument Validity*)
- Eignung der verwendeten Messvorschrift (*Protocol Validity*)

Im Folgenden liegt der Fokus auf der Eignung der verwendeten Messvorschrift. Durch den Einsatz der deklarativen Sprache zur Beschreibung der Metrik ist diese Beschreibung zugleich die ausführbare Messvorschrift. Näheres zur Eignung der für ITMS bereitgestellten Messwerkzeuge wird in Abschnitt 7.3 diskutiert.

Wichtig ist die Feststellung, dass eine Metrik kontinuierlich über den gesamten Messprozess validiert wird [Roc94, MIS95]. Nicht nur bei der Entwicklung einer Metrik, sondern auch bei ihrem späteren Einsatz werden immer wieder neue Erfahrungen gesammelt, die in der Definition oder bei der Interpretation der Metrik berücksichtigt werden müssen.

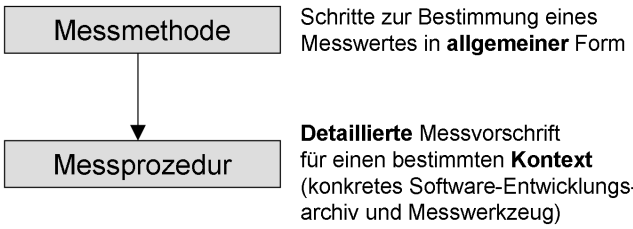


Abbildung 5.1.: Zusammenhang zwischen Messmethode und Messprozedur

Habra et. al. beschreiben diese kontinuierliche Validierung im Bezug auf drei Phasen im Messprozess: Entwurf einer Metrik, Anwendung einer Metrik und Verwertung der Messergebnisse [HALS08]. In der Praxis ist dies als iterativer Prozess anzusehen, da in jeder Phase Probleme erkannt werden können, die einen Rückgriff in eine frühere Phase notwendig machen. Beim Entwurf einer Metrik wird zwischen der Messmethode und der Messprozedur unterschieden.

- Die **Messmethode** beschreibt in allgemeiner Form die Schritte zur Bestimmung eines Messwerts [ISO07c]. Es kann weiter unterschieden werden in subjektive Messmethoden im Sinne einer systematischen Bewertung durch Experten, und objektive Messmethoden im Sinne von Zählungen oder Berechnungen auf dem Messobjekt.
- In der **Messprozedur** ist detailliert beschrieben, wie eine Messmethode konkret durchgeführt wird [ISO07c]. Die Messprozedur berücksichtigt also stärker den Kontext, in dem die Messmethode angewendet wird, und gibt gegebenenfalls Hinweise für die eingesetzten Messwerkzeuge [ISO07a].

Eine Messmethode kann durch eine strukturierte natürlichsprachliche Beschreibung angegeben werden. Weiterhin kann eine Messmethode auch bereits in der Sprache ITMS spezifiziert werden. Für die Messprozedur kann diese Spezifikation dann als Schablone verwendet werden. Dazu muss die Spezifikation an die konkrete Interpretation von Attributen und den Workflow-Zuständen in einer Organisation angepasst werden, und das Messwerkzeug an das konkret eingesetzte Issue-Tracking-System angebunden werden. Der Zusammenhang zwischen Messmethode und Messprozedur wird in Abbildung 5.1 verdeutlicht.

Validierung auf der Ebene der Messmethode bedeutet aus theoretischer Sicht die Überprüfung der Repräsentationsbedingung. Praktisch bedeutet dies die Überprüfung, ob die Messwerte der Metrik mit der Experteneinschätzung des zu vermessenden Attributs übereinstimmen [HALS08]. Dies kann experimentell untermauert oder widerlegt, aber nicht formal bewiesen werden [KPF95, Pop80].

Bei der Validierung auf der Ebene der Messprozedur muss überprüft werden, ob die Messmethode in dem gegebenen Kontext durch die Messprozedur korrekt implementiert ist. Probleme können beispielsweise daraus resultieren, dass die Arbeitsabläufe und Zustände im Issue-Tracking-System anders interpretiert werden, als bei der Definition der Messmethode angenommen, oder die Qualität der eingegebenen Daten nicht ausreichend ist.

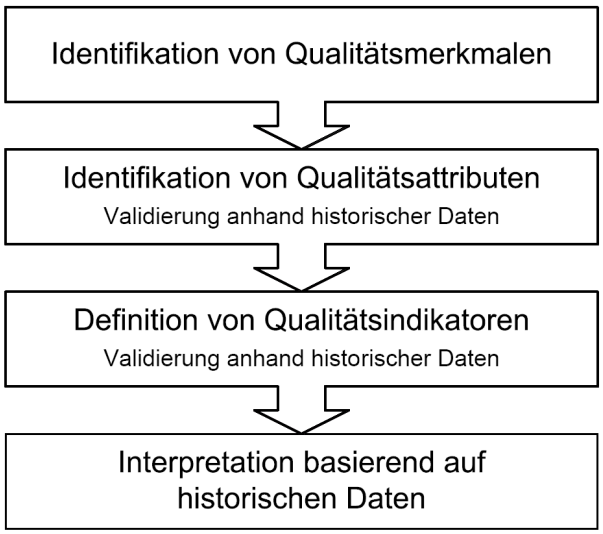


Abbildung 5.2.: Vorgehen zur Entwicklung von Metriken mit ITMS

5.2. Ein Vorgehen zur Entwicklung von Metriken

Der Einsatz der Sprache ITMS zur Definition von Metriken erleichtert es, während der Entwicklung einer Metrik Messwerte zu erheben. Wenn historische Daten im Issue-Tracking-System vorhanden sind, können mit der vorhandenen Werkzeugunterstützung Messergebnisse interaktiv abgerufen und hinab bis zur Detailebene der einzelnen Änderungsanträge untersucht werden. Diese Möglichkeit kann genutzt werden, um eine schrittweise Entwicklung und Validierung von Metriken durchzuführen. Ein Vorgehen dazu ist in den folgenden Abschnitten dargestellt (vgl. Abb. 5.2). Dieses Vorgehen kann konstruktiv bei der Entwicklung von Metriken in ITMS eingesetzt werden. Die in den einzelnen Schritten beschriebenen Überlegungen sind außerdem für die Interpretation von Messwerten relevant. Beispiele für die konkrete Anwendung dieses Verfahrens finden sich in einem entsprechenden Konferenzbeitrag [SL08].

5.2.1. Identifikation von Qualitätsmerkmalen

Zunächst muss eingegrenzt werden, welche Qualitätsmerkmale des Prozesses von Interesse sind. Unter einem Qualitätsmerkmal ist eine subjektive Qualitätserwartung zu verstehen (vgl. Abschnitt 6.3). Dabei muss von den Qualitätszielen der Organisation (beispielsweise ein Unternehmen, ein Geschäftsbereich, oder ein Entwicklungsteam) ausgegangen werden [ED07].

Zur Herleitung von Qualitätszielen ist es hilfreich, die Organisation aus einer Außensicht zu betrachten. Die Frage ist, welche aktuellen Probleme und welche zukünftigen Herausforderungen für die Organisation bestehen. Nützlich bei dieser Analyse können auch die im Abschnitt 2.3.2 genannten Referenzmodelle

sein. Beispiele für die Auswahl von Zielen und für die Identifikation verwandter Qualitätsmerkmale finden sich in Abschnitt 8.1.

5.2.2. Identifikation von Qualitätsattributen

Ausgehend von den Qualitätsmerkmalen müssen messbare Attribute gefunden werden, mit deren Hilfe eine Aussage über das Qualitätsmerkmal getroffen werden kann. Dazu muss analysiert werden, wie das Issue-Tracking-System verwendet wird. Es reicht dabei nicht aus, den definierten Workflow der Änderungsanträge zu betrachten. In der Praxis können die Zustände und Attribute unterschiedlich interpretiert und verwendet werden. Außerdem ist zu klären, ob zu Attributen die Werte ausreichend konsistent und vollständig eingetragen werden, um sie zu Bewertungszwecken heranzuziehen.

Zur Identifikation von Attributen kann dann entsprechend dem *Goal Question Metric* Ansatz vorgegangen werden [BCR94]. Möglicherweise ist ein Qualitätsmerkmal zu vielschichtig, um geeignete Fragestellungen bzw. Metriken abzuleiten. In diesem Falle muss dieses Qualitätsmerkmal zunächst in Submerkmale verfeinert werden [KPF95].

Um geeignete Metriken aufzufinden, muss betrachtet werden, wie sich ein Qualitätsmerkmal im Lebenslauf eines einzelnen Änderungsantrags wiederspiegelt. Auf dieser Basis können Attribute identifiziert werden, die sich auf das Vorkommen von bestimmten Ereignissen, die Länge von Zeitintervallen oder Verweildauern im Lebenslauf eines Änderungsantrags beziehen. Zu solchen Attributen kann eine Metrik-Spezifikation in ITMS formuliert werden, die ein entsprechendes Bewertungsverfahren anwendet.

Diese Metrik-Spezifikation kann auf historischen Daten ausgewertet werden. Dann müssen stichprobenartig Änderungsanträge ausgesucht und inspiziert werden. Dabei wird überprüft, ob sich die bei der Festlegung der Metrik gemachten Annahmen korrekt im Lebenslauf dieser Änderungsanträge widerspiegeln. Somit lassen sich frühzeitig mögliche falsche Annahmen oder Irrtümer bei der Spezifikation der Metrik aufdecken und korrigieren. Die Gruppenwertberechnung wurde zum jetzigen Zeitpunkt noch nicht festgelegt, da zunächst nur die Ergebnisse für einzelne Änderungsanträge betrachtet werden.

Schließlich sei noch angemerkt, dass es unter Umständen nicht für jedes im vorherigen Schritt identifizierte Qualitätsmerkmal möglich ist, eine Bewertung auf Basis der Daten aus Software-Entwicklungsarchiven durchzuführen.

5.2.3. Definition von Qualitätsindikatoren

In der im vorherigen Schritt formulierten ITMS Metrik-Spezifikation ist nur beschrieben, wie Bewertungszahlen für individuelle Änderungsanträge ermittelt

werden. Im nächsten Schritt müssen diese Zahlen mit Hilfe von Gruppenwertberechnungen zusammengefasst werden, um einen Qualitätsindikator zu erhalten, der zwischen Projekten verglichen werden kann. Dazu muss eine Metrik-Spezifikation folgende Anforderungen erfüllen:

- **Ausschluss von ungewünschten Einflussfaktoren**: Es ist zu vermeiden, dass die Messwerte überwiegend durch Faktoren wie Größe oder Alter des Projekts beeinflusst sind. Es muss also eine geeignete Normalisierung gefunden werden. Dazu können bekannte Verfahren zur Normalisierung verwendet werden, wie die Normalisierung mit einem Größenmaß oder die Bildung von prozentualen Verhältnissen. Dabei müssen Auswirkungen auf Eigenschaften der Metrik, wie etwa ihren Skalentyp berücksichtigt werden [Zus97]. Weiterhin kann auch mit dem Bewertungsverfahren schon eine Normalisierung durchgeführt werden. Es ist hier oft sinnvoll, bei der Betrachtung von Intervalllängen oder Verweildauern nicht die absoluten Zeiten, sondern nur die Verletzungen einer bestimmten oberen Schranke zu betrachten.

- **Zeitlicher Bezug zu beobachteten Problemen im Prozess**: Jede für einen Änderungsantrag ermittelte Bewertungszahl ist einer bestimmten Zeitperiode zugeordnet. Um irreführende Interpretationen zu vermeiden, muss sichergestellt werden, dass der Zeitpunkt, zu dem die Bewertungszahl zugeordnet ist, in zeitlichem Bezug zu einer möglichen Ursache im Prozess steht.

  Ein solches Problem kann beispielsweise auftreten, wenn als Bewertungszahl die Lösungszeit für einen Änderungsantrag betrachtet, und die Bewertungszahl zu dem Zeitpunkt der Lösung eines Änderungsantrags vergeben wird. Wenn dann etwa im Rahmen einer Bereinigung der Issue-Tracking-Datenbank eine Anzahl sehr alter Änderungsanträge geschlossen wird, könnte dies als lange Bearbeitungsdauer in der betroffenen Zeitperiode interpretiert werden. Die Gründe für die Nichtbearbeitung dieser Änderungsanträge liegen allerdings in früheren Zeitperioden. In einem solchen Fall ist wiederum die Verwendung von oberen Schranken bei der Berechnung von Intervalllängen oder Verweilzeiten sinnvoll. Die Verletzung der oberen Schranke wird dann als negative Auswirkung auf ein Qualitätsmerkmal des Prozesses interpretiert.

- **Geeignete Granularität der Zeitperioden**: Die Wahl der betrachteten Zeitperioden (z.B. Woche, Monat, Quartal) kann unter Umständen zu verzerrten Ergebnissen führen. Sind die Zeitperioden zu klein gewählt, ist die Anzahl der in einer einzelnen Zeitperiode betrachteten Änderungsanträge oder Ereignisse zu gering, um sinnvolle Mittelwerte zu bilden. Weiterhin kann die Phase, in der sich ein Projekt befindet, Auswirkungen auf die Messergebnisse haben. Daher ist es oft sinnvoll, die betrachteten Zeitperioden ähnlich zu den Release-Zyklen zu wählen, um Vergleiche über mehrere Releases ziehen zu können.

Die Metrik-Spezifikationen können wiederum unmittelbar auf historischen Daten von Projekten ausgewertet werden, und mit Erfahrungen oder einer Experteneinschätzung über diese Projekte verglichen werden. Zudem ist es hier hilfreich, solche Messwerte genauer zu untersuchen, die Ausreißer sind oder bei denen starke Schwankungen über die Zeit vorliegen. Oft lassen sich dadurch Probleme in der Metrik-Spezifikation aufdecken.

Nicht alle der genannten Probleme lassen sich bei der Spezifikation einer Metrik vollständig ausschließen. Insbesondere gilt dies bei unregelmäßig auftretenden Bereinigungen der Issue-Tracking-Datenbank, oder Veränderungen bei der Bedeutung von Attributwerten. In diesen Fällen muss sichergestellt werden, dass solche Gegebenheiten bei der Interpretation der Messwerte berücksichtigt werden.

5.2.4. Interpretation basierend auf empirischen Daten

Zu einer weiteren experimentellen Validierung der entwickelten Metrik kann ein empirischer Vergleich innerhalb einer Gruppe von Messobjekten herangezogen werden. Dabei wird dann nicht mehr der absolute Messwert für ein Messobjekt betrachtet, sondern dessen Einordnung in der Verteilung von Messwerten einer Gruppe vergleichbarer Messobjekte. Dazu können beispielsweise die Quartile der Verteilung herangezogen werden, um die Messwerte zu klassifizieren (vgl. Abschnitt 6.3). Dadurch wird eine intuitive Interpretation der Messergebnisse ermöglicht. Auf Basis von historischen Daten der Issue-Tracking-Datenbank lassen sich dann die Einordnung der Messobjekte und Veränderungen über mehrere Zeitperioden betrachten. Wiederum kann dann überprüft werden, ob diese Ergebnisse mit Erfahrungen oder einer Expertenbewertung übereinstimmen. Sind über den Verlauf der Zeit sprunghafte Veränderungen festzustellen, muss insbesondere geklärt werden, ob dies mit der Expertenbewertung im Einklang steht.

Insgesamt ergibt sich aus den vorgenannten Schritten also ein konstruktives Vorgehen zur zielgerichteten Entwicklung von Metrik-Spezifikationen in ITMS. Das Verfahren zielt darauf ab, möglichst frühzeitig zu erkennen, ob die bei der Entwicklung der Metrik zu Grunde gelegten Annahmen stimmig sind. Dazu finden Validierungsschritte auf mehren Ebenen statt (vgl. Abb. 5.2), und die Metrik-Spezifikation kann iterativ angepasst werden. Das vorgestellte Verfahren kann im Kontext der Auswertung von Software-Entwicklungsarchiven als Ergänzung zum GQM-Ansatz verwendet werden, da es die Schritte zur Definition von Metriken konkretisiert.

6. Ein Metamodell für Qualitätsmodelle

> The quality, not the longevity, of one's life is what is important.
>
> *(Martin Luther King Jr., 1929-1968)*

Inhaltsangabe

Die Sprache ITMS und das im vorherigen Abschnitt vorgestellte Verfahren erleichtern die Definition und Validierung von Metriken. Wie schon in Abschnitt 2.1.1 erläutert, reichen die reinen Messergebnisse von Metriken für eine Qualitätsbewertung noch nicht aus. Die Messergebnisse müssen aggregiert werden, um verdichtete Informationen zu Qualitätsmerkmalen auf einer höheren Ebene zu erhalten. Die Form der Aggregation wird in einem Qualitätsmodell festgelegt (vgl. Abschnitt 2.2). Das Qualitätsmodell ist oft nicht explizit, sondern nur implizit in Werkzeugen zur Metrikberechnung oder Report-Erstellung gegeben.

Da die relevanten Metriken von den Informationsbedürfnissen der Organisation abhängen, müssen Qualitätsmodelle typischerweise organisationsspezifisch angepasst werden. Ist das Qualitätsmodell allerdings implizit in einem Werkzeug eingebettet, sind die Anpassungsmöglichkeiten des Modells eingeschränkt. Des Weiteren ist es dadurch für den Nutzer des Werkzeugs oft schwierig nachzuvollziehen, wie die Messergebnisse aggregiert werden.

Aus den vorgenannten Gründen entstand die Idee, die Bearbeitung benutzerdefinierter Qualitätsmodelle und deren Auswertung auf Basis empirischer Vergleichsdaten durch ein eigenes Softwarewerkzeug zu unterstützen. Grundlage eines solchen Werkzeugs muss ein Metamodell sein, welches die zulässigen Modellelemente und deren Verknüpfungen beschreibt [KLPN97]. In dem folgenden Abschnitt wird zunächst die Strukturierung von gegebenen Modellen zur Qualitätsbewertung analysiert. Darauf werden Anforderungen an ein Metamodell für Qualitätsmodelle formuliert. In Abschnitt 6.3 wird schließlich das entworfene Metamodell vorgestellt.

6.1. Wiederkehrende Konzepte beim Aufbau von Qualitätsmodellen

In Modellen zur Qualitätsbewertung findet sich eine Reihe wiederkehrender Konzepte, unabhängig von der zu bewertenden Entität. In diesem Abschnitt werden diese Konzepte herausgearbeitet, da sie eine wichtige Grundlage für das in Abschnitt 6.3 vorgestellte Metamodell bilden. Jedes Qualitätsmodell benutzt für gewöhnlich eigene Begrifflichkeiten. Im Folgenden wird nach Möglichkeit die im Standard ISO/IEC 15939 gegebene Terminologie genutzt.

Die betrachteten Messobjekte sind in der Regel hierarchisch strukturiert. Zudem finden sich Inklusionsbeziehungen zwischen den Messobjekten, beispielsweise Methode, Klasse und Paket bei der Betrachtung von Java-Quellcode.

Eine weitere Dimension bei der Betrachtung der Messobjekte ist die Zeit. Liegen historische Daten über die Messobjekte vor, können auch Messungen zu einem früheren festgelegten Zeitpunkt, oder Messungen während eines bestimmten Zeitintervalls herangezogen werden.

Basismetriken werden verwendet, um ein einzelnes Attribut eines Messobjekts zu bestimmen. Die Beschreibung der Metrik muss angeben, nach welcher Messmethode das Attribut bestimmt wird, und auf welche Skala abgebildet wird. Bei der Vermessung wird die Entität bezüglich des betrachteten Merkmals abstrahiert, es findet also eine Modellbildung statt [LL07] (z.B. *Lines of Code* [SSM06]).

Basismetriken werden dann weiter zu abgeleiteten Metriken kombiniert. **Abgeleitete Metriken** vereinen Informationen über mehrere Attribute eines Messobjektes, oder über das selbe Attribut mehrerer Messobjekte. Typische Operationen dazu sind:

- **Aggregation**, um ein Messobjekt auf einer höheren Ebene zu charakterisieren. Beispiele sind die *Weighted Methods per Class* Metrik [CK94] oder die mittlere Lösungszeit für die Supportanfragen zu einem Software-Produkt.
- **Normalisierung** mit einem Größenmaß (z.B. Defekte pro Quellcodezeile).
- Zählung des Auftretens von bestimmten Problemmustern. Angelehnt an Simon et al. [SSM06] ist ein **Problemmuster** eine Ausprägung eines oder mehrerer Attribute, die von Ausprägungen anderer vergleichbarer Messobjekte abweicht, und begründete negative Auswirkungen auf ein Qualitätsmerkmal besitzt. Problemmuster werden üblicherweise durch die Überschreitung von festgelegten Grenzwerten definiert, beispielsweise für die Länge einer Klasse oder die Lösungszeit für einen Problembericht.

Vereint eine abgeleitete Metrik genügend Informationen über das Messobjekt, um vom Anwender zur Bewertung eines bestimmten Qualitätsmerkmals herangezogen zu werden, wird diese Metrik als **Indikator** bezeichnet. Ein Indikator

übermittelt also Informationen über bestimmte Attribute des Messobjekts, und dient als Basis für Analyse und Entscheidungsfindung [ISO07b].

Ein weiterer Schritt ist die Aggregation zu einer Indexzahl. Simon et al. geben dazu die folgende Definiton:

Definition 6.1 Eine **Indexzahl** ist eine kollektive Kenngröße für eine Vielzahl unter einem bestimmten Gesichtspunkt zusammenfassbarer Einzelphänomene, die durch eine geeignete Kombination individueller Einzelmerkmale entsteht [SSM06].

Die Ermittlung einer Indexzahl kann zum einen auf einer mathematischen Funktion basieren. Üblich sind lineare Gleichungen (z.B. im Modell von McCall [CM78]). Seltener finden sich kompliziertere Berechnungen (z.B. *Maintainability Index* [CLO95]). Zum anderen können zur Bestimmung einer Indexzahl **Qualitätsstufen** definiert werden. Beispiele dazu sind das *Capability Maturity Model* (CMMI) [CMM06] und der *Code Quality Index* zur Bewertung der technischen Qualität eines Softwareprodukts [SSM06]. Folgende Ansätze werden typischerweise bei der Festlegung der Qualitätsstufen angewandt:

- **Indikatorzuwachs**: In höheren Qualitätsstufen werden weitere Indikatoren berücksichtigt. Dadurch kann ein typischer Weg der Verbesserung aufgezeigt werden. Weiterhin können Aspekte wie der unmittelbare Nutzen von Verbesserungen, als auch die mit einer Verbesserungsmaßnahme verbundenen Kosten berücksichtigt werden.
- **Schwellwerttunnel**: In höheren Qualitätsstufen müssen die Indikatoren engere Grenzwerte erfüllen.

Somit können zwei unterschiedliche Verwendungen von **Grenzwerten** unterschieden werden:

- Grenzwerte zur Identifikation von Problemmustern
- Grenzwerte zur Festlegung von Qualitätsstufen

Die Herleitung von Grenzwerten kann zum einen aus gegebenen Konventionen oder Richtlinien motiviert sein, zum anderen durch den Vergleich mit empirischen Daten ermittelt werden. Konventionen oder Richtlinien können beispielsweise durch Kodierungsrichtlinien gegeben sein. So wird in Sun's *Java Code Conventions* gefordert, dass Klassen länger als 2000 Zeilen zu vermeiden sind [Sun99]. Im Bereich des Problem Managements können Grenzwerte beispielsweise durch die in *Service Level Agreements* festgelegten Antwortzeiten gegeben sein.

Die Definition von Grenzwerten auf Basis empirischer Daten kann sich einerseits auf die Betrachtung vergleichbarer Messobjekte beziehen. Beispielsweise wird beim *Code Quality Index* eine Datenbank von Messergebnissen zu Codequalität in großen industriellen Softwareprodukten herangezogen [SSM06]. Dazu ist es allerdings nötig, über repräsentative Daten zu verfügen. Eine zweite Möglichkeit

ist es, nur solche Messobjekte für den Vergleich heranzuziehen, die als qualitativ akzeptabel angesehen werden. Diesen Ansatz findet sich in einem von Washizaki et al. vorgestellten Framework zur Bewertung von Code-Qualität [WNF+07].

Bei der Klassifikation der Messergebnisse auf Basis der Werteverteilung empirischer Daten, können die Quantile der Verteilung als Grenzwerte herangezogen werden [SSM06]. Es sind aber auch andere Funktionen denkbar. So verwendet Washizaki eine Bewertungsfunktion, bei der Messwerte größer als das dritte Quartil auf 100, Messwerte kleiner als das erste Quartil auf 0, und dazwischen liegende Werte in den Wertebereich zwischen 0 und 100 abgebildet werden [WNF+07]. Somit wird für Werte im mittleren Bereich eine differenziertere Klassifikation ermöglicht.

Neben der hierarchischen Struktur können auch weitere Abhängigkeiten zwischen Qualitätsmerkmalen betrachtet werden. So schlagen Khaddaj und Horgan vor, folgende Abhängigkeiten zu beschreiben [KH05]:

- **Neutral**: Eine Verbesserung des Qualitätsmerkmals A hat wahrscheinlich keinen Einfluss auf das Qualitätsmerkmal B.
- **Direkt**: Eine Verbesserung des Qualitätsmerkmals A führt wahrscheinlich zu einer Verbesserung des Qualitätsmerkmals B.
- **Invers**: Eine Verbesserung von A führt wahrscheinlich zu einer Verschlechterung des Qualitätsmerkmals B.

Wagner und Deissenboeck schlagen ein sogenanntes zweidimensionales Qualitätsmodell vor [WD07, BDP06]. Eine Dimension sind die unterschiedlichen Aktivitäten bei der Entwicklung, aus ökonomischer Sicht also die Kostenfaktoren. Die zweite Dimension sind Eigenschaften des Messobjektes, die automatisch oder manuell bewertet werden können. Damit soll erfasst werden, welche Aktivitäten von welchen Qualitätsmerkmalen beeinflusst werden. Plösch et al. schlagen eine zusätzliche Klassifizierung von internen Qualitätsmerkmalen nach den technischen Themen Laufzeit, Code und Design vor [PGK+09].

Die drei vorgenannten Ansätze verdeutlichen bestimmte Aspekte im Sinne eines Modells zur Definition von Qualität (vgl. Abschnitt 2.2). Allerdings wird nicht präzisiert, wie und ob solche Abhängigkeiten in einen Modell zur automatisierten Qualitätsbewertung auf Basis von Metriken genutzt werden können.

6.2. Anforderungen

Dem im nächsten Abschnitt beschriebenen Metamodell liegen folgende übergeordnete Anforderungen zu Grunde:

- Das Metamodell soll flexibel genug sein, um Qualitätsmodelle für unterschiedliche Kontexte zu erstellen. Es soll also nicht nur für Qualitätsmodelle im Kontext der Prozessbewertung auf Basis von Software-Entwicklungsarchiven einsetzbar sein, sondern auch für andere Anwendungen, beispielsweise für die Bewertung der technischen Qualität von Quellcode.

- Der Fokus liegt auf Modellen zur metrik-basierten Qualitätsbewertung. Die erstellten Qualitätsmodelle sollen also operativ einsetzbar sein. Auf Basis des Modells soll eine Auswertung mit Hilfe von Metrikwerkzeugen durchgeführt werden können.

- Das Metamodell muss es ermöglichen, typische Konzepte von bekannten Qualitätsmodellen zu beschreiben, wie verschiedene Strukturierungsmöglichkeiten bei der Hierarchisierung von Qualitätsmerkmalen, die Ermittlung von Indexzahlen und die Definition von Qualitätsstufen.

- Eine möglicherweise gegenläufige Anforderung dazu ist, dass die erstellten Qualitätsmodelle konzeptuell einfach sein sollen. Sowohl für Modellierer als auch für Anwender eines Qualitätsmodells müssen die in dem Modell beschriebene Strukturierung von Qualitätsmerkmalen und die Aggregation zu Indexzahlen verständlich und nachvollziehbar sein. Dazu soll sich das Metamodell an etablierten Begrifflichkeiten orientieren. Es soll nur eine überschaubare Anzahl von Modellelementen zur Verfügung stehen, ohne die Ausdruckskraft zu sehr einzuschränken.

6.3. Metamodell

Wie in Abschnitt 6.1 erkennbar, basieren Modelle zur metrik-basierten Qualitätsbewertung auf einer Unterscheidung zwischen subjektiven Qualitätserwartungen oder -zielen und objektiv messbaren Attributen. Im Sinne des von Simon et al. eingeführten bidirektionalen Qualitätsmodells bilden Qualitätsindikatoren das Bindeglied zwischen den subjektiven Qualitätserwartungen und den objektiven Messungen [SSM06]. Die Messung der Attribute selbst wird durch die Beschreibung einer Metrik angegeben. Die Form der Beschreibung hängt von dem zu Grunde liegenden Werkzeug zur Metrikauswertung ab. Somit ergeben sich folgende Hauptelemente des Qualitätsmodells:

- Ein **Qualitätsmerkmal** (englisch: *Quality Characteristic*) beschreibt eine subjektive Qualitätserwartung.

- Ein **Qualitätsindikator** beschreibt, wie quantitativ messbare Attribute einer Entität zur Bewertung eines Qualitätsmerkmals herangezogen werden können. Dazu werden eine oder mehrere Metriken definiert, und deren Verbindung zu einem Auswertungswerkzeug festgelegt.

Wie bereits erwähnt, verwenden unterschiedliche Qualitätsmodelle eine sehr unterschiedliche Terminologie. Das hier vorgestellte Metamodell orientiert sich an

der Terminologie des Standards ISO/IEC 9126, um so zu einer einheitlichen Begriffsverwendung beizutragen.

Qualitätsmerkmale lassen sich in untergeordnete Qualitätsmerkmale verfeinern. Beispielsweise lässt sich das Qualitätsmerkmal Planungsgenauigkeit aufgliedern in Aufwandseinhaltung, Termineinhaltung und Prozesstransparenz. Um im Metamodell möglichst wenig Einschränkungen für die Struktur der Qualitätsmerkmale zu geben, können diese als gerichteter azyklischer Graph hierarchisiert werden. Damit sind auch nicht strikte Hierarchien und mehrere Wurzelknoten möglich. Quellen in diesem gerichteten Graphen sind die Qualitätsindikatoren.

An allen Knoten im Graphen können die folgenden Informationen annotiert werden:

- Erläuterung: Als freier Text können eine Begründung als auch Hinweise für die Interpretation angegeben werden.
- Beschreibung des möglichen Wertebereichs: Dazu wird der kleinste und größte mögliche Wert angegeben, der für das Qualitätsmerkmal oder den Qualitätsindikator zurückgeliefert wird.
- Optional kann außerdem angegeben werden, ob kleinere oder größere Werte als besser zu interpretieren sind.

Abbildung 6.1 zeigt das Metamodell in Form eines UML-Klassendiagramms. Qualitätsmerkmale und -indikatoren sind die Knoten in einem gerichteten azyklischen Graphen. Die Gemeinsamkeiten von Qualitätsmerkmalen und -indikatoren sind im Begriff Qualitätsknoten abstrahiert. Die Kanten des Graphen sind nicht als eigener Begriff modelliert. Sie ergeben sich durch die Assoziationsbeziehung zwischen Qualitätsmerkmal und Qualitätsknoten. Ein Qualitätsmerkmal ist also durch eingehende Kanten mit anderen Qualitätsknoten verbunden. Qualitätsindikatoren haben somit nur ausgehende Kanten und bilden die Quellen eines gerichteten azyklischen Graphen. Innere Knoten und Senken des Graphen sind somit immer Qualitätsmerkmale.

Für beide Arten von Knoten muss jeweils angegeben werden, wie ein Wert zur Qualitätsbewertung berechnet wird, also wie das Modell operativ angewendet wird. Dies geschieht in der **Berechnungsvorschrift**, die für Qualitätsmerkmale und Qualitätsindikatoren unterschiedlich ausgeprägt ist.

Zu einem Qualitätsindikator gehört eine **Metrikberechnungsvorschrift**, die beschreibt, wie ein Wert unter Verwendung eines Metrikwerkzeugs ermittelt wird. Die Metrikberechnungsvorschrift muss wiederum konkret für ein verwendetes Metrikwerkzeug ausgeprägt werden. Bei Verwendung von ITMS wird dazu die deklarative Spezifikation der Metrik angegeben. Dabei können bereits innerhalb der Metrik-Spezifikation Berechnungen vorgenommen werden, die mehrere Messungen kombinieren, beispielsweise die Normalisierung mit einem Größenmaß, oder das Zählen von Problemmustern.

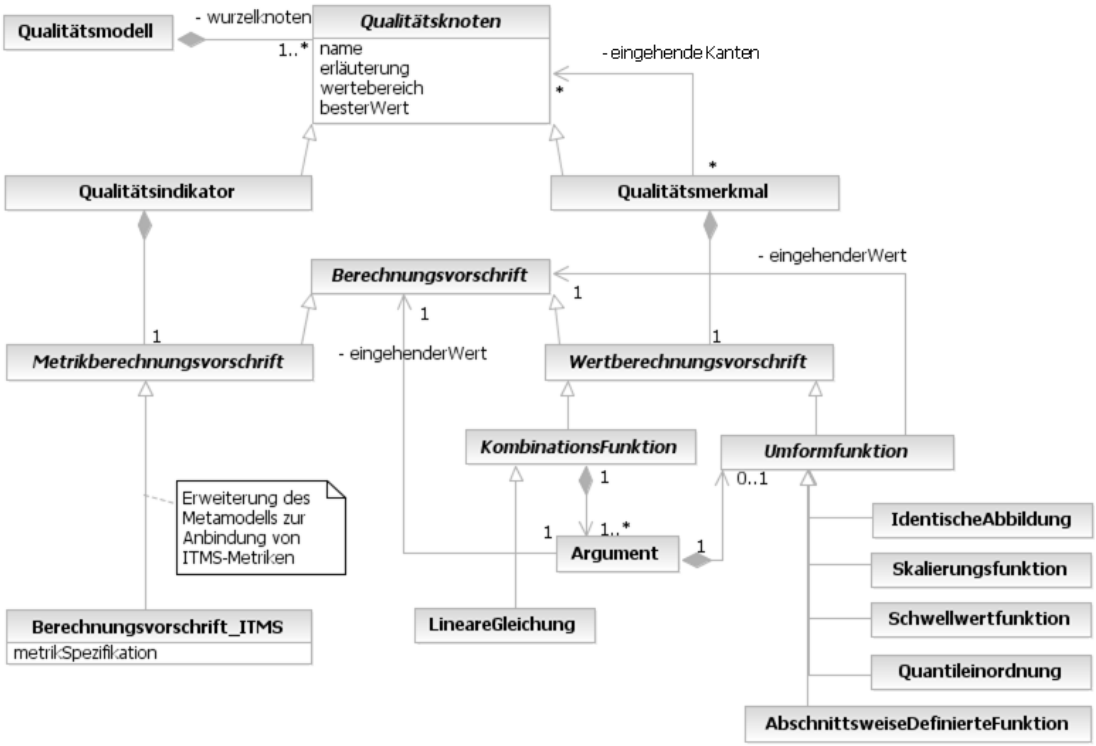


Abbildung 6.1.: Metamodell für Qualitätsmodelle

Zu einem Qualitätsmerkmal gehört eine **Wertberechnungsvorschrift**. Diese gibt an, wie der Wert zu einem Qualitätsmerkmal auf Basis der Werte der mit eingehenden Kanten verbundenen Qualitätsknoten berechnet wird.

Bei den Elementen, die zur Festlegung der Wertberechnungsvorschrift zur Verfügung stehen, musste ein Kompromiss gefunden werden zwischen der Ausdrucksstärke und der Verständlichkeit der entstehenden Beschreibungen. Das Metamodell zielt darauf ab, mit einer kleinen Zahl einfacher Konzepte eine hohe Ausdruckstärke zu erzielen. Zudem soll das Modell für zukünftige Anwendungen erweiterbar sein.

Die Wertberechnungsvorschrift kann mit Hilfe von zwei Arten von Funktionen definiert werden: Einstellige Umformfunktionen und mehrstellige Kombinationsfunktionen. Abbildung 6.3 verdeutlicht die Verwendung dieser Funktionen am Beispiel eines Qualitätsknotens A mit drei eingehenden Kanten. Jeder der verbundenen Knoten A_i liefert einen Wert w_i. Auf jeden dieser Werte wird zunächst eine Umformfunktion u_i angewendet. Die Ergebnisse bilden die Eingangswerte für die Kombinationsfunktion k. Diese liefert schließlich den für Konten A berechneten Wert.

Mögliche **Umformfunktionen** sind:

- Die **identische Abbildung** gibt den eingehenden Wert zurück.

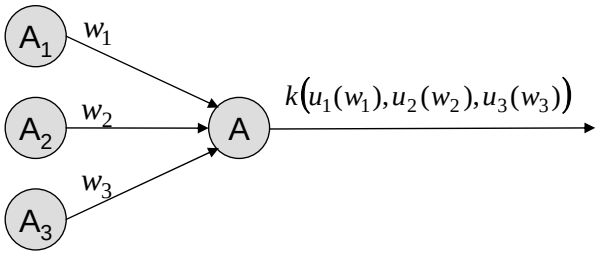


Abbildung 6.2.: Kombinationsfunktion und Umformfunktion

- Die **Skalierung auf einen vorgegebenen Wertebereich** dient dazu, Messergebnisse vergleichbar zu machen, indem sie auf einen einheitlichen Wertebereich übertragen werden, beispielsweise 0-100. Voraussetzung ist, dass für den eingehenden Wert ein endlicher Wertebereich definiert wurde.

- Die **Schwellwertfunktion** gibt für einen eingehende Wert oberhalb einer Grenze s einen Wert a zurück, ansonsten einen Wert b.

- Die **abschnittsweise definierte Funktion** kann vom Benutzer definiert werden, und bildet jeweils Wertintervalle auf eine Konstante ab. Mit Hilfe dieser Funktion können beispielsweise für bestimmte Wertebereiche je eine Indexzahl vergeben werden. Der Wert dieser Indexzahl kann dann etwa interpretiert werden als „unzulässig“, „akzeptabel“ oder „gut“.

- Die **Quantileinordnung** liefert eine Einordnung des Werts auf Basis der Verteilung von Vergleichsdaten. Dazu müssen zu dem eingehenden Knoten entsprechende Vergleichdaten gegeben sein. Zudem muss festgelegt werden, welche Quantile betrachtet werden. Üblich sind Quartile. Eine Einordnung für Quartile kann beispielsweise folgendermaßen definiert werden:

  Definition 6.2 Sei $V = \{v_1, \ldots, v_k\}$ eine Menge von Vergleichsdaten, und $Q_i(V)$ das i-te Quartil von V für $i = 1 \ldots 3$. Dann ist die Quartileinordnung definiert als:

$$f_{quartil,V}(w) = \begin{cases} 1 & Q_3(V) < w \\ 2 & Q_2(V) < w \leq Q_3(V) \\ 3 & Q_1(V) < w \leq Q_2(V) \\ 4 & w \leq Q_1(V) \end{cases}$$

  .

Eine **Kombinationsfunktion** berechnet ein Ergebnis aus den Werten von n mit eingehenden Kanten verbundenen Qualitätsknoten. Auf jedes Argument der Funktion kann optional eine Umformfunktion angewandt werden. Die am häufigsten gebräuchliche Kombinationsfunktion ist die lineare Gleichung:

$$f(x_1, \ldots, x_n) = c_1 x_1 + c_2 x_2 + \ldots + c_n x_n$$

Das Metamodell ist offen, um mit weiteren Funktionen ergänzt zu werden. Auf das Ergebnis einer Kombinationsfunktion kann optional eine Quantileinordnung angewendet werden.

Von den in Abschnitt 6.1 genannten Konzepten wurden die Qualitätsstufen noch nicht betrachtet. Die Erreichung einer Qualitätsstufe lässt sich mit Hilfe eines Teilbaums im Graphen, sowie der Schwellwertfunktion modellieren. Der für das Qualitätsmerkmal an der Wurzel des Teilbaums berechnete Wert gibt dann an, ob die entsprechende Qualitätsstufe erreicht wurde.

6.4. Fazit

An dieser Stelle lässt sich eine erste Bewertung des vorgestellten Metamodells vornehmen.

Das Metamodell ist unabhängig von der Struktur der Messobjekte. Bei der Quellcode-Vermessung sind die Messobjekte beispielsweise in Pakete, Klassen und Methoden strukturiert. Bei der Vermessung von Issue-Tracking-Daten sind die Messobjekte beispielsweise Produkte oder Komponenten.

In der Metrikberechnungsvorschrift eines Qualitätsindikators sind die Messobjekte noch nicht vollständig definiert. Dies erfolgt erst bei der Festlegung einer konkreten Auswertung auf Basis des Qualitätsmodells. Somit findet sich die Struktur der Messobjekte auch im Ergebnisdokument der Metrikauswertung wieder.

Das hier beschriebene Metamodell ist operativ einsetzbar und deckt typische Konzepte aus bekannten Qualitätsmodellen ab. Die Erfüllung der weiteren Anforderungen, insbesondere im Hinblick auf Flexibilität, Ausdrucksstärke und Einfachheit des Modells, kann nur vor dem Hintergrund der praktischen Anwendung diskutiert werden. Dies erfolgt in Abschnitt 8.4.

Wichtig ist hier festzuhalten, dass die Erstellung eines solchen Qualitätsmodells keine wertfreie Tätigkeit ist [Gli05]. Einem Qualitätsmodell liegen bestimmte Annahmen zu Grunde, die bei der Verwendung des Modells überprüft werden müssen. Sein Einsatz verändert die Wahrnehmung des modellierten Problembereichs. Somit wird zunächst eine deskriptive Modellbildung stattfinden [Sta73], mit dem Zweck der Dokumentation von Qualitätsmerkmalen. Durch den Einsatz eines solchen Qualitätsmodells zur Bewertung von Prozessen können dann Erfahrungen gesammelt werden, inwieweit das Modell angemessen ist. Gleichzeitig wird dabei das Verständnis über den zu bewertenden Prozess verbessert. Erst auf Basis ausreichender Erfahrungen können Qualitätsvorgaben im Sinne einer präskriptiven Modellbildung gemacht werden.

7. Werkzeugunterstützung

Ce n'est pas le but qui est intéressant,
ce sont les moyens pour y parvenir.

(Georges Braque, 1882-1963)

Inhaltsangabe

In den vorherigen Kapiteln wurden die konzeptionellen Grundlagen für die deklarative Spezifikation von Metriken sowie für operativ einsetzbare Qualitätsmodelle gelegt. In diesem Kapitel wird eine entsprechende Unterstützung durch eine Reihe von Software-Werkzeugen vorgestellt. Diese Werkzeug-Suite trägt den Namen *QMetric*. Ein Überblick über die an die Werkzeuge gestellten Anforderungen wird in Abschnitt 7.1 gegeben. Die Erläuterung der Architektur erfolgt in Abschnitt 7.2. Abschnitt 7.3 beschreibt Qualitätssicherungsmaßnahmen für die Referenzimplementierung zur Berechnung von ITMS Metrik-Spezifikationen.

7.1. Anforderungen

Die hier beschriebene QMetric-Werkzeug-Suite besteht aus einer Referenzimplementierung zur Berechnung von ITMS-Metriken, einer Web-Anwendung zur Abfrage von Metriken, sowie Werkzeugen zur Definition und Auswertung von Qualitätsmodellen.

Aus den in Kapitel 3 formulierten Zielen ergeben sich eine Reihe von zentralen Anforderungen an die Werkzeugunterstützung. Die Werkzeuge sind eingebettet in einen organisatorischen Kontext. Sie sollen helfen zu bewerten, ob organisationsspezifische Ziele erreicht wurden. Dazu sollen die in der Organisation vorhandenen Entwicklungsarchive ausgewertet werden.

Des Weiteren sollen die Werkzeuge für unterschiedlich erfahrene Benutzer benutzbar sein. Deswegen soll die technische Einstiegshürde zur Installation und Anpassung der Werkzeuge, als auch die individuelle Einstiegshürde für den Anwender möglichst gering sein. Der Einstieg für Anwender soll durch Bereitstellung einer grafischen Benutzungsschnittstelle zur Abfrage und Definition von Metriken erleichtert werden.

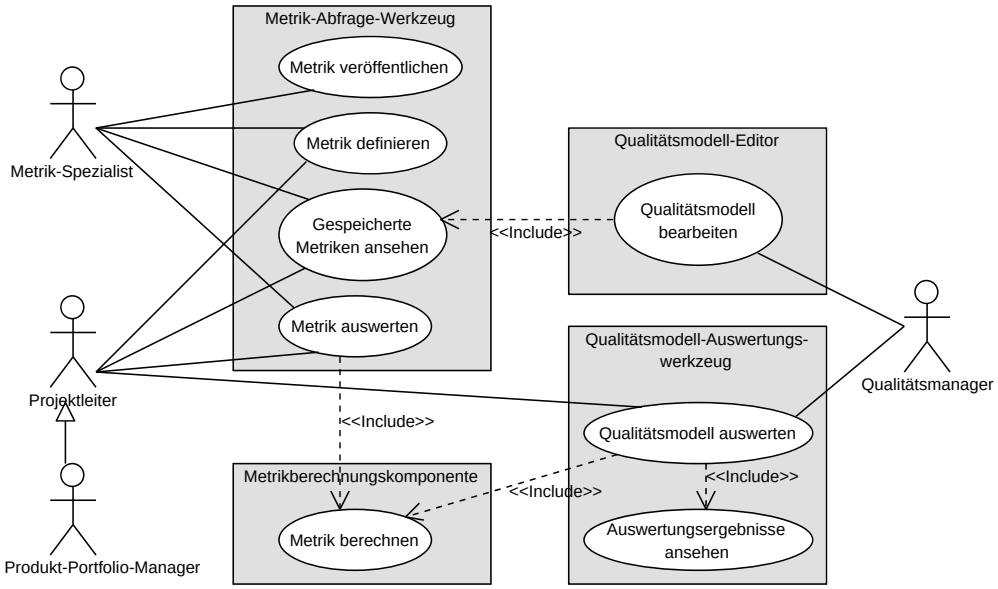


Abbildung 7.1.: Anwendungsfalldiagramm der QMetric-Werkzeug-Suite

Das Anwendungsfalldiagramm in Abbildung 7.1 zeigt die verschiedenen Benutzerrollen, die wesentlichen Anwendungsfälle und die Systemgrenzen der Werkzeuge. Im Folgenden sind die verschiedenen Benutzerrollen erläutert:

- Ein **Metrik-Spezialist** verantwortet die Entwicklung und Validierung von Metriken. Dazu legt er Metrik-Spezifikationen mit einem eindeutigen Namen und einer Beschreibung an. Er kann diese auswerten und überarbeiten. Validierte Metriken können veröffentlicht werden, um sie für andere Benutzer bereitzustellen.

- Ein **Qualitätsmanager** ist verantwortlich für die Definition und Anpassung von Qualitätsmodellen. Dazu muss er bestimmen, welche Qualitätsmerkmale von Interesse sind, wie diese Qualitätsmerkmale weiter untergliedert sind, und welche Metriken als Qualitätsindikator herangezogen werden.

- Ein **Projektleiter** kann auf verschiedenen Wegen Informationen zu seinem Projekt abfragen. Zum einen kann er vordefinierte Metrik-Spezifikationen für ein Projekt auswerten und eigene Metrik-Spezifikationen erstellen. Weiterhin kann er die Prozessqualität auf Basis eines vorgegebenen Qualitätsmodells bewerten lassen.

- Ein **Produkt-Portfolio-Manager** verantwortet mehrere Produkte, die inhaltlich, technisch oder organisatorisch verwandt sind. Ebenso wie bei der Projektleiter-Rolle können vom Produkt-Portfolio-Manager eigene oder vordefinierte Metrik-Spezifikationen angewendet werden als auch die Bewertung auf Basis eines Qualitätsmodells durchgeführt werden. In beiden

Rollen muss es jeweils möglich sein, Details der Ergebnisse abzufragen. Dabei kann entlang der Hierarchie der Qualitätsmerkmale und -indikatoren, in der zeitliche Auflösung und in Bezug auf die Messobjekte verfeinert werden.

Je nach Rolle werden die Werkzeuge mit unterschiedlicher Intensität genutzt. Dies reicht von gelegentlichen Ad-hoc-Abfragen von Metriken, bis hin zur Erstellung und Anpassung von Qualitätsmodellen. Daher besteht die Suite aus mehreren Werkzeugen mit jeweils einem eingegrenzten Einsatzzweck (vgl. Abb. 7.1):

- Das **Metrik-Abfrage-Werkzeug** ist eine Web-Anwendung, die das Auswerten von vordefinierten Metriken, die Definition von eigenen Metriken sowie das Speichern und Veröffentlichen von Metriken unterstützt. Metrik-Spezifikationen können über einen grafischen Dialog erzeugt werden, sodass keine Kenntnis der Sprache ITMS notwendig ist. Die Umsetzung als Web-Anwendung soll organisationsweit einen einfachen Zugang ermöglichen. Optional kann eine Zugriffskontrolle auf Basis der im Issue-Tracking-System angemeldeten Benutzer erfolgen.

- Der **Qualitätsmodell-Editor** ermöglicht das Anlegen und Bearbeiten von Qualitätsmodellen. Dabei wird auf die mit dem Metrik-Abfrage-Werkzeug angelegten Metriken zurückgegriffen.

- Das **Qualitätsmodell-Auswertungswerkzeug** führt eine metrik-basierte Qualitätsbewertung anhand eines gegebenen Qualitätsmodells durch. Dazu müssen vom Anwender eine Reihe von Parametern für die Auswertung angegeben werden, beispielsweise die betrachteten Produkte und der Zeitraum.

- Die **Metrikberechnungskomponente** ist schließlich die Referenzimplementierung für Berechnungen von ITMS Metrik-Spezifikationen.

Im Folgenden werden Feature-Modelle [KCH+90] genutzt, um die an die einzelnen Komponenten der Werkzeug-Suite gestellten Anforderungen zu modellieren. Neben den bereits genannten Benutzerrollen möchten wir auch Features aus Sicht des Administrators betrachten, da die Anpassbarkeit der Werkzeug-Suite für unterschiedliche Datenquellen in unserem Kontext eine wichtige Anforderung ist.

Ein Feature-Modell ist mit Domänenbeziehungen hierarchisch strukturiert. Es wird zwischen obligatorischen Domänenbeziehungen sowie optionalen, alternativen und *Oder*-Domänenbeziehungen unterschieden. Die drei letztgenannten sind Variabilitätsbeziehungen, das heißt, aus ihnen ergeben sich Entscheidungsmöglichkeiten bei der Konfiguration von Produkten. Feature-Modelle bieten somit eine Übersicht über die Funktionen einer Software aus der Sicht des Anwenders, und ermöglichen die Analyse von Variabilitäten bei den Anforderungen [vdM07]. In den folgenden Abbildungen wird die erweiterte FORM-Notation verwendet [KKL+98, FFB02, Sch04]. Eine Legende findet sich in Tabelle 7.1.

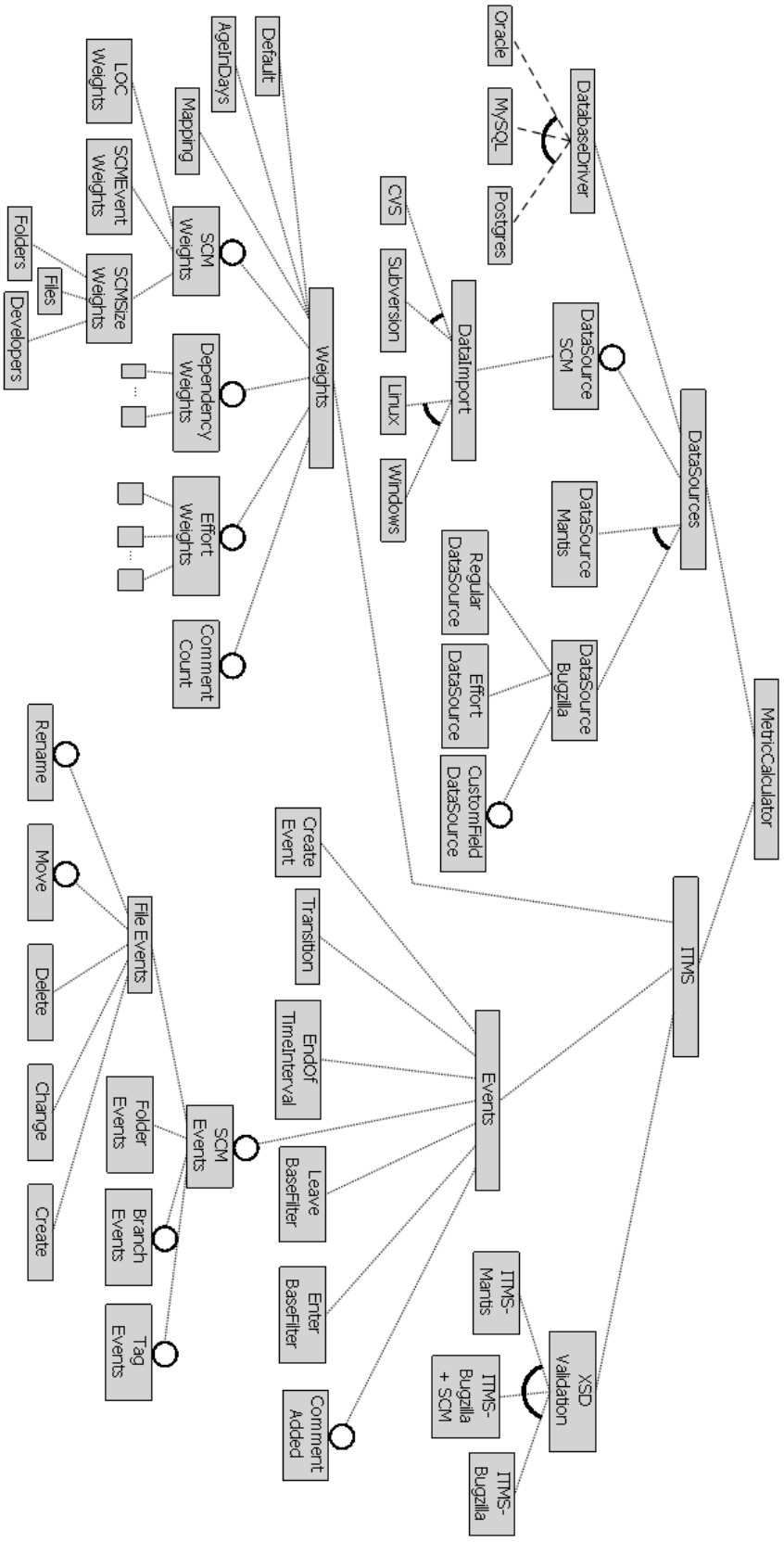


Abbildung 7.2.: Feature-Diagramm für die Metrikberechnungskomponente (Legende siehe Tabelle 7.1)

Tabelle 7.1.: Verwendete Modellierungselemente im Feature-Diagramm

Element	Darstellung
Feature	
Obligatorische Domänenbeziehung	
Optionale Domänenbeziehung	
Alternative Domänenbeziehung	
Oder-Domänenbeziehung	

7.1.1. Metrikberechnungskomponente

Abbildung 7.2 zeigt ein Feature-Modell für die Metrikberechnungskomponente. Aus Gründen der Übersichtlichkeit sind einige Sub-Features nicht mehr dargestellt. Im Folgenden werden die wesentlichen Features erläutert.

Ausgehend von der Wurzel des Feature-Baums liegen auf der höchsten Ebene die Features *DataSources* und *ITMS*. Das Feature *DataSources* beschreibt die Anbindung von Software-Entwicklungsarchiven. Es muss als Datenquelle genau ein Issue-Tracking-System angebunden sein. Zusätzlich kann optional ein Konfigurationsmanagement-System angebunden werden. Durch die unterschiedlichen Datenquellen ergibt sich die Anforderung, auf unterschiedliche Datenbankmanagementsysteme zugreifen zu können. Dies ist im Feature *Database-Driver* modelliert.

Das Feature *DataImport* hat folgenden Hintergrund. Bei der Anbindung eines Konfigurationsmanagement-Systems können die erforderlichen historischen Daten nicht direkt ausgelesen werden, sondern müssen regelmäßig in einem geeigneten Format in eine eigene Datenbank importiert werden. Dies muss für unterschiedliche Konfigurationsmanagement-Systeme und betriebssystemunabhängig möglich sein.

Am Beispiel des Features *BugzillaDataSource* sind einige auswertbare Attribute der Datenquelle ebenfalls als Features dargestellt. Optional können auch benutzerdefinierte Attribute aus Bugzilla ausgewertet werden.

Auf der rechten Seite von Abbildung 7.2 ist das Feature ITMS gezeigt, welches den Sprachumfang von ITMS beschreibt. Je nach Art der erfassten Daten können unterschiedliche Ereignisse und Gewichtungen in den Metrik-Spezifikationen verwendet werden. Wie im Feature-Diagramm ersichtlich ist, sind eine Reihe von Ereignissen und Gewichtungen allgemein für alle Issue-Tracking-Systeme auswertbar. Andere Ereignisse und Gewichtungen stehen beispielsweise nur bei

Erhebung von Aufwänden im Issue-Tracking-System, oder bei der Anbindung von Konfigurationsmanagement-Systemen zur Verfügung.

Die Metrikberechnungskomponente führt weiterhin eine Validierung von Metrik-Spezifikationen gegen ein XML-Schema durch. Diese Schema-Datei muss je nach Sprachumfang ebenfalls unterschiedlich ausgeprägt werden.

Der Übersichtlichkeit halber wurde im Feature-Diagramm auf die Darstellung von Abhängigkeiten wie *exclude* oder *imply* verzichtet. Solche Abhängigkeiten ergeben sich aus der Auswahl der Datenquellen. So sind bei Auswahl des Features *DataSourceSCM* ebenfalls die Features *SCM Events* und *SCM Weights* auszuwählen.

Weitere Details zu Anforderungen an die Berechnungskomponente finden sich in der Arbeit von Grammel [Gra07] sowie in Bezug auf die Anbindung von Konfigurationsmanagement-Systemen in der Arbeit von Lischkowitz [Lis08].

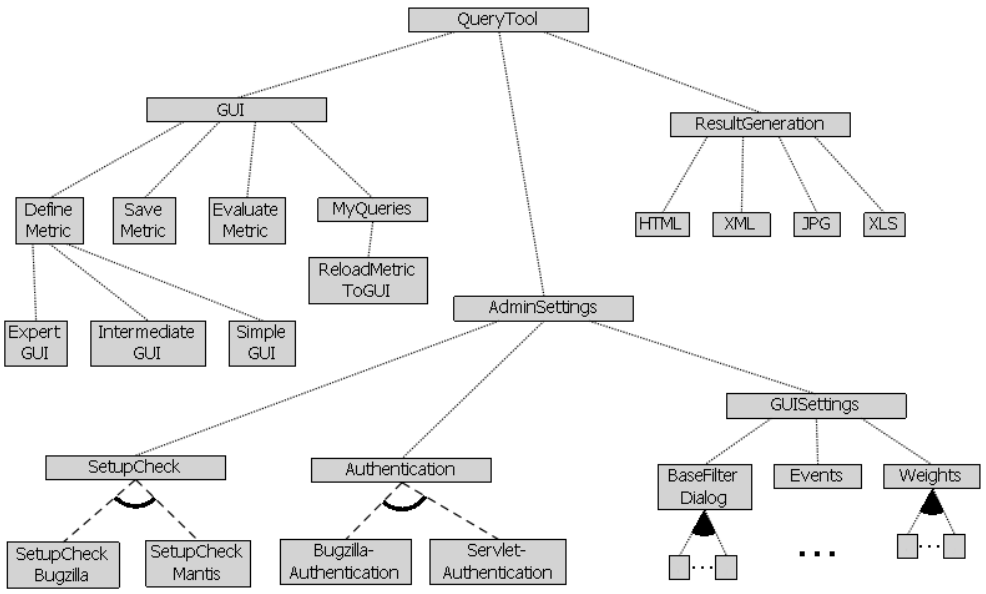


Abbildung 7.3.: Feature-Diagramm für das Metrik-Abfrage-Werkzeug (Legende siehe Tabelle 7.1)

7.1.2. Metrik-Abfrage-Werkzeug

Ein Feature-Modell für das Metrik-Abfrage-Werkzeug ist in Abbildung 7.3 gegeben. Das wichtigste Feature aus Anwendersicht ist auf der linken Seite zu sehen: die grafische Benutzeroberfläche zur Definition und Abfrage von Metriken (*GUI*). Metriken können auf mehreren Wegen definiert werden, angepasst an unterschiedliche Erfahrungsstufen der Benutzer. Dies reicht von der einfachen Anpassung vordefinierter Metrik-Spezifikationen (*Simple GUI*), über

die Generierung von Metrik-Spezifikationen mit Hilfe einer grafischen Benutzeroberfläche (*Intermediate GUI*), bis hin zur direkten Bearbeitung von Metrik-Spezifikationen in ITMS (*Expert GUI*). Weiterhin können Metrik-Spezifikationen gespeichert und wieder geladen werden.

Das auf der rechten Seite des Diagramms gezeigte Feature *ResultGeneration* zeigt die unterschiedlichen Ausgabeformate für Ergebnisse der Metrikberechnung. Nähere Details zu Anforderungen an die Benutzeroberfläche des Metrik-Abfrage-Werkzeugs finden sich in der Arbeit von Obaid [Oba07].

Weitere Features beziehen sich auf die Sicht des Administrators der Web-Anwendung (siehe Feature *AdminSettings*). Der *Setup Check* ermöglicht eine Überprüfung der serverseitigen Konfiguration. Die Art dieser Überprüfungen ist abhängig von den zur Verfügung stehenden Datenquellen.

Optional kann eingestellt werden, dass Anwender authentifiziert werden müssen. Wie im Feature *Authentication* erkennbar, stehen dazu verschiedene alternative Verfahren bereit.

Schließlich muss konfiguriert werden, welche Ereignisse und Gewichtungen an der grafischen Benutzeroberfläche zur Auswahl stehen, und welche Dialogelemente zur Festlegung des Basisfilters dargestellt werden (siehe Feature *GUISettings*). Die Auswahl dieser Elemente wird beeinflusst durch die zur Metrikberechnung herangezogenen Datenquellen (vgl. Abb. 7.5).

7.1.3. Qualitätsmodell-Werkzeuge

In Abbildung 7.4 findet sich schließlich ein Feature-Modell für die Werkzeugunterstützung im Bereich der Qualitätsmodelle. Diese kann wiederum untergliedert werden in das Editieren von Qualitätsmodellen (*Quality Model Editor*) und die Qualitätsbewertung mit Hilfe von Qualitätsmodellen (*Quality Evaluation Tool*).

Der Editor für Qualitätsmodelle bietet eine grafische Benutzeroberfläche zum Erstellen und Editieren hierarchisch strukturierter Modelle bestehend aus Qualitätsmerkmalen und Qualitätsindikatoren (vgl. Abschnitt 6.3). Zur Festlegung von Details dieser Elemente steht jeweils ein Assistentendialog zur Verfügung. Im Falle der Qualitätsindikatoren muss hier eine Verbindung zu einem Messwerkzeug hergestellt werden. Dazu kann die Metrikberechnungskomponente für ITMS Metrik-Spezifikationen angebunden werden. Bei Qualitätsmerkmalen muss jeweils festgelegt werden, wie sich der Wert für das Qualitätsmerkmal auf Basis der untergeordneten Qualitätsmerkmale- oder Qualitätsindikatoren berechnet. Schließlich kann mit dem Editor für Qualitätsmodelle auch deren Konsistenz und Vollständigkeit überprüft werden, und mögliche Fehler oder Warnungen angezeigt werden.

Das Auswertungswerkzeug bietet ebenfalls einen Assistenten, der den Anwender über mehrere Dialoge durch die Konfiguration einer Metrik-Auswertung leitet. Dazu muss zunächst ein gegebenes Qualitätsmodell ausgewählt werden.

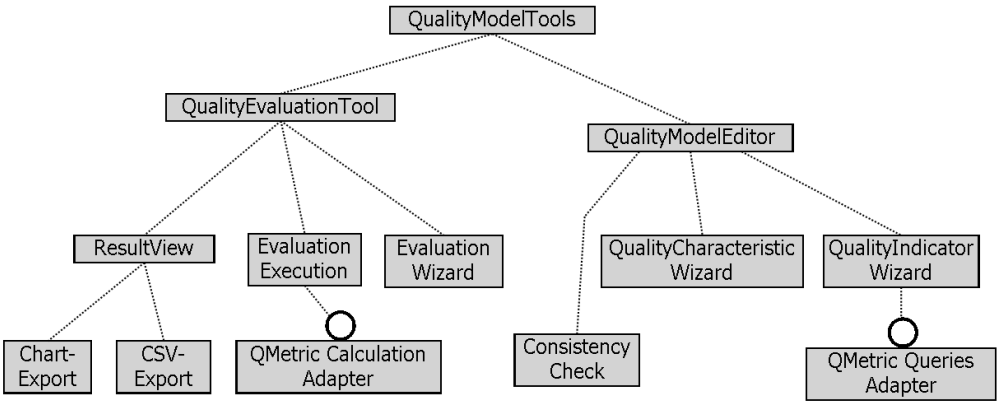


Abbildung 7.4.: Feature-Diagramm für die Qualitätsmodell-Werkzeuge (Legende siehe Tabelle 7.1)

Mittels eines Zustandsfilters werden die Messobjekte bestimmt. Dann muss der Auswertungszeitraum bestimmt werden. Optional können Messobjekte und ein Zeitraum bestimmt werden, die als empirische Vergleichsdaten herangezogen werden.

Auf Basis der gegebenen Konfiguration können mit dem Auswertungswerkzeug dann die Messwerte für das Qualitätsmodell ermittelt und die Werte für die einzelnen Qualitätsmerkmale bestimmt werden. Zur Erhebung der Messwerte müssen externe Messwerkzeuge aufgerufen werden. Hier steht als eine Option wiederum die Anbindung an die Metrikberechnungskomponente für ITMS Metrik-Spezifikationen zur Verfügung. Schließlich können die Ergebnisse der Auswertung dem Benutzer angezeigt werden, wobei wiederum unterschiedliche Formate für den Export von Ergebnissen zur Verfügung stehen.

Weitere Einzelheiten zu den Anforderungen an diese beiden Werkzeuge finden sich in der Arbeit von Jansen [Jan09].

7.2. Architektur

Wie im vorherigen Abschnitt angedeutet, sind die Erweiterbarkeit und Anpassbarkeit zwei wesentliche Ziele bei der Entwicklung der Werkzeug-Suite. Anpassbarkeit bezieht sich auf die in den Feature-Modellen eingeführten Variabilitätspunkte. Erweiterbarkeit kann sich beispielsweise auf eine Erweiterung des Sprachumfangs von ITMS oder Erweiterungen des Metamodells für Qualitätsmodelle beziehen.

Beim Entwurf der Werkzeuge muss also eine geeignete technische Umsetzung der Variabilität gefunden werden [SvGB05]. Zudem sollen die Werkzeuge möglichst lose gekoppelt sein, und es soll auf etablierte technische Standards zurückgegriffen werden.

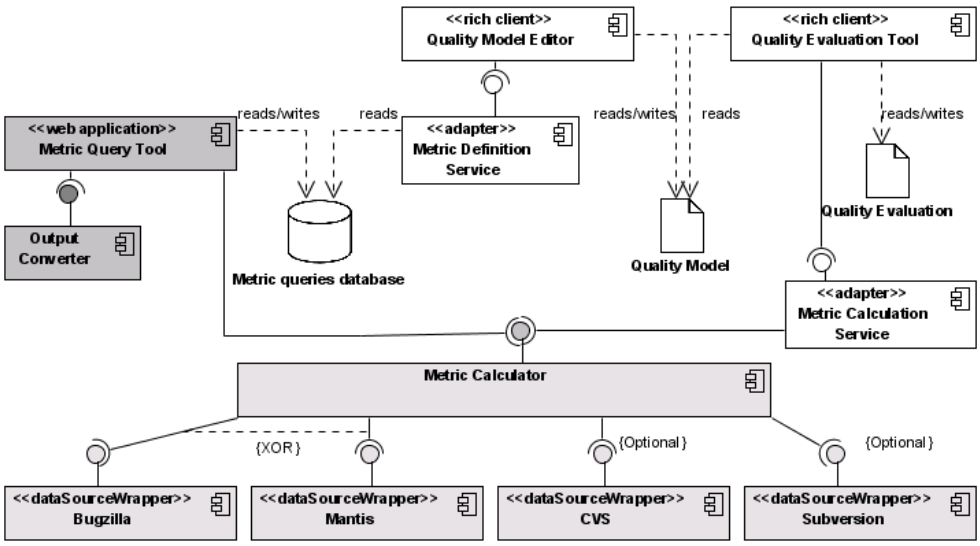


Abbildung 7.5.: Überblick über die Architektur der QMetric-Werkzeug-Suite

7.2.1. Überblick

Abbildung 7.5 gibt eine Übersicht über die Architektur der QMetric-Werkzeug-Suite. Im Kern steht die Metrikberechnungskomponente (*Metric Calculator*). Unterschiedliche Datenquellen können durch die als *DataSourceWrapper* markierten Adapter angebunden werden. Die Schnittstelle der Metrikberechnungskomponente erwartet eine ITMS Metrik-Spezifikation als Eingabe. Als Ergebnisse der Metrikberechnung werden Zeitreihen im XML-Format zurückgegeben.

Die Komponente *Metric Query Tool* stellt eine Web-Anwendung zur Definition und Abfrage von Metriken zur Verfügung. Serverseitig wird dazu eine ITMS Metrik-Spezifikation an die Metrikberechnungskomponente übergeben. Die Komponente *Output-Converter* konvertiert Berechnungsergebnisse in alternative Ausgabeformate. Metrik-Spezifikationen können in einer Datenbank abgelegt werden, und über das *Metric Query Tool* organisationsweit zur Verfügung gestellt werden.

Schließlich gibt es die beiden Qualitätsmodell-Werkzeuge. Der *Quality Model Editor* unterstützt die Erstellung von Qualitätsmodellen. Zur Definition von Qualitätsindikatoren wird auf die mit dem Metrik-Abfrage-Werkzeug gespeicherten Metriken zurückgegriffen.

Ein gegebenes Qualitätsmodell kann von der Komponente *Quality Evaluation Tool* ausgelesen werden. Dieses Werkzeug ermöglicht die schrittweise Definition einer Auswertung auf Basis des Qualitätsmodells. Die Konfiguration der Auswertung und die Messergebnisse werden wiederum in einer eigenen Datei abgelegt.

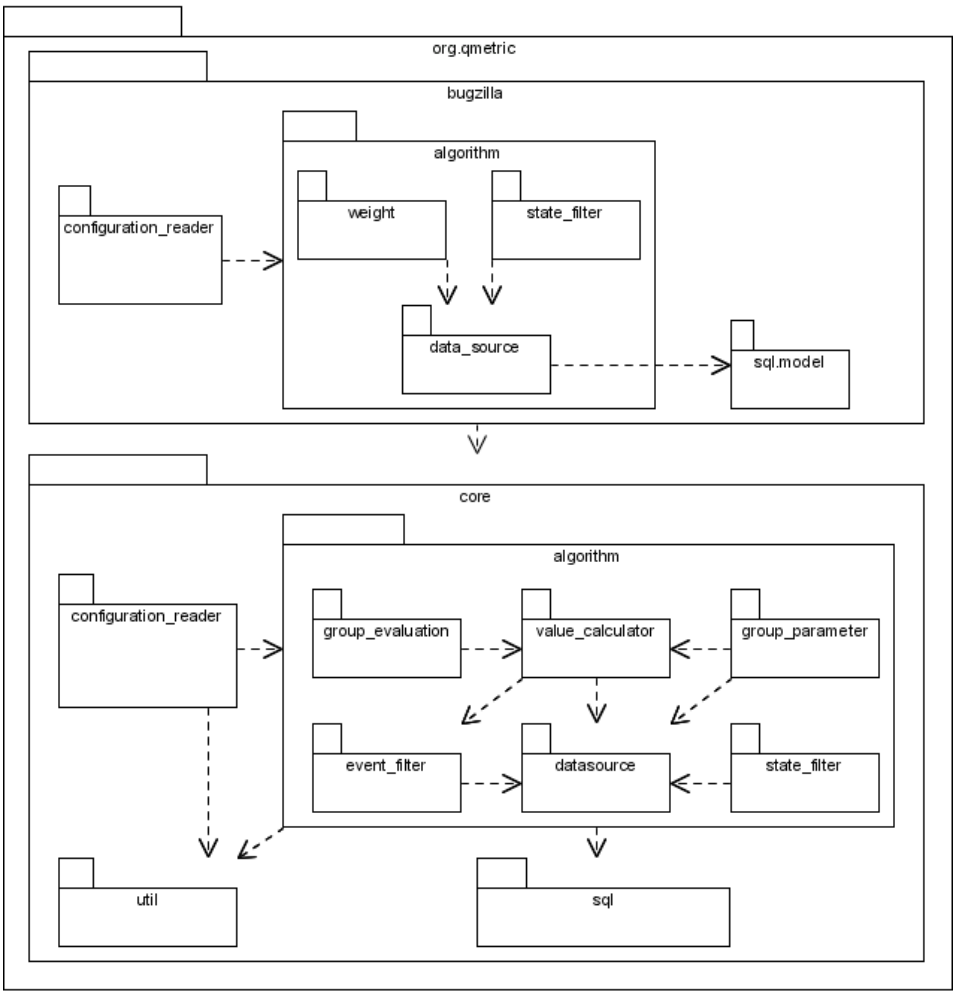


Abbildung 7.6.: Paketdiagramm: Metrikberechnungskomponente und Adapter für Bugzilla

Da die beiden Qualitätsmodell-Werkzeuge als Client-Anwendung implementiert sind und nur lose mit den übrigen Teilen der Werkzeug-Suite gekoppelt sein sollen, erfolgt die Anbindung der Metrik-Datenbank und der Metrikberechnungskomponente über einen XML-RPC Webservice, dargestellt in den Komponenten *Metric Definition Service* und *Metric Calculation Service*.

7.2.2. Metrikberechnungskomponente

Im Folgenden soll zunächst der strukturelle Aufbau der Metrikberechnungskomponente vorgestellt, und anschließend der Ablauf des Berechnungsalgorithmus skizziert werden.

Das Paketdiagramm in Abbildung 7.6 zeigt im unteren Bereich die Pakete der Metrikberechnungskomponente (*core*) und ihre Abhängigkeiten. Im oberen Be-

reich sind am Beispiel des Pakets *bugzilla* Erweiterungen zur Anbindung eines konkreten Issue-Tracking-Systems dargestellt. Die Subpakete im Paket *bugzilla* sind analog zu den Subpaketen in *core* benannt. Dadurch wird deutlich, welche Klassen der Metrikberechnungskomponente erweitert werden.

Zunächst wird hier das Paket *core* beschrieben. In Abschnitt 7.2.2.1 wird dann das Paket *bugzilla* vorgestellt.

Allgemein verwendbare Basisklassen beispielsweise für Zeitangaben, oder zur Anbindung einer *Logging*-Bibliothek sind im Paket *util* enthalten. Das Paket *sql* enthält Klassen zur Anbindung der Datenbank und zur Erzeugung von SQL-Anweisungen.

Das Paket *configuration_reader* enthält Klassen zum Parsen einer im XML-Format gegebenen Metrik-Spezifikation. Als Ergebnis des Parse-Vorgangs wird ein Objekt der Klasse *Metric* erzeugt. In einem solchen Objekt sind für die Berechnung benötigten Elemente, wie beispielsweise das Bewertungsverfahren sowie Filter und Gewichtungen konfiguriert. Die Klassen zur Repräsentation einer Metrik und ihrer Elemente finden sich im Paket *algorithm* und dessen Unterpaketen.

Wie schon in Abschnitt 7.1.1 dargestellt wurde, gibt es eine ganze Reihe von Variationspunkten, die auch in der Architektur geeignet zu berücksichtigen sind [SvGB05]. Zum einen können unterschiedliche Software-Entwicklungsarchive als Datenquellen eingebunden werden. Daraus ergeben sich dann auch unterschiedliche Ereignisse, Filter, und Gewichtungen für die Metrikberechnungen. Zum anderen können bei der Definition neuer Metriken neue Arten von Berechnungen benötigt werden, beispielsweise weitere Gruppenwertberechnungen (siehe auch [GSL07a]).

Die im im Paket *algorithm* enthaltenen Unterpakete *state_filter*, *event_filter*, *group_evaluation*, *group_parameter* und *value_calculator* entsprechen jeweils den verschiedenen Elementen einer Metrik-Spezifikation (vgl. Abschnitt 4.5). Jedes dieser Pakete enthält eine Klasse, welche die allgemeine Schnittstelle eines solchen Elements (beispielsweise Filter oder Gewichtungsfunktion) vorgibt. Für Elemente, die unabhängig von den ausgewerteten Software-Entwicklungsarchiven sind, sind bereits konkrete Klassen implementiert. Dazu gehören beispielsweise die Bewertungsverfahren, die Gruppenwertberechnungen sowie bestimmte Ereignisfilter und Zustandsfilter. Für ein konkretes auszuwertendes Archiv können entsprechende Unterklassen abgeleitet werden.

Um solche Erweiterungen für die Metrikberechnungskomponente bekannt zu machen, wird eine *Singleton*-Klasse verwendet. Der Name der Klasse muss in der Konfigurationsdatei der Metrikberechnungskomponente angegeben werden. Die Schnittstelle der Klasse enthält Methoden, mit deren Hilfe die Berechnungskomponente abfragt, welche Datenquellen verwendet werden. Die Variationspunkte werden also erst zur Startzeit der Anwendung aufgelöst. Somit kann die Berechnungskomponente für unterschiedliche Datenquellen angepasst werden, ohne sie neu zu kompilieren.

7.2.2.1. Anbindung eines konkreten Issue-Tracking-Systems

Im oberen Bereich von Abbildung 7.6 sind als Beispiel die ergänzenden Klassen für Auswertungen auf dem Issue-Tracking-System Bugzilla gezeigt. Das Paket *sql.model* enthält eine abstrakte Beschreibung für das Datenbankmodell von Bugzilla. Im Paket *algorithm.data_source* sind dann auf dieser Basis alle Attribute beschrieben, die ausgewertet werden können. Durch die Abstraktion von einem konkreten Datenbankschema ist das Werkzeug ohne weitere Anpassung für unterschiedliche Versionen des Bugzilla-Datenbankschemas verwendbar. Es reicht die Angabe der Bugzilla-Version in der Konfigurationsdatei.

Weiterhin ergeben sich Gewichtungen und Filter, die spezifisch für das Issue-Tracking-System Bugzilla sind. Diese sind in den Paketen *weight* und *state_filter* enthalten. Das Paket *configuration_reader* enthält ergänzende Klassen, um Bugzilla-spezifische Elemente der Metrik-Spezifikation zu parsen.

7.2.2.2. Berechnungsalgorithmus

Die Metrikberechnungskomponente wird mit einer ITMS-Metrik-Spezifikation als Argument aufgerufen. Der Algorithmus zur Berechnung der Metriken läuft dann in den folgenden Schritten ab:

1. Beim **Parsen der Metrik-Spezifikation** wird eine Objektstruktur erzeugt, die die Metrik repräsentiert. Ein Objekt der Klasse *Metric* ist der Einstiegspunkt zu dieser Objektstruktur.

2. **Abfrage der Datenbank**: Nur die wirklich für die Berechnung der Metrik erforderlichen Daten werden aus der Datenbank abgefragt. Es werden dann Objekte für die betrachteten Änderungsanträge erzeugt, und mit der aktuellen Belegung der benötigten Attribute initialisiert.

3. **Ermittlung der Bewertungszahlen**: Die Berechnung läuft dann zeitlich gesehen rückwärts. Es werden dabei die relevanten Ereignisse im Lebenslauf der Änderungsanträge analysiert. Die Belegung der Attribute zu früheren Zeitpunkten wird dabei rekonstruiert. Ereignisse, die konform sind zu einem der Ereignisfilter in den verwendeten Bewertungsfunktionen, stoßen die Berechnung von Bewertungszahlen an.

4. **Gruppenwertberechnung**: Die im vorherigen Schritt ermittelten Bewertungszahlen werden für jede Zeitperiode mit Gruppenwertberechnungen zusammengefasst.

5. **Erzeugung des Ergebnis-Dokuments**: Die Ergebnisse werden in Form eines XML-Dokuments zurückgegeben, welches die Zeitreihen enthält. Diese sind gegebenenfalls nach bestimmten Attributwerten gruppiert.

Dieser Algorithmus abstrahiert also davon, wie die benötigten Informationen aus den Software-Entwicklungsarchiven gespeichert sind. Die Berechnung erfolgt

auf den Attributen und Ereignissen, die in den jeweiligen *data_ source*-Klassen abstrakt beschrieben sind.

7.2.3. Anbindung von Konfigurationsmanagement-Systemen

Die Verknüpfung von Änderungen im Quellcode mit den dazugehörigen Änderungsanträgen verbessert die Rückverfolgbarkeit von Änderungen und die Dokumentation der Änderungsanträge [MR05]. Metrikauswertungen, die diese Verknüpfung zwischen Konfigurationsmanagement-System und Issue-Tracking-System nutzen, können ebenfalls nützliche Informationen liefern. Beispielsweise kann ein Problembericht durch die Größe von Änderungen oder die Anzahl der betroffenen Dateien charakterisiert werden.

Für die Metrikauswertung auf Konfigurationsmanagement-Systemen steht eine Reihe von Werkzeugen zur Verfügung [DP03, ED]. Ansätze, die Informationen aus Konfigurationsmanagement-System und Issue-Tracking-Systemen zu kombinieren, zielen in der Regel auf spezialisierte Fragestellungen ab, und bieten keine allgemeine Schnittstelle zur Spezifikation und Abfrage von Metriken (vgl. Abschnitt 2.4.2).

Dies war die Motivation, um QMetric um Auswertungen auf Konfigurationsmanagement-Systemen zu ergänzen. Dabei wurden die beiden gängigen Konfigurationsmanagement-Systeme CVS und Subversion betrachtet. Für die Herstellung der Verknüpfungen wird vorausgesetzt, dass das Werkzeug *Scmbug* verwendet wird. *Scmbug* ist eine generische Integrationslösung für Issue-Tracking-Systeme und Konfigurationsmanagement-Systeme [MR05].

Konfigurationsmanagement-Systeme bieten typischerweise eine Log-Schnittstelle, mit welcher die Änderungshistorie abgefragt werden kann. Die Log-Ausgaben der Änderungshistorie sind unterschiedlich strukturiert. CVS arbeitet dateiorientiert, Subversion arbeitet revisionsorientiert. Es hat sich gezeigt, dass eine direkte Abfrage dieser Informationen auf Seiten der Konfigurationsmanagement-Systeme zu viel Zeit in Anspruch nimmt, um interaktive Metrikabfragen durchzuführen. Daher wurde es notwendig, die benötigten Informationen aus dem Konfigurationsmanagement-System in eine eigene Datenbank zu importieren. Diese Datenbank abstrahiert von den konkreten Konfigurationsmanagement-Systemen. Die Datenbank enthält Informationen zu den Entitäten Datei, Ordner, *Branch* und *Tag*. Weiterhin enthält sie die auf den Entitäten durchgeführten Aktivitäten, beispielsweise Änderung einer Datei, Verschieben eines Ordner oder Hinzufügen eines *Tags*. Zu jeder dieser Aktivitäten werden die für die Metrik-Auswertung relevanten Informationen erfasst, beispielsweise der Entwickler, der Zeitpunkt und die Anzahl der geänderten, hinzugefügten oder gelöschten Zeilen.

Für den Importvorgang wird eine betriebssystemunabhängige Zugriffsschicht verwendet, die Unterschiede beim Aufruf der Log-Befehle verbirgt. Die Log-Dateien werden spezifisch für das jeweilige Konfigurationsmanagement-System geparst und in eine systemunabhängige Datenstruktur eingelesen. Auf dieser

Datenstruktur wird dann eine Reihe von Verarbeitungsschritten durchgeführt. Dazu gehören das Verknüpfen von Ordnern und den darin enthaltenen Dateien und Unterordnern, die Rekonstruktion von Aktivitäten, bei denen Dateien verschoben wurden, sowie die Erstellung von Referenzen zwischen Aktivitäten und den davon betroffenen Elementen. Schließlich wird das Ergebnis in der Datenbank gesichert.

Die Erweiterung der Berechnungskomponente gestaltet sich dann analog zu der Anpassung auf ein spezifisches Issue-Tracking-System. Zu Änderungsanträgen können zusätzliche Attribute ausgewertet werden, wie die von dem Änderungsantrag betroffenen Dateien, Ordner, *Branches* und *Tags*, sowie die Entwickler, die in Bezug auf den Änderungsantrag eine Änderung durchgeführt haben. Diese Attribute sind mehrwertige Attribute, das heißt, alle historischen Werte sind bei einer Abfrage gültig (vgl. Abschnitt 4.4.1.1).

Weiterhin kommen zusätzliche Ereignisse hinzu, die sich auf die oben bereits erwähnten Aktivitäten auf den im Konfigurationsmanagement-System verwalteten Objekten beziehen. Mit einem Ereignisfilter können dann solche Änderungsanträge erfasst werden, für die im Konfigurationsmanagement ein bestimmtes Ereignis stattfand, beispielsweise das Anlegen eines Branches, oder das Erstellen einer neuen Datei.

Schließlich kommen neue Gewichtungsfunktionen hinzu, beispielsweise die Anzahl der geänderten Code-Zeilen oder die Anzahl der betroffenen Dateien. Dabei ist zu beachten, dass eine solche Gewichtung sich immer auf eine bestimmte Kategorie von Ereignissen bezieht, beispielsweise eine Änderung an einer Datei. Nicht alle Kombinationen von Ereignissen und Gewichtungsfunktionen können also sinnvoll eingesetzt werden.

Abbildung 7.7 zeigt die hier beschriebenen Komponenten für Auswertungen über Bugzilla und Subversion. Die *Scmbug*-Integration wird über *hook*-Methoden an Subversion angebunden, und greift auf Bugzilla über eine API zu. Der Import verwendet die Log-Schnittstelle von Subversion. Zusätzlich muss bei *Scmbug* die Abbildung zwischen den Benutzernamen aus Bugzilla und Subversion abgefragt werden. Auf die Bugzilla-Datenbank wird über eine JDBC-Schnittstelle zugegriffen. Die Metrikberechnungskomponente wird über ein Konfigurationsobjekt konfiguriert (*CoreConfiguration*). Sie greift nur auf die Bugzilla-Datenbank sowie auf die Datenbank der aus Subversion extrahierten Informationen zu. Nähere Details zur Architektur dieser Lösung finden sich in der Arbeit von Lischkowitz [Lis08].

7.2.4. Web-Anwendung zur Abfrage von Metriken

Wie bereits in Abschnitt 7.1.2 gefordert, soll die Web-Anwendung für Benutzer mit unterschiedlicher Erfahrung bedienbar sein. Dazu stehen in Form von Kategorieseiten unterschiedliche Dialoge zur Verfügung:

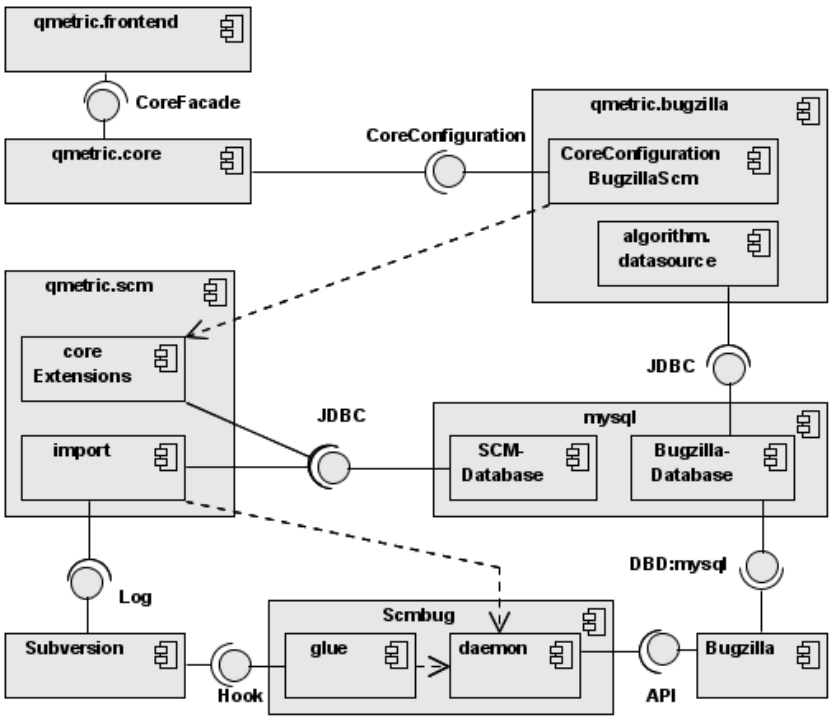


Abbildung 7.7.: Komponentendiagramm: Beteiligte Systeme im Kontext der Auswertung von Bugzilla und Subversion

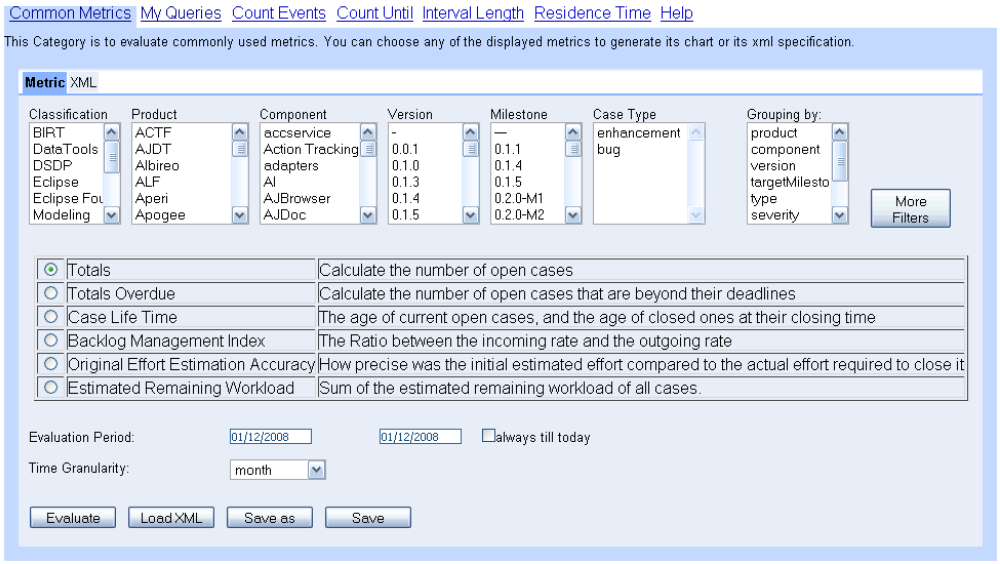


Abbildung 7.8.: Dialog zur Auswahl häufig genutzter Metriken

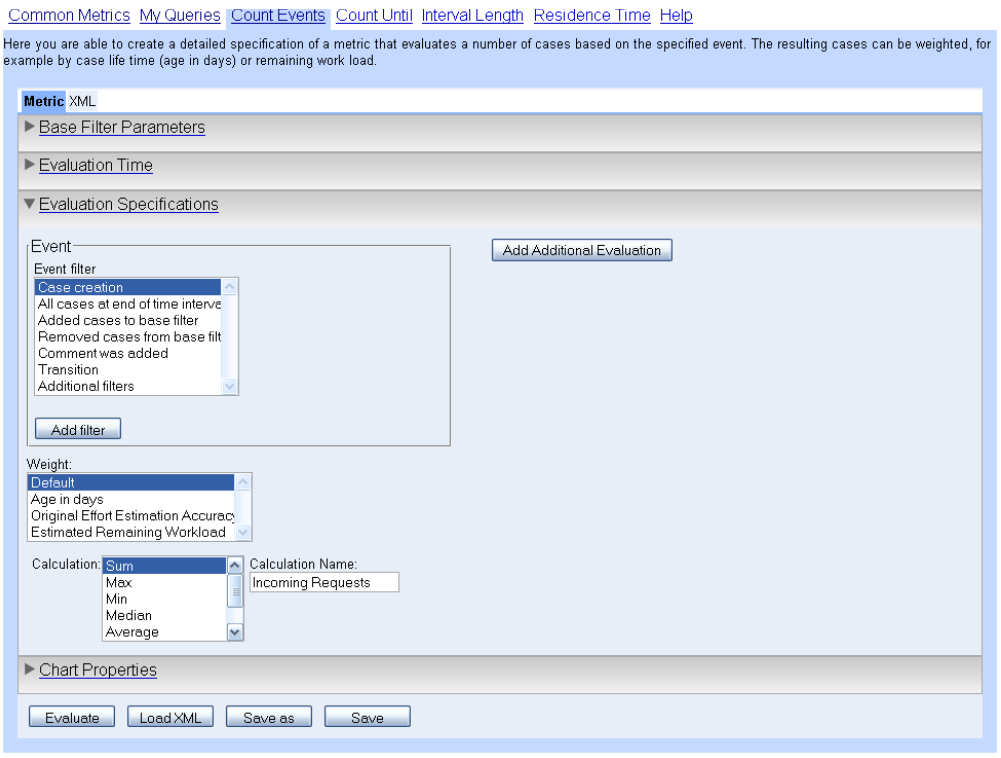


Abbildung 7.9.: Dialog zur Konfiguration des Bewertungsverfahrens Zählen von Ereignissen (*CountEvents*)

- Häufig genutzte Metriken können in einem einfachen Dialogfenster direkt ausgewählt werden (siehe Abb. 7.8). Zusätzlich kann der Basisfilter sowie Berechnungszeitraum- und Granularität angepasst werden.
- Für jede der vier Bewertungsverfahren (vgl. Abschnitt 4.4.2.4) steht eine Kategorieseite bereit, um die dazu nötigen Filter und Gewichte mit Hilfe von grafischen Dialogen zu konfigurieren (siehe Abb. 7.9).
- Für alle über die Dialogfenster konfigurierten Metriken kann in eine Ansicht der Metrik im XML-Format gewechselt werden. Dies erleichtert es neuen Anwendern, die Sprachelemente von ITMS kennenzulernen.
- Schließlich gibt es einen Dialog, um auf gespeicherte Metriken zuzugreifen. Metrik-Spezifikationen können entweder für alle Anwender zugänglich, oder nur für den jeweiligen Anwender gespeichert werden. Metrik-Spezifikationen können auch wieder in die jeweilige Kategorieseite geladen werden, aus der sie erzeugt wurden.

Beim Entwurf der Kategorieseiten wurden unter anderem die folgenden Entwurfsmuster für interaktive Systeme eingesetzt [Tid05]:

- *Visual Framework*: Jede Kategorieseite folgt einem ähnlichen Layout, sodass sich der Anwender schnell zurecht findet.

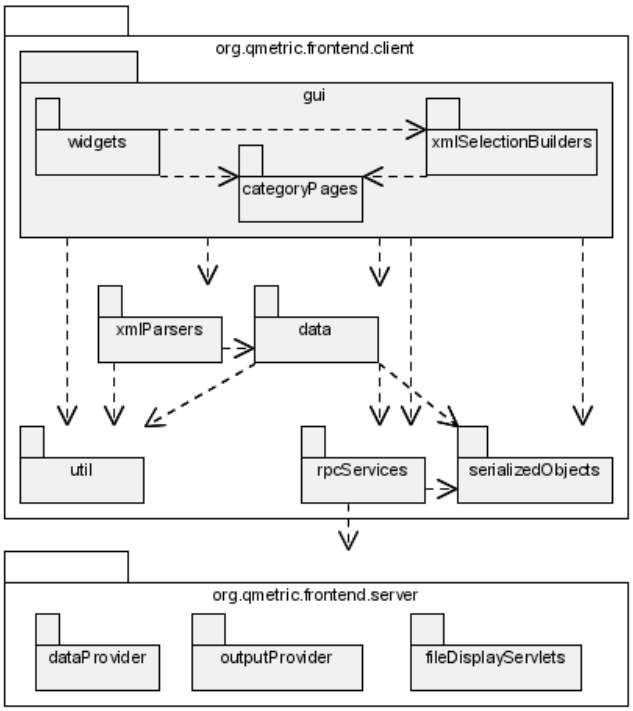


Abbildung 7.10.: Paketdiagramm der Web-Anwendung zur Abfrage von Metriken

- *Extras on demand*: Bestimmte Elemente in den Kategorieseiten werden nur dann eingeblendet, wenn der Anwender diese explizit anfordert, beispielsweise ergänzende Dialogelemente zur Definition des Basisfilters oder zusätzliche Ereignisfilter.

- *Card Stack*: Bei der Definition von Metrik-Abfragen über grafische Dialoge sind diese in einzelne Abschnitte aufgeteilt, die jeweils ein- und ausgeblendet werden können. Damit wird es dem Anwender erleichtert, die Übersicht zu behalten. Gleichzeitig wird nicht eine bestimmte Reihenfolge der Bearbeitung erzwungen.

Technisch basiert die Web-Anwendung auf dem Google Web Toolkit (GWT), einem Rahmenwerk zur Entwicklung von AJAX-Anwendungen [GWT]. Der Begriff AJAX (*Asynchronous JavaScript and XML*) bezeichnet Techniken, um Inhalte von Web-Anwendungen asynchron im Hintergrund zu laden, ohne eine komplette Seite neu laden zu müssen. Dies ermöglicht desktop-ähnliche interaktive Web-Anwendungen.

Mit GWT wird auch der clientseitige Code in der Sprache Java entwickelt, sodass die gewohnte Entwicklungsumgebung und die Werkzeuge für den Test und das Debugging zur Verfügung stehen. Mittels eines GWT-Compilers wird dann ausführbarer JavaScript-Code für alle gängigen Browser erzeugt.

Abbildung 7.10 zeigt die Paketstruktur der Web-Anwendung. Den Konventionen aus GWT folgend gibt es eine Aufteilung in die Pakete *client* und *server*. Das Paket *client.rpcServices* enthält die Schnittstellenbeschreibungen für asynchrone Methodenaufrufe an den Server. Diese werden bei der Compilierung nach JavaScript in das proprietäre Protokoll GWT-RPC übertragen. Das Paket *serializedObjects* enthält Klassen, deren Objekte als Parameter oder Rückgabewert beim Aufruf von serverseitigen Methoden erscheinen. Durch den Einsatz von GWT ist die Serialisierung und Deserialisierung dieser Objekte transparent.

Das Paket *gui* enthält Klassen zum Aufbau der grafischen Benutzungsschnittstelle. Diese sind in drei Pakete gegliedert. Im Paket *categoryPages* findet sich eine Klassenhierarchie zum Aufbau der einzelnen Seiten für die jeweiligen Kategorien. Gemeinsam genutzte Dialogelemente, beispielsweise für die Auswahl von Ereignisfiltern oder die Festlegung des Berechnungszeitraums, finden sich im Paket *widgets*. Klassen zur Generierung von Metrik-Spezifikationen auf Basis der Auswahl in einem der Dialoge sind im Paket *xmlSelectionBuilders* enthalten. Um umgekehrt gespeicherte Metrik-Spezifikationen in eine Kategorienseite zu laden, werden Klassen aus dem Paket *xmlParsers* verwendet. Zum Befüllen einiger Dialogfelder müssen Informationen aus dem Issue-Tracking-System geladen werden, beispielsweise die Namen der verfügbaren Produkte und Komponenten. Klassen zur Abfrage solcher Daten und zum Laden von Einstellungen für das Aussehen von konfigurierbaren Elementen im Dialogfenstern finden sich im Paket *data*.

Das Paket *server* enthält die Implementierungen der Service-Schnittstellen, beispielsweise für das Laden- und Speichern von Metrik-Spezifikationen, oder das Aufrufen von Metrikberechnungen. Im Paket *outputProvider* finden sich Klassen zum Konvertieren eines Metrik-Ergebnisdokuments im Format XML in andere Formate. Im Paket *fileDisplayServlets* sind entsprechende Servlets zur Anzeige von Dateien dieser Formate enthalten. Schließlich finden sich im Paket *dataProvider* einige Klassen zur Prüfung der Konfiguration des Servers sowie zum Laden von Informationen aus dem zu Grunde liegenden Issue-Tracking-System. Ähnlich wie bei der Metrikberechnungskomponente, wird auch die Web-Anwendung serverseitig über Angabe einer von *DataProvider* abgeleiteten Klasse konfiguriert.

7.2.5. Editor und Auswertungswerkzeug für Qualitätsmodelle

Die Werkzeugunterstützung für Qualitätsmodelle ist zweigeteilt. Es gibt ein Werkzeug zum Erstellen von Qualitätsmodellen, sowie ein Werkzeug zum Auswerten eines gegebenen Qualitätsmodells auf Basis von Metriken. Beide Werkzeuge sind als *Plug-Ins* für die Entwicklungsumgebung Eclipse implementiert. Somit kann auf bewährte Konzepte für die Arbeit mit komplexen Modellen zurückgegriffen werden, unter anderem *Views* und Perspektiven, das Konzept des *Workspace*, Assistenten zum Erstellen neuer Dateien, und eine Anbindung an

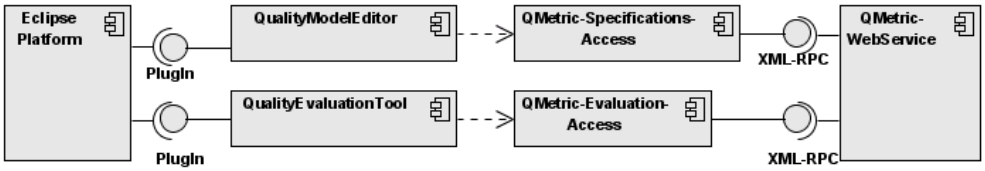


Abbildung 7.11.: Komponentendiagramm: Editor und Auswertungswerkzeug für Qualitätsmodelle

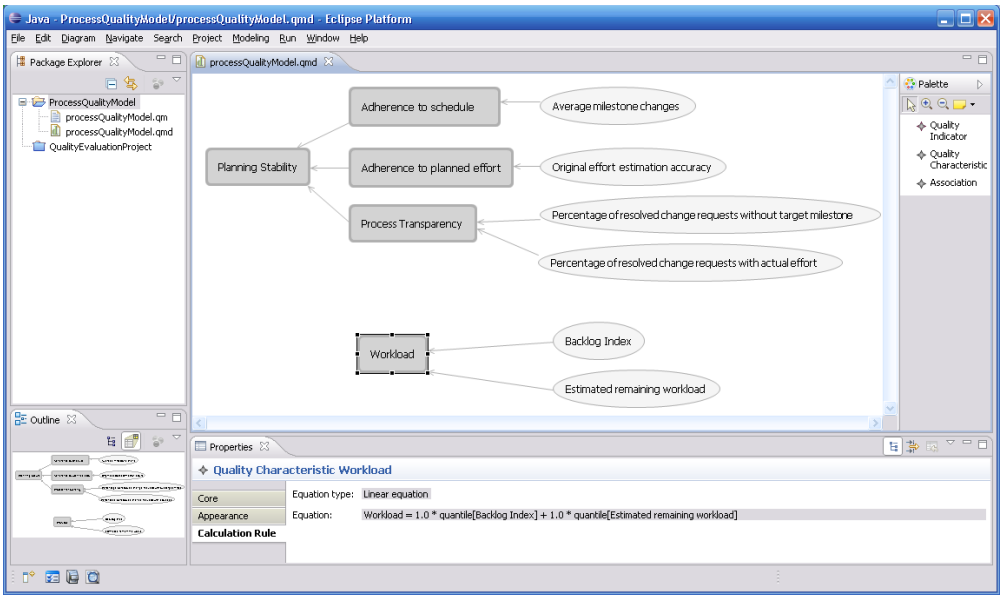


Abbildung 7.12.: Der Qualitätsmodell-Editor

Konfigurationsmanagement-Systeme. Durch die Realisierung als Plug-Ins können die Werkzeuge sowohl in einer integrierten Umgebung als auch als aufgabenspezifsche Werkzeuge bereitgestellt werden.

Abbildung 7.11 zeigt die Komponenten im Umfeld dieser beiden Werkzeuge. Der Qualitätsmodell-Editor verfügt über eine Anbindung an QMetric, um gespeicherte Metriken laden zu können. Das Auswertungswerkzeug kann solche Metriken über eine eigene Anbindung berechnen lassen. Über einen Web-Service wird jeweils auf die benötigten Daten auf einem Server zugegriffen.

Im Folgenden soll ein Überblick über die Architektur der beiden Werkzeug-Komponenten gezeigt werden.

7.2.5.1. Qualitätsmodell-Editor

Abbildung 7.12 zeigt die grafische Oberfläche des Qualitätsmodell-Editors. Das Werkzeug basiert auf Technologien aus dem Eclipse-Umfeld. Zur Bearbeitung von Qualitätsmodellen wird das *Eclipse Modeling Framework* (EMF) eingesetzt

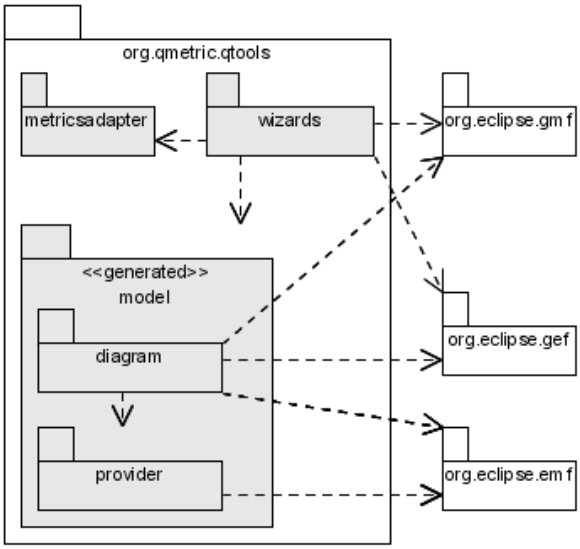


Abbildung 7.13.: Paketdiagramm zum Qualitätsmodell-Editor

[BSM+03]. Mit EMF kann für ein gegebenes Metamodell Programmcode generiert werden, der es unterstützt, Instanzen dieses Modells anzulegen, zu manipulieren, zu validieren und zu speichern. In unserem Fall dient das in Abschnitt 6.3 vorgestellte Metamodell als Ausgangspunkt zur Code-Generierung mit EMF.

Der grafische Editor für Qualitätsmodelle baut auf dem *Graphical Modeling Framework* (GMF) auf [GMF]. GMF ermöglicht die modellbasierte Entwicklung von grafischen Editoren für *Ecore*-basierte formale Modelle. Die grafische Darstellung und zugehörige Interaktionen basieren auf dem *Graphical Editing Framework* (GEF) [GEF]. Somit kann ein großer Teil der Anwendung generativ entwickelt werden. Das Metamodell für Qualitätsmodelle wird dazu um die Definition der grafischen Elemente sowie um Werkzeuge zum Erstellen neuer grafischer Elemente ergänzt. Dieser Ansatz erleichtert es, spätere Anpassungen des Metamodells in die Werkzeugunterstützung zu übertragen.

Die erstellten Qualitätsmodelle werden im standardisierten XMI-Format abgespeichert und stehen so potentiell auch für die Verwendung in anderen Werkzeugen zur Verfügung.

Die zugehörige Paketstruktur ist in Abbildung 7.13 zu sehen. Das Paket *model* enthält hauptsächlich den mittels GMF generierten Quellcode. Das Paket *provider* enthält Klassen zur Anbindung an die Eclipse-Umgebung, beispielsweise für die Anzeige von Modellelementen im *Properties View*. Im Paket *diagram* sind die Klassen des grafischen Editors enthalten.

Die Bearbeitung von Details des Qualitätsmodells, beispielsweise die Festlegung von Metriken für Qualitätsindikatoren oder der Berechnungsregeln zur Bewertung von Qualitätsmerkmalen, werden mit Hilfe von Assistenten durchgeführt. Die zugehörigen Klassen sind im Paket *wizards* enthalten.

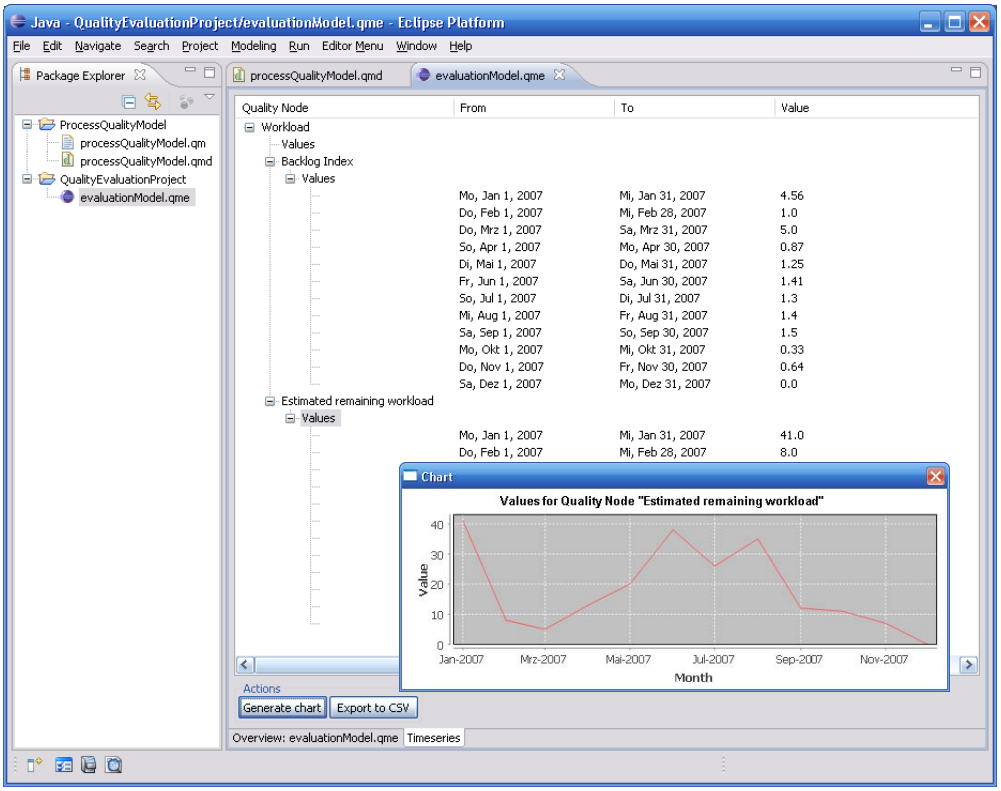


Abbildung 7.14.: Anzeige von Ergebniszeitreihen im Auswertungswerkzeug

Zur Definition von Qualitätsindikatoren kann eine Liste von bereits definierten Metriken aus der Datenbank des Metrik-Abfrage-Werkzeugs angezeigt werden. Die entsprechende Anbindung an den Web-Service ist im Paket *metricsadapter* implementiert.

7.2.5.2. Auswertungswerkzeug

Grundkonzept des Auswertungswerkzeugs ist, dass auf Basis eines gegebenen Qualitätsmodells ein eigenes Auswertungsmodell definiert wird. Das Auswertungsmodell enthält die Einstellungen zur konkreten Durchführung einer metrikbasierten Qualitätsbewertung. Dazu gehören die Bestimmung des Messobjekts und optional die Auswahl von Vergleichsdaten. Bei der Verwendung von QMetric wird das Messobjekt über einen Basisfilter definiert (vgl. Abschnitt 4.5). In diesem kann beispielsweise das betrachtete Produkt, Projekt oder eine Komponente eingegrenzt werden. Weiterhin müssen der Zeitraum und die Zeitperioden für die Qualitätsbewertung festgelegt werden. Vergleichsdaten werden ebenfalls durch einen Zustandsfilter und durch Angabe eines Vergleichzeitraums bestimmt.

Bei der Durchführung einer Qualitätsbewertung wird das Auswertungsmodell mit den Zeitreihen der Messergebnisse ergänzt. Eine Alternative wäre gewesen, diese wiederum in ein eigenes Ergebnismodell einzutragen. Darauf wurde

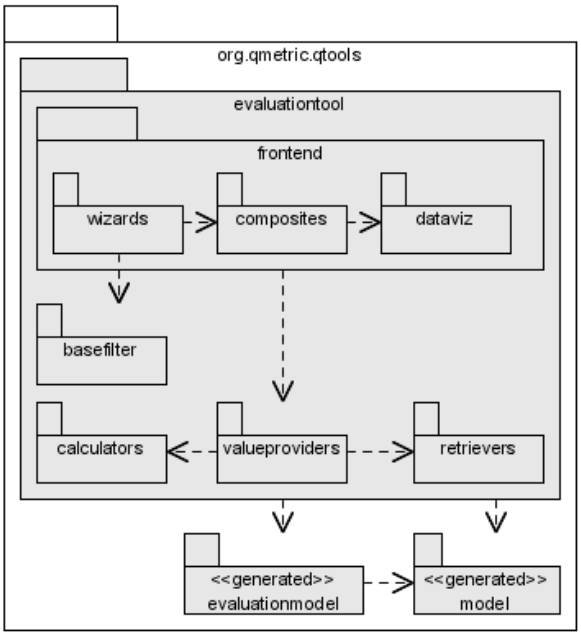


Abbildung 7.15.: Paketdiagramm zum Auswertungswerkzeug

verzichtet, um dem Anwender den Überblick über die erstellten Modelle zu erleichtern.

Die Erzeugung und Bearbeitung von Auswertungsmodellen stützt sich technisch wiederum auf das Rahmenwerk EMF. Abbildung 7.15 zeigt das Paketdiagramm zum Auswertungswerkzeug. Das Paket *evaluationmodel* ist mittels EMF generiert und enthält die Klassen zur Beschreibung und Bearbeitung von Auswertungsmodellen. Ein Auswertungsmodell muss dazu ein zugeordnetes Qualitätsmodell referenzieren. Daraus ergibt sich die Abhängigkeit zu dem bereits im vorherigen Abschnitt vorgestellten Paket *model*.

Alle grafischen Dialoge des Auswertungswerkzeugs finden sich im Paket *composites*. Das Paket *wizards* enthält Assistentendialoge zur Konfiguration eines Auswertungsmodells.

Bei Verwendung der QMetric-Berechnungskomponente als Metrik-Werkzeug soll dem Anwender ein Dialog zur Konfiguration des Basisfilters zur Verfügung stehen, ähnlich dem schon vom Abfrage-Werkzeug gewohnten Dialog. Das dort verwendete *Widget*-Paket des GWT-Frameworks unterstützt allerdings nur grafische Oberflächen in einer Browser-Umgebung. Deswegen konnten diese Klassen nicht direkt wiederverwendet werden. Stattdessen wurden die Dialoge nochmals auf Basis des SWT-Rahmenwerks implementiert (Paket *basefilter*). Das Paket *composites* enthält Registerseiten zur Anzeige von Messergebnissen. Im Paket *dataviz* finden sich schließlich Klassen zur Visualisierung der Ergebnisse als Diagramme sowie zum Export in das Format CSV (siehe auch Abb. 7.14).

Das Auswertungswerkzeug kann alle benötigten Berechnungen zur Qualitätsbe-

wertung anstoßen. Das Paket *retrievers* enthält dazu Klassen zur Anbindung von Messwerkzeugen. Eine konkrete Anbindung ist der Zugriff auf die QMetric-Berechnungskomponente über einen Web-Service. Das Pakete *calculators* enthält die Klassen zur Durchführung der für Qualitätsmerkmale definierten Berechnungen. Im Paket *valueproviders* finden sich Fassaden-Klassen zum Zugriff auf die beiden Arten der Wertermittlung.

7.3. Testumgebung

Wie bereits in Kapitel 5 eingeführt, gehört zur Validierung von Metriken auch die Prüfung des Messwerkzeugs selbst [KPF95]. Im Falle der QMetric-Werkzeug-Suite muss überprüft werden, ob die Berechnungskomponente die Metrikberechnungen entsprechend der definierten Semantik korrekt durchführt. In diesem Abschnitt wird ein Testansatz vorgestellt, der darauf abzielt, genau diese Eigenschaft sicherzustellen.

Der Ansatz folgt der Teststrategie *Data-Driven Test* [Mes07]. Die Daten für das Setup, die Test-Durchführung und den Ergebnisvergleich werden in Dateien vorgehalten. Ein Testrahmenwerk lädt diese Dateien und führt den eigentlichen Test durch. Konkret werden die folgenden Dateien verwendet.

- Datensätze zur Initialisierung der Datenbanken für die verwendeten Software-Entwicklungsarchive
- Auszuwertende Metrik-Spezifikation
- Soll-Resultat der Metrikberechnung gegeben als XML-Dokument
- Soll-Werte für die bei der Metrikberechnung durchgeführten SQL-Abfragen

Diese Dateien werden voneinander unabhängig abgelegt, sodass sie wiederverwendet und kombiniert werden können. Die optionale Prüfung der von der Metrikberechnungskomponente generierten SQL-Abfragen ermöglicht es, Probleme bei der Optimierung dieser Anfragen zu erkennen.

Das Testrahmenwerk ist unabhängig von den beim Test verwendeten Software-Entwicklungsarchiven. Die *Setup*-Methoden zur Initialisierung der Datenbank müssen jeweils für die zu Grunde liegenden Software-Entwicklungsarchive spezialisiert werden. Um die Tests hierarchisch abzulegen, wird eine Verzeichnisstruktur genutzt. Diese wird bei der Testdurchführung automatisch durchlaufen. Vor Durchführung jedes Tests wird die Datenbank bereinigt, um Wechselwirkungen zwischen verschiedenen Tests zu vermeiden (vgl. das Test-Entwurfsmuster *Fresh Fixture* [Mes07]).

Bei der Definition der Tests wird eine Abdeckung der Sprachelemente von ITMS angestrebt. Jedes Sprachelement von ITMS wird mindestens in einem Test verwendet. Weiterhin werden wichtige Kombinationen von Sprachelementen getestet, sowie die unterschiedlichen Parametrisierungen der Bewertungsverfahren. Bei der Initialisierung der Software-Entwicklungsarchive werden bestimmte

Äquivalenzklassen und Grenzfälle für die Metrikberechnung nachgebildet. Zum Test von ITMS für Bugzilla werden insgesamt über 200 Testfälle eingesetzt.

Für den Import von Daten aus Konfigurationsmanagement-Systemen wird ein ähnliches Testrahmenwerk eingesetzt. Dazu wird in jedem Testfall ein Konfigurationsmanagement-System neu aufgesetzt und bestimmte Aktionen darauf durchgeführt. Dann wird die Änderungshistorie abgefragt und in eine Datenbank importiert. Der Inhalt der Datenbank wird schließlich mit einem Soll-Resultat verglichen.

Schließlich wird noch ein Stress-Test verwendet, der Abfragen von mehreren gleichzeitig aktiven Benutzern simuliert, um Aspekte wie die Threadsicherheit der Berechnungskomponente zu prüfen.

Die folgende Tabelle zeigt die erreichte Testüberdeckung beim Berechnungsalgorithmus für ITMS-Metriken.

Tabelle 7.2.: Erreichte Überdeckung bei dem Paket *org.qmetric.core.algorithm* [Cov]

Überdeckungskriterium	Überdeckungsgrad
Anweisungsüberdeckung	92,0%
Zweigüberdeckung	82,7%
Bedingungsüberdeckung	78,9%

Eine Anweisungsüberdeckung von 100% wird nicht erreicht. Nicht überdeckt werden *Exception Handler* für spezielle Probleme in der Umgebung, wie etwa eine fehlende Konfigurationsdatei oder Inkonsistenzen in der Datenbank. Weiterhin redefinieren einige Klassen die *toString*-Methoden. Diese Methoden werden allerdings nur vom Debugger für die Ausgabe von überwachten Ausdrücken verwendet, sodass sie beim Test ebenfalls nicht überdeckt werden.

Der *Data-Driven* Testansatz erleichterte die inkrementelle Erweiterung der Test-Suites. Zum Teil können die Tests für unterschiedliche Entwicklungsarchive wiederverwendet werden. Nach den bisherigen Erfahrungen besteht somit ein gutes Sicherheitsnetz für Änderungen an der Metrik-Berechnungskomponente.

8. Evaluierung

All models are wrong,
but some are useful.

(George E.P. Box, 1919 -)

Inhaltsangabe

Eine realistische Bewertung der entwickelten Sprache und der zugehörigen Konzepte und Werkzeuge kann nur im Rahmen einer praktischen Anwendung erfolgen. Dazu werden in Abschnitt 8.1 ausgewählte Fallstudien für die praktische Anwendung vorgestellt. Die dabei gesammelten Erfahrungen fließen in die Bewertung der entwickelten Konzepte und Werkzeuge ein (Abschnitte 8.2 bis 8.5). In Abschnitt 8.6 werden die Ergebnisse schließlich den in Kapitel 3 formulierten Zielen gegenübergestellt.

8.1. Praktische Anwendung

Die folgenden Abschnitte berichten über die Anwendung der entwickelten Konzepte und Werkzeuge zur Bewertung von Entwicklungsprozessen, zum einen im Open Source Bereich, zum anderen im industriellen Umfeld.

8.1.1. Bewertung von Open Source Entwicklungsprozessen

Wie bereits in Abschnitt 2.4.2 erwähnt, bieten die Software-Entwicklungsarchive von Open Source Projekten eine breite Datenbasis für die praktische Anwendung von Auswertungsansätzen. Die in dieser Arbeit entwickelten Konzepte wurden unter anderem in Fallstudien im Kontext von Open Source Software eingesetzt. Dabei wurde die Prozessqualität von *GNOME*-Projekten [SL09], sowie von *Eclipse*-Projekten [Sch09, SSL09] betrachtet. Eine weitere Studie befasst sich mit der Beteiligung von Anwendern beim Issue Tracking im Kontext von Open Source im Vergleich zum *Open Commercial* Entwicklungsansatz [GST+],

der von IBM im Rahmen des Projekts *Jazz* eingeführt wurde. Im Folgenden sollen überblicksartig das Vorgehen und Ergebnisse aus der Fallstudie zu Eclipse dargestellt werden.

Um die Prozesse im Umfeld von Eclipse zu bewerten, muss zunächst geklärt werden, was die Ziele und Vorgaben des eingesetzten Entwicklungsprozesses sind. Die *Eclipse Foundation* verwendet einen eigenen agilen Entwicklungsansatz der unter der Bezeichnung „*Eclipse Way*“ bekannt ist [Gam05]. Der Ansatz basiert auf einer Reihe sich ergänzender Praktiken. Zu den wesentlichen Zielen zählen die Vorhersagbarkeit, im Sinne der Einhaltung von geplanten Release-Zeitpunkten, sowie eine durchgängig hohe Qualität auch von Zwischen-Releases. Ein weiterer Schwerpunkt liegt darauf, die Anwendergemeinde einzubinden, um frühzeitig Feedback zu erhalten. Dazu muss schnell und angemessen auf Änderungsanträge der Anwender reagiert werden. Zudem muss der Entwicklungsprozess transparent sein, beispielsweise muss öffentlich bekannt sein, zu welchem Meilenstein bestimmte Features eingeplant sind.

Weitere Vorgaben gibt es außerdem noch für den Umgang mit dem eingesetzten Issue-Tracking-System Bugzilla [EBU]. Beispielsweise sollen neu eingegangene Änderungsanträge innerhalb eines Tages geprüft und klassifiziert werden.

Ausgehend von diesen Prozessvorgaben und -zielen wurden Qualitätsmerkmale identifiziert. Dabei ergab sich, dass die Qualitätsmerkmale nach den folgenden unterschiedlichen Sichtweisen strukturiert werden können:

- Strukturierung aus Sicht der unterschiedlichen Beteiligten: Es kann unterschieden werden zwischen Qualitätsmerkmalen des Prozesses aus Sicht der Entwickler und aus Sicht der Anwender [Sch09].
- Strukturierung nach den verschiedenen Vorgaben für den Prozess: Für Eclipse existiert zum einen eine Anleitung für die Verwendung von Bugzilla, zum anderen gibt es Vorgaben aus dem Eclipse-eigenen Entwicklungsansatz [Sch09].
- Strukturierung orientiert an Schritten im Change Request Prozess [SSL09]: Dabei unterscheiden wir die Qualität der eingehenden Änderungsanträge, die Qualität bei der Klassifizierung von Änderungsanträgen, bei der Planung von Änderungsanträgen, bei der Abarbeitung und schließlich bei der Prüfung von bearbeiteten Änderungsanträgen.

Die Qualitätsmerkmale wurden gegebenenfalls weiter verfeinert und schließlich auf bestimmte Fragen zurückgeführt. Abbildung 8.1 zeigt ein Beispiel für ein solches Qualitätsmodell. Die als Qualitätsindikatoren verwendeten Metriken sind in Tabelle 8.1 kurz erläutert.

Im Folgenden soll nicht das komplette Qualitätsmodell diskutiert werden, sondern das Vorgehen anhand eines Beispiels verdeutlicht werden. Dazu möchten wir die Qualität der Klassifikation von eingehenden Änderungsanträgen betrachten (*Triage*). Es liegt die Annahme zu Grunde, dass die Anwender eine schnelle und angemessene Reaktion auf Änderungsanträge erwarten. Wenn Anwender den Eindruck gewinnen, dass ihre Änderungsanträge keine Auswirkung haben,

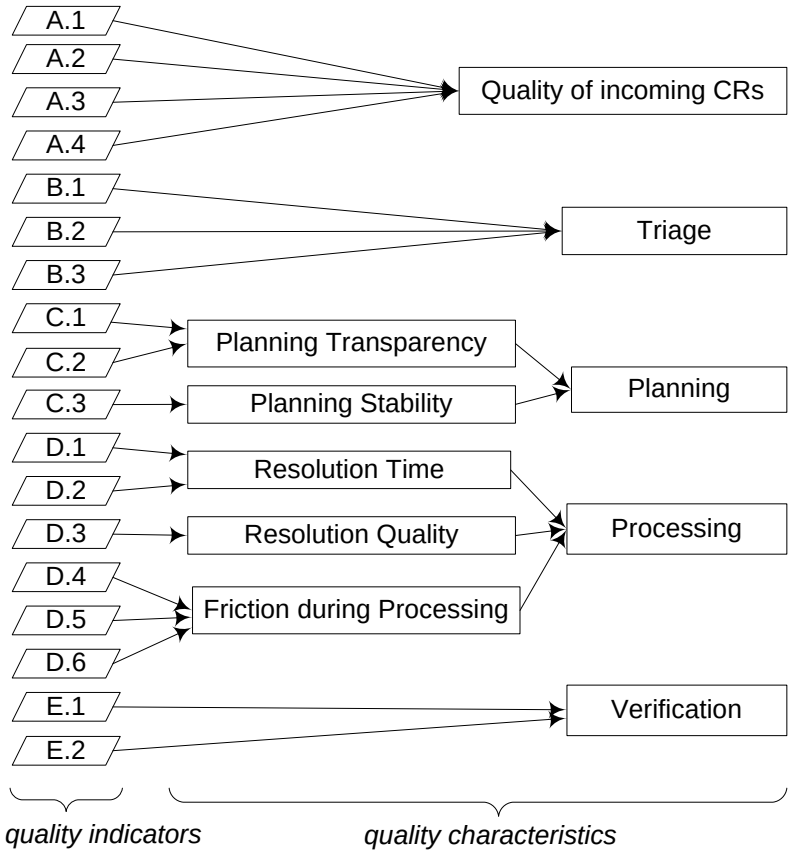


Abbildung 8.1.: Qualitätsmodell zu Eclipse strukturiert nach den Schritten im Change Request Prozess [SSL09]

Tabelle 8.1.: Metriken zum Qualitätsmodell aus Abbildung 8.1 [SSL09] (CR steht als Abkürzung für Change Request.)

Id	Metric	Description
A.1	Duplicated CRs	Number of CRs marked as *Duplicate* relative to the number of all resolved CRs in a time interval.
A.2	Invalid CRs	Number of CRs marked as *Invalid* relative to the number of resolved CRs in a time interval.
A.3	Defect reports without version	Number of reported defects with unspecified version number relative to the number of all reported defects in a time interval.
A.4	Comments before leaving status *New*	Average number of comments before a CR changes into status *Assigned* or *Resolved* for the first time.
B.1	CRs with no reaction within 2 days	Percentage of CRs created in a time interval where the first reaction takes longer than 2 days.
B.2	Reopened rate of rejected CRs	Number of triaged CRs with resolution *Duplicate, Invalid, NotEclipse, WontFix,* or *WorksForMe* that have been reopened, relative to the number of rejected CRs in a time interval.
B.3	Priority of severe bugs	Average priority of CRs with severity *critical* or *blocker* that had been resolved in a time interval.
C.1	Assigned without milestone	Number of CRs that change into status *Assigned* with no valid target milestone relative to the number of all CRs that change into *Assigned*.
C.2	Fixed without milestone	Number of CRs that are fixed and have no valid target milestone relative to the number of all CRs fixed in a time interval.
C.3	Frequency of milestone changes	Number of changes to defined target milestones relative to the number of CRs with a defined target milestone.
D.1	Time until fixed	Median age in days of CRs that change into the status *Resolved/Fixed*.
D.2	High Priority Lifetime Ratio	Average lifetime of fixed CRs with priority *P1* relative to the average lifetime of all fixed CRs.
D.3	Reopened Rate of fixed CRs	Number of fixed CRs that are reopened relative to the number of fixed CRs in a time interval.
D.4	High priority CRs	Number of fixed CRs with priority *P1* relative to the number of all CRs resolved in a time interval.
D.5	Average Assignee Changes	Number of assignee changes relative to the number of CRs assigned in a time interval.
D.6	Enhancements during Endgame	Number of enhancement requests fixed during the Endgame phase relative to the number of all enhancement requests fixed in the release cycle.
E.1	Closed/ Resolved Ratio	Number of closed CRs in a time interval relative to the number of resolved CRs in a time interval.
E.2	Closed without Verified	Number of closed CRs in a time interval that had not been in the status *Verified*.

werden sie wahrscheinlich nicht weiter zu dem Projekt beitragen. Das Qualitätsmerkmal kann mit den folgenden Fragen genauer charakterisiert werden:

- Wie schnell wird auf eingehende Änderungsanträge reagiert?
- Werden Änderungsanträge korrekt klassifiziert?
- Wird bei der Priorisierung von Änderungsanträgen der vom Anwender angegebene Schweregrad berücksichtigt?

Dazu wurden die folgenden Metriken mittels ITMS spezifiziert [SSL09]. Die zugehörigen Metrik-Spezifikationen finden sich in Anhang A.1.

1. **Änderungsanträge ohne Reaktion in zwei Tagen**: Prozentsatz der in einem Zeitintervall eingegangenen Änderungsanträge, bei denen die erste Reaktion erst nach zwei Tagen oder später erfolgte.
2. **Falsch negativ klassifizierte Änderungsanträge**: Anzahl der Änderungsanträge die zurückgewiesen wurden und später wieder geöffnet wurden, relativ zur Anzahl aller zurückgewiesenen Änderungsanträge. Ein Änderungsantrag gilt dabei als zurückgewiesen, wenn er als ungültig, nicht reproduzierbar oder als Duplikat eines anderen Änderungsantrags klassifiziert wurde.
3. **Priorität von Problemberichten mit hohem Schweregrad**: Durchschnittliche Priorität von Problemberichten mit den beiden höchsten Schweregraden *Critical* oder *Blocker*.

Zur Interpretation der Metrikergebnisse im Qualitätsmodell wurde ein empirischer Vergleich verwendet. Die Vergleichsgruppe waren die 29 Projekte aus Eclipse, die am *Simultaneous Release* im Juni 2008 teilnahmen. Als Vergleichszeitraum wurde der Entwicklungszeitraum für das *Ganymede* Release vom 1. Juli 2007 bis zum 30. Juni 2008 gewählt. In der Untersuchung wurde dann die Qualitätsentwicklung von 2004 bis 2008 untersucht, wobei sich die betrachteten Zeitperioden am jährlichen Release-Zyklus orientieren.

Zunächst kann die Werteverteilung innerhalb der Vergleichsgruppe betrachtet werden. Dies ermöglicht bereits einen allgemeinen Eindruck über die Produkte. Als Beispiel sind in der folgenden Tabelle die Quartile für die Metrik „Priorität von Problemberichten mit hohem Schweregrad“ angegeben.

	Minimum	1.Quartil	Median	3.Quartil	Maximum
Durchschnittliche Priorität	1,15	2,54	2,92	3,00	3,00

Es zeigt sich, dass sowohl der Medianwert mit 2,92 als auch das untere Quantil mit 2,54 nahe an der standardmäßig vergebenen Prioritätsstufe 3 liegen. Somit spiegelt sich der vom Ersteller eines Änderungsantrags vergebene Schweregrad häufig nicht in der Priorisierung wieder.

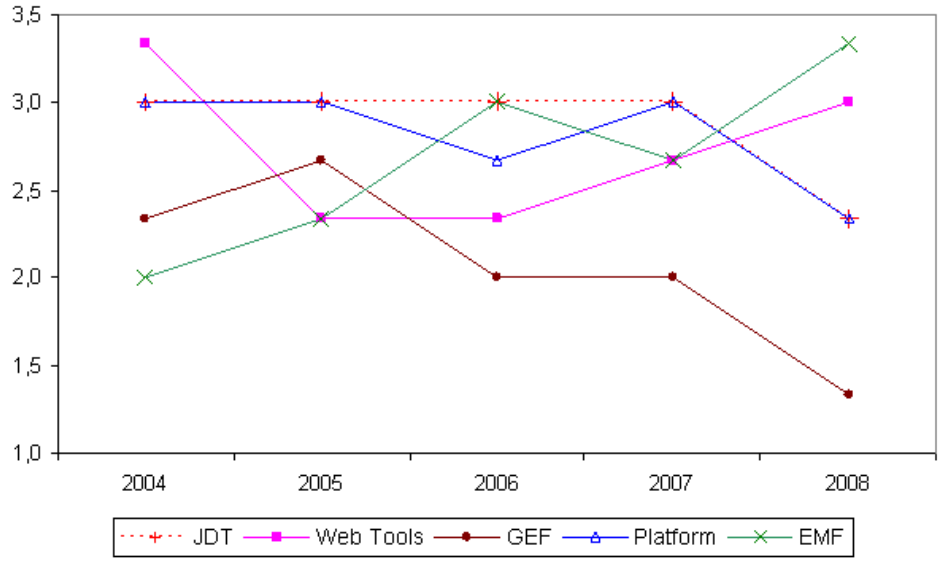


Abbildung 8.2.: Qualität der Klassifikation von Änderungsanträgen bewertet auf einer Ordinalskala von 1 bis 4

Bei der Aggregation der Messwerte zu einem Qualitätsmerkmal wurde jeweils die Einordnung nach Quantilen der Vergleichsgruppe verwendet, und dabei jeweils eine Indexzahl zwischen 1 und 4 vergeben. Der Wert 4 besagt, dass ein Produkt besser abschneidet als 75% der Produkte in der Vergleichsgruppe. Der Wert 1 besagt, dass das Produkt schlechter abschneidet als 75% der Produkte in der Vergleichsgruppe.

Bei dem Qualitätsmerkmal „Klassifikation von eingehenden Änderungsanträgen“ wurden alle drei Metriken gleich gewichtet. Der aggregierte Wert für das Qualitätsmerkmal wurde wiederum auf den Wertebereich 1 bis 4 normalisiert, wobei die eben gegebene Interpretation erhalten bleibt. Der Wert 2,5 kann als durchschnittliche Qualität interpretiert werden.

Abbildung 8.2 zeigt die Entwicklung des Qualitätsmerkmals „Klassifikation von eingehenden Änderungsanträgen“ für eine Reihe von bekannten Produkten aus Eclipse. Zur Validierung dieses Ergebnisses muss die im Diagramm erkennbare Qualitätsentwicklung bei den Produkten mit entsprechenden Experteneinschätzungen verglichen werden. Eine vollständige Prüfung ist dabei unrealistisch, da dazu Expertenwissen über die Prozesse bei allen betrachteten Projekten notwendig ist.

Basierend auf den im Rahmen der Entwicklung einer auf EMF [BSM+03] und GEF [GEF] basierten Werkzeugumgebung [ViP, Nyß09] gesammelten Erfahrungen können wir die Aussagen für diese beiden Eclipse-Produkte bestätigen. Bei Änderungsanträgen zu EMF erfolgte eine schnelle Reaktion, und Probleme wurden schnell behoben. Dagegen konnten bei GEF in den letzten Jahren

zunehmend zu späte, keine oder unpassende Reaktionen auf eingestellte Änderungsanträge beobachtet werden. Zudem kamen bei GEF in den Releases seit 2007 wenig neue Features hinzu, und der Schwerpunkt lag mehr auf der Behebung von bekannten Fehlern.

Des Weiteren wurde im Rahmen der Fallstudie untersucht, ob auf Basis empirischer Vergleiche unterschiedliche Qualitätsstufen der Produkte unterschieden werden können [Sch09]. Allerdings zeigte sich, dass die Produkte sehr heterogen bezüglich der betrachteten Qualitätsmerkmale sind. Somit würde ein Großteil der Produkte nur in der untersten Qualitätsstufe eingeordnet, da jeweils die Mindestanforderungen bei einem bestimmten Qualitätsmerkmal nicht erfüllt sind.

8.1.2. Analyse von industriellen Softwareentwicklungsprozessen

Eine weitere Fallstudie widmete sich der Analyse von industriellen Softwareentwicklungsprozessen bei unserem Kooperationspartner KISTERS AG [Ric09]. Im Bereich Umweltinformatik der KISTERS AG werden unter anderem Informationssysteme für den Energiemarkt, die Wasserwirtschaft sowie den Umwelt- und Arbeitsschutz entwickelt. In der Studie wurde die Prozessqualität bei der Entwicklung und Wartung verschiedener Produkte analysiert. Ausgangspunkt waren die erfassten Erweiterungswünsche, Problemberichte und Support-Anfragen. Die dabei betrachteten Produkte sind recht unterschiedlich hinsichtlich ihrer Größe, ihres Alters, und ihrer Entwicklungsdynamik, beispielsweise eine Neuentwicklung, die Weiterentwicklung eines etablierten Produkts, oder ein zunehmend von Wartungsaufgaben geprägtes Produkt.

In der Arbeit von Richlofsky wurde für die betrachteten Prozesse bei der KISTERS AG ein eigenes Qualitätsmodell entwickelt [Ric09]. Die dabei betrachteten Informationsbedürfnisse waren zum einen motiviert aus Referenzmodellen zur Prozessbewertung wie beispielsweise CMMI und ITIL (vgl. Abschnitt 2.3), zum anderen abgeleitet aus den Verbesserungszielen für die Softwareentwicklung aus dem Qualitätsmanagementsystem der KISTERS AG. Ein Überblick über dieses Modell findet sich im Anhang A.2. Das Qualitätsmodell betrachtet Qualitätsmerkmale in den folgenden Bereichen:

1. **Management der Anforderungen**: Zunächst werden hier die Größenordnungen der eingehenden Änderungsanträge betrachtet, jeweils klassifiziert nach Quelle und nach Typ des Änderungsantrags. Weiterhin wird untersucht, wie zuverlässig die initiale Klassifikation des Typs ist, und wie häufig parallele oder doppelte Arbeit an inhaltlich verwandten Änderungsanträgen auftritt.

2. **Planung**: In diesem Bereich wird untersucht, inwieweit geplanter Fertigstellungstermin und Aufwand dokumentiert wird, und wie gut diese Planung eingehalten werden kann.

3. **Überwachung und Verfolgung des Entwicklungsfortschritts**: Der Entwicklungsprozess gibt eine Reihe von Bearbeitungszuständen für Änderungsanträge vor, um den Fortschritt der Bearbeitung sichtbar zu machen. Es wird entsprechend betrachtet, wie diese Bearbeitungszustände genutzt werden.
4. **Lösungsgeschwindigkeit**: Bei diesem Qualitätsmerkmal wird zum einen die Dauer bis zum Abschluss der Bearbeitung eines Änderungsantrags, zum anderen die Dauer bis zur Abnahme durch den Kunden untersucht.
5. **Software-Qualitätssicherung**: Im Bezug auf die Qualitätssicherung wird beispielsweise der Umfang und Dauer für ungeplante Nacharbeiten, sowie die Anzahl der im Feldeinsatz aufgetretenen Probleme betrachtet.
6. **Software-Konfigurationsmanagement**: Hier wird betrachtet, inwieweit Vorgaben für das Konfigurationsmanagement eingehalten werden, wie etwa die Angabe eines geplanten Meilensteins sowie die Dokumentation von Änderungen für die *Release Notes*.
7. **Koordination**: In diesem Bereich wird der für Änderungsanträge benötige Koordinationsaufwand untersucht, bezogen auf die Kommunikation zwischen den Beteiligten sowohl innerhalb der Organisation als auch in der Kommunikation mit dem Kunden.
8. **Supportprozess**: In diesem Bereich wird die Anzahl der Sofortlösungen von Support-Anfragen betrachtet.
9. **Fire-Fighting-Aktivitäten**: Unter *Fire-Fighting*-Aktivitäten verstehen wir ungeplant auftretende Änderungsanträge mit höchster Dringlichkeit. Es wird die Häufigkeit des Auftretens solcher Aktivitäten betrachtet, sowie die durchschnittliche Lösungszeit der zugeordneten Änderungsanträge.
10. **Kundenzufriedenheit**: Schließlich werden als Teilaspekte der Kundenzufriedenheit die Kommunikation über ein Kundenportal sowie die Einhaltung von Terminen gegenüber dem Kunden untersucht.

Ähnlich dem im vorherigen Abschnitt vorgestellten Beispiel wurde für jeden Bereich eine Reihe von Fragestellungen abgeleitet, und für jede Fragestellung eine oder mehrere Metriken entwickelt. Die Metriken wurden jeweils im Kontext der in der Organisation gelebten Prozesse validiert, entsprechend dem in Kapitel 5 beschriebenen Vorgehen. Dabei waren sowohl zeitliche Änderungen im gelebten Prozess, als auch die Unterschiede bei den verschiedenen Produkten zu berücksichtigen.

Zur Analyse der Messergebnisse wurde kein empirischer Vergleich vorgenommen. Da keine geeigneten externen Vergleichsdaten vorlagen, hätte ein Vergleich innerhalb des Produkt-Portfolios der KISTERS AG stattfinden müssen. Allerdings war zum einen bei einem Teil der betrachteten Produkte die Anzahl der eingehenden Änderungsanträge zu gering, um sinnvolle Vergleichswerte zu ermitteln. Zum anderen befinden sich die Produkte in sehr unterschiedlichen Phasen, sodass ein solcher Vergleich wenig brauchbare Aussagen geliefert hätte.

Bei der Analyse der Messergebnisse muss zunächst betrachtet werden, wie vollständig bestimmte Informationen zu Änderungsanträgen überhaupt erfasst werden, beispielsweise die Dokumentation bestimmter Bearbeitungszustände oder die Schätzung von Aufwänden. Hier konnte in verschiedenen Bereichen eine Steigerung festgestellt werden, etwa bei der Dokumentation des geplanten Bearbeitungsbeginns, des zugeordneten Meilensteins, sowie bei der Angabe von *Release Notes*.

Wurden ausreichend vollständige Daten erfasst, kann für die einzelnen Produkte jeweils die zeitliche Veränderung in Bezug auf bestimmte Fragestellungen betrachtet werden, beispielsweise bei Aufwandseinhaltung, Termineinhaltung, Lösungsgeschwindigkeit oder der Häufigkeit von Nacharbeiten.

Weiterhin werden in der Arbeit von Richlofsky jeweils Darstellungsformen für vertiefende Analysen zu den betrachteten Qualitätsmerkmalen aufgezeigt [Ric09]. So kann die Verwendung unterschiedlicher Planungsmethoden mit Hilfe eines Streudiagramms gegenübergestellt werden (vgl. Abb. 8.3). Dabei wird auf der x-Achse der Anteil der Änderungsanträge aufgetragen, bei denen ein vorgesehener Fertigstellungstermin dokumentiert wurde, und auf der y-Achse der Anteil der Änderungsanträge, bei denen der geplante Bearbeitungsbeginn dokumentiert wurde. Für jede der betrachteten Zeitperioden und jedes betrachtete Produkt ergibt sich dann ein Datenpunkt. Diese Darstellung zeigt produktspezifisches Vorgehen bei der Planung auf.

Die Übergänge zwischen den einzelnen Bearbeitungszuständen der Änderungsanträge können in einer sogenannten Flusskarte visualisiert werden (vgl. Abb. 8.4). Eine Flusskarte zeigt die einzelnen Bearbeitungszustände als Knoten eines gerichteten Graphs. Die Dicke der Kanten im Graph entspricht der Anzahl der Änderungsanträge, bei denen ein bestimmter Zustandsübergang auftrat. Normalisiert wird dabei über die Anzahl aller Zustandsübergänge in dem betrachteten Zeitraum.

Mit diesen und ähnliche Visualisierungsformen kann das unterschiedliche Vorgehen bei den Produkten verdeutlicht, sowie entsprechende Veränderungen über die Zeit aufgezeigt werden. Auf diesem Wege können Schwächen und Verbesserungsmöglichkeiten im Prozess erkannt werden.

In Abschnitt 8.5 werden weitere gesammelte Erfahrungen aus den Fallstudien, sowie Gefahren für die Validität der Ergebnisse diskutiert.

8.2. Bewertung der Werkzeugunterstützung

Dieser Abschnitt diskutiert die innere und äußere Qualität der QMetric-Werkzeuge. Dabei orientieren wir uns an den Begriffen aus dem Standard ISO/IEC 9126 [ISO01]. Es werden jeweils diejenigen Komponenten der Werkzeug-Suite betrachtet, für die ein bestimmtes Qualitätsmerkmal relevant ist.

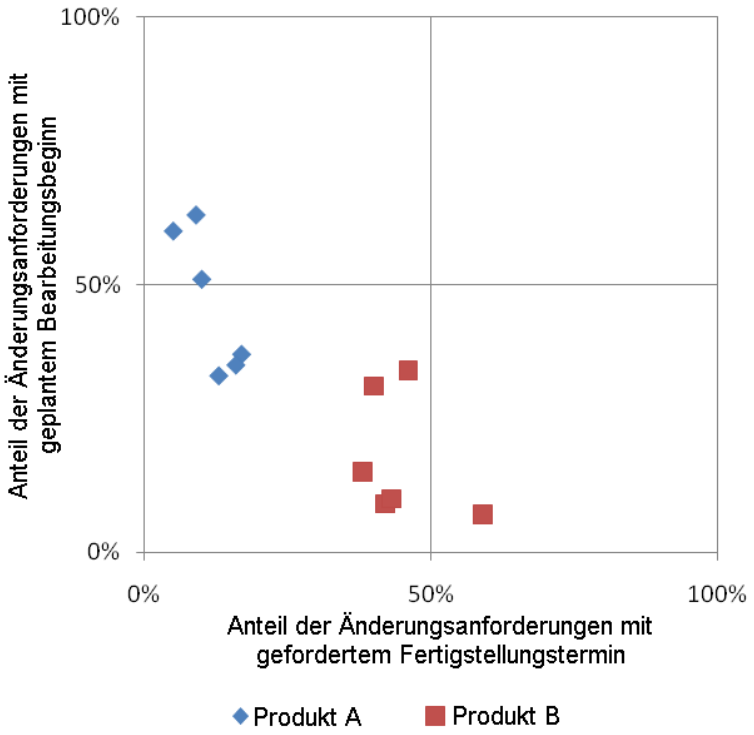


Abbildung 8.3.: Streudiagramm zur Visualisierung unterschiedlicher Planungsansätze [Ric09]

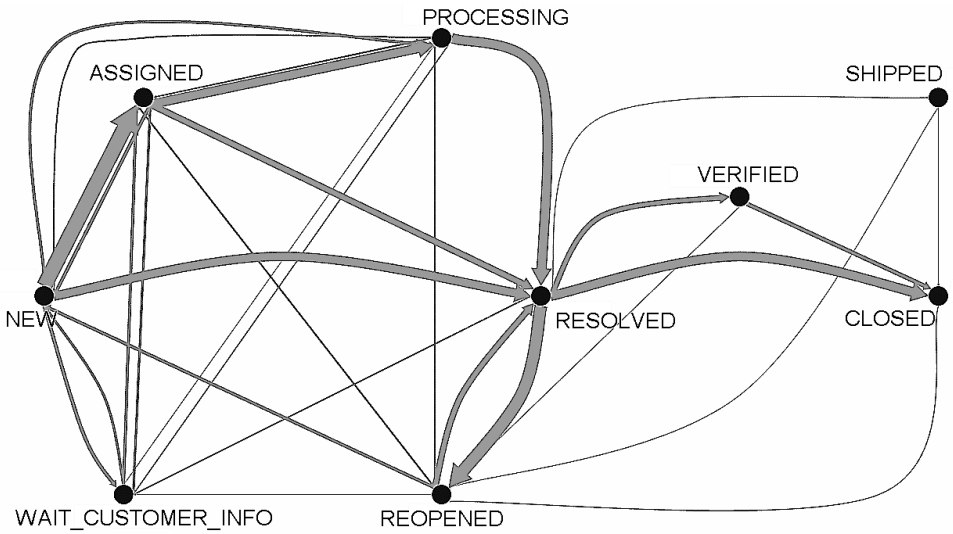


Abbildung 8.4.: Beispiel einer Flusskarte [Ric09]

8.2.1. Funktionalität

Eignung: Dieses Qualitätsmerkmal bezieht sich auf die Eignung von Funktionen für spezifizierte Aufgaben. Mit dem jetzigen Stand der Werkzeug-Suite können die Anforderungen im Umfeld der KISTERS AG erfüllt werden. Die Akzeptanz des Open Source Projekts QMetric untermauert auch die Eignung in anderen Organisationen.

In den Fallstudien konnte die Skalierbarkeit der Werkzeuge, also insbesondere der Berechnungskomponente gezeigt werden. So fanden die Auswertungen für das Projekt GNOME auf einem Issue-Tracking-System mit ca. 500.000 Änderungsanträgen statt.

Die Web-Anwendung ermöglicht einen unternehmensweiten Zugriff auf definierte Metriken. Die abgespeicherten Metriken werden vor der Speicherung auf Syntaxfehler geprüft und automatisch formatiert. Damit ist sichergestellt, dass die gespeicherten Metriken ausführbar sind. Die Web-Anwendung unterstützt keine versionierte Speicherung, was als zusätzliche Unterstützung zur Team-Zusammenarbeit nützlich wäre.

Die Werkzeuge zur Erstellung von Qualitätsmodellen und deren automatische Auswertung bieten eine wesentliche Erleichterung bei der Durchführung einer metrik-basierten Prozessbewertung. Damit werden die bei der Auswertung anfallenden Datenmengen besser handhabbar [Sch09]. Wiederum können beim Erstellen der Modelle die Randbedingungen möglichst früh geprüft werden. Zudem wird im Rahmen der Eclipse-Umgebung die versionierte Speicherung der Modelle unterstützt.

Genauigkeit: Die Genauigkeit der Messergebnisse wird im Wesentlichen durch die Qualität der Daten in den zu Grunde liegenden Software-Entwicklungsarchiven beeinflusst. An dieser Stelle sei auf die Erläuterungen in Abschnitt 8.5 verwiesen. Bei der Anbindung von Konfigurationsmanagement-Systemen wird eine Heuristik zur Rekonstruktion von zusammenhängenden Transaktionen eingesetzt. Die vom Benutzer zu wählenden Parameter für diese Heuristik können ebenfalls die Genauigkeit der Ergebnisse beeinflussen [Lis08].

Interoperabilität: Wie in Abschnitt 7.2.2 dargelegt, wurde die Werkzeug-Suite explizit für die Anwendbarkeit auf unterschiedlichen Software-Entwicklungsarchiven hin entworfen.

Die Anbindung eines neuen Issue-Tracking-Systems erfordert die Implementierung der entsprechenden Datenquellen, sowie gegebenenfalls weiterer Gewichte und Filter. Beim Issue-Tracking-System *Mantis* erforderte dies beispielsweise den Aufwand von einer Arbeitswoche. Die Komponenten von QMetric können dann durch Einstellungen in der Konfigurationsdatei an unterschiedliche Datenquellen angebunden werden.

Sicherheit: Auf die ausgewerteten Entwicklungsarchive wird grundsätzlich nur lesend zugegriffen. Somit sind diese vor vorsätzlichen oder fälschlichen Veränderungen geschützt. Die Web-Anwendung zur Abfrage von Metriken ermöglicht eine Zugriffskontrolle auf Basis der im Issue-Tracking-System angemeldeten Benutzer. Bei der Berechnung von Metriken werden Einschränkungen bei der Sichtbarkeit von Änderungsanträgen für bestimmte Benutzer oder Benutzergruppen allerdings nicht berücksichtigt.

8.2.2. Zuverlässigkeit

Reife: Die Metrikberechnungskomponente und die Web-Anwendung zur Abfrage von Metriken sind seit 2007 bei unserem Kooperationspartner KISTERS AG im produktiven Einsatz. Im Laufe der Zeit konnten dadurch auch sporadisch auftretende Probleme beim Mehrbenutzerbetrieb erkannt und behoben werden. Über die Bereitstellung als Open Source Produkt konnten weitere Anwender gewonnen werden, was unter anderem dazu beitrug, die Anforderungen beim Einsatz in anderen Systemumgebungen zu klären und besser zu erfüllen.

Die Qualitätsmodell-Werkzeuge sind die „jüngsten“ Komponenten der Werkzeug-Suite, daher besteht hier im Vergleich zu den vorgenannten Komponenten weniger Erfahrungen und intensive praktische Anwendung. Erfahrungen aus den ersten durchgeführten Fallstudien sind in die Entwicklung dieser Werkzeuge eingeflossen. Bei späteren Fallstudien wurde das Werkzeug dann praktisch eingesetzt [SL09, SSL09]. Eine Reihe von möglichen Verbesserungen wurde dabei erkannt und zum Teil schon umgesetzt.

Fehlertoleranz: Hier betrachten wir zunächst die Fehlertoleranz gegenüber Fehleingaben des Benutzers. Syntaxfehler in Metrik-Spezifikationen werden in der Berechnungskomponente abgefangen. In der Web-Anwendung wird dem Benutzer eine entsprechende Fehlermeldung angezeigt. Die Anwendung wird nicht beeinträchtigt.

Bei den Qualitätsmodell-Werkzeugen werden fehlerhafte Eingaben nach Möglichkeit direkt in den Assistentendialogen abgefangen. Weiterhin ermöglicht der Qualitätsmodell-Editor eine Validierung der erstellten Modelle gegen das gegebene Metamodell. Kann bei Durchführung einer Auswertung eine Metrik nicht ermittelt werden, so wird mit der Auswertung von den übrigen Metriken fortgefahren.

Eine weitere Fehlerquelle sind Inkonsistenzen in den auszuwertenden Software-Entwicklungsarchiven. Diese Erfahrung machten wir in einer der in Abschnitt 8.1 vorgestellten Fallstudien. Nach Wartungen des Issue-Tracking-Systems waren in der Datenbank Einträge mit ungültigen Attributwerten verblieben. Der Adapter für die Anbindung der Datenquelle wurde daraufhin so angepasst, dass solche Einträge bei der Metrikberechnung ignoriert werden, und eine Warnung in eine Log-Datei ausgegeben wird.

Wiederherstellbarkeit: Da im Auswertungswerkzeug, abgesehen von den benutzerdefinierten Anfragen, keine Daten persistent gespeichert werden, droht hier kein Datenverlust. Eine Störung kann prinzipiell durch einen Neustart der Anwendung innerhalb des Anwendungsservers behoben werden.

Bei den Qualitätsmodell-Werkzeugen könnte prinzipiell die Gefahr bestehen, dass die gespeicherten Modelle im Folge einer Fehlfunktion inkonsistent werden. Ein solches Problem konnte bisher allerdings nicht beobachtet werden. Die Manipulation und Speicherung der Modelle erfolgt über das ausgereifte EMF-Framework [BSM+03].

8.2.3. Verwendbarkeit

Anmerkungen zu Verständlichkeit und Erlernbarkeit der zu Grunde liegenden Konzepte finden sich in den Abschnitten 8.3.1 und 8.4.1. An dieser Stelle möchten wir uns auf die Komponenten der Werkzeug-Suite konzentrieren, die eine Benutzerschnittstelle anbieten, also die Web-Anwendung und die Qualitätsmodell-Werkzeuge.

Verständlichkeit: Die Abfrage von gespeicherten und vordefinierten Metrik-Abfragen orientiert sich an ähnlichen Konzepten für Suchanfragen in Issue-Tracking-Systemen, die typischen Anwendern also schon bekannt sind. Die Spezifikation von Details einer Metrik erfordert eine intensivere Beschäftigung mit dem Thema, wobei die Benutzerdokumentation zu Rate gezogen werden kann [Bug09b]. Die Verwendung der Web-Anwendung ist dort aufgaben-orientiert beschrieben. Dies erleichtert es dem Benutzer, für sein konkretes Ziel passende Hilfe zu finden.

Die grafische Darstellung von Qualitätsmodellen im Qualitätsmodell-Editor orientiert sich am bekannten Konzept des Qualitätenbaums [BBL76]. Die Auswertungsergebnisse können analog dazu in einer Baumansicht betrachtet werden, und deren zeitliche Entwicklung kann grafisch angezeigt werden. Die Benutzerdokumentation bietet einen schnellen Einstieg in die Grundkonzepte und typischen Arbeitsschritte [QMT09].

Erlernbarkeit und Bedienbarkeit: Bei der Entwicklung der Web-Anwendung zur Abfrage von Metriken wurde besonders auf die Lernförderlichkeit und die Selbstbeschreibungsfähigkeit Wert gelegt. Die Möglichkeit zwischen grafischen Dialogen und der Ansicht der zu Grunde liegenden Metrik-Spezifikation zu wechseln, erleichtert es, die Konzepte der Sprache ITMS nach und nach zu erlernen. Durch das direkte Editieren der ITMS-Metriken kann die volle Mächtigkeit der Sprache genutzt werden. Jede Kategorieseite enthält eine prägnante Beschreibung ihres Einsatzzwecks. Die Selbstbeschreibungsfähigkeit könnte allerdings durch den Einsatz von *Tooltips* noch weiter verbessert werden.

Die Qualitätsmodell-Werkzeuge nutzen die in der Eclipse-Umgebung bewährten Bedienkonzepte und verhalten sich somit erwartungskonform. In den einzelnen Dialogen der Assistenten wird entsprechend den Eclipse-Konventionen jeder Schritt kurz beschrieben [EHLP07]. Insgesamt ist die Selbstbeschreibungfähigkeit bei den Qualitätsmodell-Werkzeugen allerdings weniger stark ausgeprägt. Für unerfahrene Benutzer kann die Eclipse-Umgebung überfrachtet wirken [Jan09]. Die Bedienschritte sind komplexer und diese Werkzeuge sind im Vergleich zu der Web-Anwendung für fortgeschrittene Anwender vorgesehen.

Attraktivität: Die Resonanz auf das Open Source Produkt lässt vermuten, dass die Web-Anwendung auch attraktiv für die Anwender ist. Zudem kann die Gestaltung von einigen grafischen Dialogen über die Konfigurationsdatei an die Bedürfnisse einer Organisation angepasst werden. Um genauere Angaben zur Attraktivität zu treffen, müsste eine systematische Befragung der Nutzer durchgeführt werden.

Für die Qualitätsmodell-Werkzeuge liegen noch wenige Erfahrungswerte vor. Der Kreis von potentiellen Anwendern ist deutlich kleiner, da der Einsatz dieser Werkzeuge erst Sinn macht, wenn bereits ein gutes Verständnis über metrikbasierte Qualitätsbewertung erreicht ist.

8.2.4. Effizienz

Zeitverhalten: Bei der Betrachtung des Zeitverhaltens steht die Laufzeit des Algorithmus zur Metrikberechnung im Vordergrund. Zunächst möchten wir hierzu eine grobe Laufzeitabschätzung geben. Dabei ist zu berücksichtigen, dass es sehr viele Einflussfaktoren auf die Laufzeit gibt, wie die Struktur der ausgewerteten Software-Entwicklungsarchive sowie die in einer konkreten Metrik verwendeten Filter und Bewertungsverfahren.

Laufzeitmessungen haben gezeigt, dass der Hauptanteil der Laufzeit in Schritt 3 des Berechnungsalgorithmus verwendet wird (vgl. Abschnitt 7.2.2.2), also bei der Ermittlung der Bewertungszahlen. Im vorhergehenden Schritt 2 werden dazu schon die relevanten Änderungsanträge und Ereignisse aus der Datenbank geladen. Sei nun n die Anzahl der betrachteten Änderungsanträge. Diese Menge ist durch den Basisfilter und die in der Metrik verwendeten Ereignisfilter gegeben. Weiterhin sei m die Anzahl der relevanten Ereignisse und b die Anzahl der in der Metrik spezifizierten Bewertungsverfahren.

Der Algorithmus iteriert über die m relevanten Ereignisse. Je nach der Art des Ereignisses sind ein oder mehrere Änderungsanträge von dem Ereignis betroffen. Im schlechtesten Fall muss über alle n betrachteten Änderungsanträge iteriert werden. Für einen einzelnen Änderungsantrag muss dann gegebenenfalls ein Bewertungsverfahren angewandt werden, im schlechtesten Falle also insgesamt b verschiedene Bewertungsverfahren. Die Bestimmung der Bewertungszahl selbst kann dann mit $\mathcal{O}(1)$ abgeschätzt werden. Insgesamt ergibt sich also eine Laufzeit in $\mathcal{O}(m \cdot n \cdot b)$.

Zur Einschätzung der praktischen Verwendbarkeit kann das öffentliche Demo-System betrachtet werden[1], bei dem die Issue-Tracking-Daten des Apache-Projekts ausgewertet werden (ca. 30.000 Änderungsanträge; Hardware: Pentium 4 mit 3,2 GHz). Metriken können in aller Regel in weniger als 10 Sekunden berechnet werden. Metrikberechnungen für einzelne Produkte können interaktiv durchgeführt werden.

Ressourcenverbrauch: Im Hinblick auf den Ressourcenverbrauch muss beim Auswertungsalgorithmus der benötigte Hauptspeicher betrachtet werden. Alle für eine einzelne Metrikberechnung benötigten Daten werden gleichzeitig in den Speicher geladen. Als Erfahrungswert gilt hier, dass bei großen Berechnungen (>100.000 betrachtete Änderungsanträge) der Speicherverbrauch über 1 GB steigt. Sobald dann Speicherseiten ausgelagert werden müssen, führt dies zu einer verschlechterten Laufzeit.

8.2.5. Wartbarkeit

8.2.5.1. Analysierbarkeit und Änderbarkeit

Mit Hilfe des Werkzeugs *SonarJ* wurde sichergestellt, dass die Abhängigkeiten zwischen Paketen konform zu der in Abschnitt 7.2 dargestellten Architektur sind [Son]. Weiterhin wurden mit dem Werkzeug eine Reihe von Code-Metriken erhoben (siehe Tabelle 8.2). Dabei wird nur der Java-Code betrachtet. Modelle zur Code-Generierung sind nicht berücksichtigt. Die aufgelisteten Entwurfseinheiten sind in Kapitel 7.2 erläutert. Die Anzahl der Codezeilen ohne Leerzeilen und Kommentare (LOC) und die Anzahl der Typen, also Klassen und Interfaces, gibt zunächst einen groben Eindruck über die Größenordnung.

Komplexität: Tabelle 8.2 enthält weiterhin Werte für die zyklomatische Komplexität (CC) [McC76]. Es sind jeweils die Maximal- und die Durchschnittswerte für Typen und für Methoden angegeben. Die höchsten Werte finden sich hier bei dem generierten Code, was sich also nicht negativ auf die Änderbarkeit auswirkt. Vielmehr unterstützt der modellgetriebene Ansatz die Änderbarkeit beim Qualitätsmodell-Editor und dem Auswertungswerkzeug.

Der Maximalwert der zyklomatischen Komplexität pro Typ von 110 findet sich beim Paket *frontend.client*. Bei der entsprechenden Klasse ist eine Restrukturierung angebracht. Der Wert von 106 bei der Komponente *scm* betrifft eine Klasse, die den Charakter einer Funktionsbibliothek hat. Diese Klasse enthält sehr viele, recht einfache Methoden mit ähnlichen Aufgaben. Eine Restrukturierung wäre hier nicht sinnvoll. Eine zyklomatische Komplexität von weniger als 100 pro Klasse gilt als unproblematisch [RSG99]. Für die Komplexität einer Methode kann ein Wert kleiner als 10 als gut und ein Wert kleiner als 20 als

[1] http://demo.bugzillametrics.org

Tabelle 8.2.: Codemetriken für QMetric (Stand 01.09.2009)

Package (org.qmetric.)	LOC	CC-accumulated per type		CC-average per method		Number of types	Relational Cohesion
		average	max	average	max		
core	6.022	5,0	56	1,37	9	229	3,52
core.test	994	5,6	19	1,29	6	25	1,40
bugzilla	3.473	5,1	36	1,36	8	83	2,43
mantis	1.203	5,8	20	1,33	5	28	1,96
scm.coreExtensions	2.669	5,2	41	1,49	22	86	2,28
scm.import	4.857	13,7	51	1,61	15	67	3,88
scm.test	1.680	3,5	22	1,31	4	42	1,48
scm (gemeinsam genutzte Klassen)	2.537	14,0	106	1,07	4	28	1,00
chart	718	15,8	44	1,80	6	5	1,00
frontend.client	5.855	6,5	110	1,68	26	177	3,00
frontend.server	2.170	9,9	27	1,54	8	30	2,03
qtools.model (generiert)	16.641	13,1	149	1,91	111	275	3,23
qtools.wizard & .metricsadapter	2.363	6,4	44	1,69	11	53	1,60
qtools.evaluationmodel (generiert)	3.979	28,2	102	2,04	70	61	5,21
qtools.evaluationtool.frontend	2.732	5,3	27	1,50	8	63	1,54
qtools.evaluationtool (übrige Klassen)	4.569	5,5	38	1,52	15	79	2,65
Gesamt	62.462	8,8	149	1,77	111	1.331	n.a.
Gesamt (ohne generierten Code)	41.842	6,4	110	1,52	26	995	n.a.

akzeptabel bewertet werden [FGR+97, NAS]. Diese Grenzwerte werden auch von den Methoden mit der höchsten Komplexität selten überschritten.

Kohäsion: Die letzte Spalte in Tabelle 8.2 enthält Werte für die relationale Kohäsion [Mar06]. Die relationale Kohäsion ist definiert als die Anzahl innerer Typbeziehungen in einer Entwurfseinheit, geteilt durch die Anzahl Typen in einer Entwurfseinheit. Dieser Wert soll den inneren Zusammenhalt einer Entwurfseinheit charakterisieren. Nach Martin können Werte zwischen 1,5 und 4,0 als gut bezeichnet werden [Mar06]. Fast alle hier betrachteten Entwurfseinheiten liegen in diesem Bereich. Für *scm* und *chart* beträgt die relationale Kohäsion 1. Dies ist allerdings unproblematisch, da diese Entwurfseinheiten den Charakter einer Klassenbibliothek haben.

Kopplung Wie in Abschnitt 7.2 erkennbar wurde, bestehen nur schmale Schnittstellen zwischen den einzelnen Komponenten der Werkzeug-Suite. Die ursprünglich im Kontext von Bugzilla entwickelte Berechnungskomponente ist inzwischen unabhängig vom auszuwertenden Software-Entwicklungsarchiv. Bei der Web-Anwendung gibt das Google Web Toolkit einen sinnvollen Rahmen für die Kommunikation zwischen Client und Server vor. Die Qualitätsmodell-Werkzeuge werden über Web-Services an die übrigen Komponenten angebunden.

Abstraktheit und Instabilität: Die Betrachtung dieser beiden Eigenschaften kann Hinweise darauf geben, ob die Entwurfseinheiten richtig geschnitten sind. Die Abstrakheit wird berechnet als die Anzahl der abstrakten Typen geteilt durch die Anzahl aller Typen in einer Entwurfseinheit [Mar94]. Instabilität

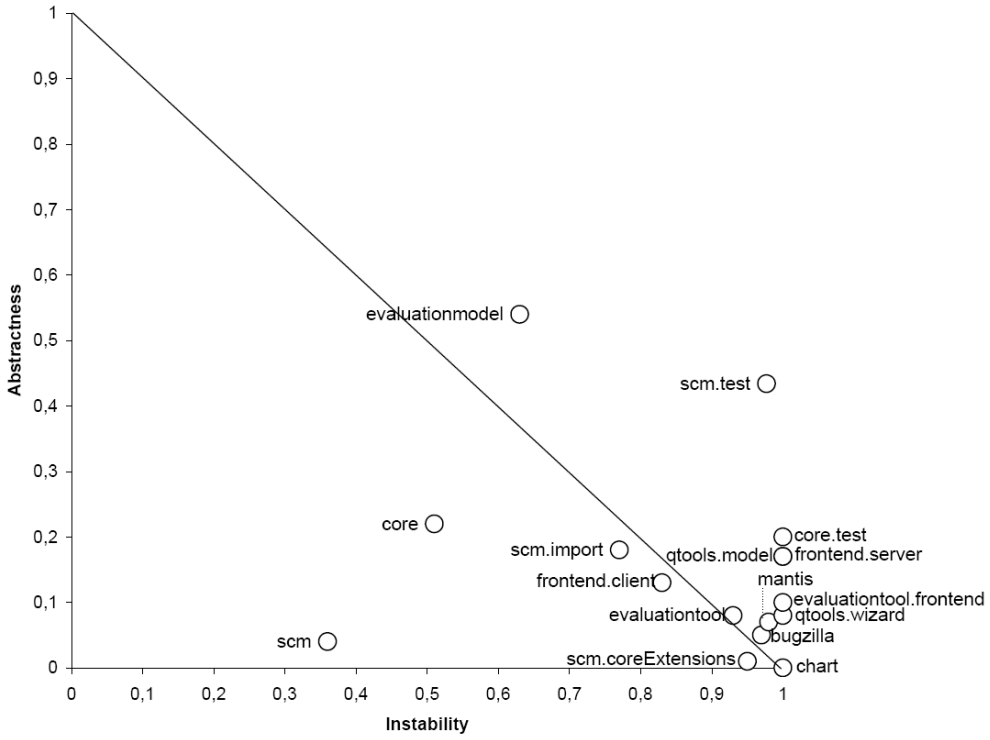


Abbildung 8.5.: Distanzgraph nach Martin [Mar94]

ist die Anzahl der ausgehenden Abhängigkeiten einer Entwurfseinheit geteilt durch alle Abhängigkeiten der Entwurfseinheit. Eine Entwurfseinheit, die keine ausgehenden Abhängigkeiten besitzt, gilt als vollkommen stabil, da sie nicht von anderen Entwurfseinheiten abhängig ist. Nach Martin soll Abstraktheit und Instabilität bei einer Entwurfseinheit in einem ausgeglichenen Verhältnis stehen [Mar94]. Bei Entwurfseinheiten, die von vielen anderen Entwurfseinheiten genutzt werden, ist also eine höhere Abstraktheit erforderlich.

In dem Diagramm in Abbildung 8.5 sind die einzelnen Entwurfseinheiten entsprechend ihrer Werte für Abstraktheit und Instabilität aufgetragen. Die Balance zwischen diesen beiden Werten gilt als um so besser, desto geringer die Distanz zur eingezeichneten Diagonalen ist. Die meisten Entwurfseinheiten liegen weiter rechts, da auch Abhängigkeiten zu einer Vielzahl von Basisbibliotheken bei der Berechnung der Instabilität berücksichtigt werden. Wie in der Abbildung erkennbar ist, liegen die Entwurfseinheiten *scm* und *scm.test* am weitesten von der Diagonalen entfernt. Die Werte der übrigen Entwurfseinheiten sind akzeptabel.

Die meisten Klassen aus *scm* dienen der Anbindung der Datenbank zur Speicherung von Daten aus Konfigurationsmanagement-Systemen. Nach dem Diagramm zu urteilen, ist die Abstraktheit zu gering, sodass die Entwurfseinheit möglicherweise schwer wartbar ist. In diesem Falle ist es allerdings tolerierbar, da die Methoden in den einzelnen Klassen recht einfach sind (vgl. Tabelle 8.2). Bei *scm.test* gibt es für eine Reihe von Aktivitäten auf Konfigurationsmanagement-

Systemen jeweils eine abstrakte Klasse, die dann für die Systeme CVS und Subversion unterschiedlich ausgeprägt wird. Daher liegt hier eine hohe Abstraktheit vor, ohne dass dies die Nutzbarkeit einschränkt.

8.2.5.2. Testbarkeit und Stabilität

Für die Metrikberechnungskomponente und den Import von Daten aus Konfigurationsmanagement-Systemen wurden umfangreiche automatische Testumgebungen aufgebaut (siehe auch Abschnitt 7.3). Auch für die Anbindung von Metrikwerkzeugen und die Auswertungen bei den Qualitätsmodell-Werkzeugen wurden automatische Tests erstellt. Die inkrementell gewachsenen Test-Suiten erhöhen die Stabilität gegen unbeabsichtigte Auswirkungen von Änderungen an der Software. Die Web-Anwendung muss manuell getestet werden.

Weiterhin können durch die an den Schnittstellen durchgeführten Überprüfungen von Eingabedaten (vgl. Abschnitt 8.2.2) unbeabsichtigte Auswirkungen von Änderungen erkannt werden.

8.2.6. Portabilität

Anpassbarkeit: Alle Komponenten der Werkzeug-Suite können betriebssystemunabhängig eingesetzt werden. Durch die Bereitstellung als Open Source Produkt konnte weiterhin die Einsatzmöglichkeit in unterschiedlichen Systemumgebungen bestätigt und weiter verbessert werden.

Installierbarkeit: Für den Endanwender sind Metrikabfragen über die Web-Anwendung ohne weitere Installation zugänglich. Die Installation des Servers erfordert die Erzeugung der Datenbanktabellen, das Entpacken der Web-Anwendung in den Anwendungsserver, sowie das Anpassen von Einstellungen in einer Konfigurationsdatei. Nach Erfahrungen aus dem Open Source Projekt hat es sich bewährt, dass bei Aufruf der Web-Anwendung eine automatische Überprüfung der Installation durchgeführt wird. Dabei werden alle bisher bekannten Fehlerquellen überprüft (z.B. Zugriff auf die Konfigurationsdateien, Konfiguration der Datenbank, Datenbank-Treiber, Zugriffseinschränkungen im Anwendungsserver) und im Fehlerfall eine aussagekräftige Fehlermeldung ausgegeben.

Die Qualitätsmodell-Werkzeuge können über den in Eclipse bewährten *Update-Site* Mechanismus installiert werden.

Koexistenz: Der Zugriff auf Software-Entwicklungsarchive erfolgt nur lesend, sodass dies nicht zu Konflikten mit sonstiger Software führen kann. Ist das Issue-Tracking-System und die Metrikberechnungskomponente auf dem selben Rechner installiert, beeinträchtigt eine intensive Nutzung von Metrikberechnungen die Antwortgeschwindigkeit des Issue-Tracking-Systems.

In Bezug auf die Qualitätsmodell-Werkzeuge bietet die Eclipse-Umgebung geeignete Rahmenbedingungen für die Koexistenz mit sonstigen Plug-Ins.

Ersetzbarkeit: Wie in Abschnitt 7.2 dargestellt, ist die Werkzeug-Suite modular aufgebaut. Es bestehen nur schmale Schnittstellen zwischen den Haupt-Komponenten. Auf Fremdsysteme wird über Standard-Schnittstellen zugegriffen. Diese Voraussetzungen erleichtern einen möglichen späteren Austausch von Komponenten in der Werkzeug-Suite.

8.3. Bewertung der Sprache ITMS

In diesem Abschnitt werden Vor- und Nachteile der deklarativen Sprache ITMS zur Spezifikation von Metriken diskutiert. Die Diskussion ist strukturiert nach unterschiedlichen Erfolgsfaktoren für domänenspezifische Sprachen, die von Hermans et al. auf Basis einer Literaturrecherche ermittelt wurden [HPvD09].

8.3.1. Erlernbarkeit

In Abschnitt 8.2 wurde dargestellt, in welcher Form die grafische Benutzeroberfläche die Erlernbarkeit von ITMS fördert. An dieser Stelle soll die eigentliche Sprache betrachtet werden. In Abschnitt 4.5 wurde das konzeptuelle Modell der Sprache erläutert. Die Erlernbarkeit hängt davon ab, wie gut es dem Anwender möglich ist, sich ein mentales Modell über die Sprache zu bilden. Mentale Modelle entwickeln sich evolutionär und sind für gewöhnlich weder vollständig noch technisch exakt. Grundsätzlich sind solche mentalen Modelle durch den technischen Hintergrund und die Erfahrung des Anwenders eingeschränkt [Nor83].

Um die Erlernbarkeit systematisch zu bewerten, werden wir im Folgenden die von Green und Petre vorgeschlagenen kognitiven Dimensionen verwenden [GP96]. Diese **kognitiven Dimensionen** beschreiben allgemeine Designprinzipien für textliche oder visuelle Notationen. Zum besseren Verständnis ist im Folgenden jede Dimension durch eine kurze Frage charakterisiert.

Nähe der Abbildung: *Wie nah ist die gegebene Notation an der Problemwelt?* Stellt man eine ITMS-Spezifikation einer strukturierten textlichen Beschreibung der Metrik gegenüber, gibt es eine Reihe von Elementen die sich direkt entsprechen, wie der Basisfilter oder die mathematische Beschreibung der Gruppenwertberechnung. Durch die bestehenden vier Kategorien von Bewertungsverfahren ist eine gewisse Systematik für die Bestimmung von Metriken auf einzelnen Änderungsanträgen vorgegeben. Hier kann die textlichen Beschreibung unter Umständen einen anderen „Rechenweg“ vorgeben. In diesem Fall ist es für den Anwender schwieriger die Konzepte aufeinander abzubilden.

Neben den einzelnen Elementen der Sprache muss auch betrachtet werden, wie gut es dem Anwender möglich ist, diese zu größeren Einheiten zusammenzusetzen. Diese Komposition von Elementen ist in ITMS relativ starr vorgegeben, sodass dies wenig Schwierigkeiten bereitet.

Viskosität: *Wie schwierig ist es, eine Änderung durchzuführen?* Generell ist diese Eigenschaft bei textlichen Notationen besser als bei visuellen Notationen [PMdCH08]. Typische konzeptionelle Änderungen an einer Metrik, wie beispielsweise Filter oder mathematische Operationen, sind in der ITMS-Spezifikation an einer Stelle lokalisiert, sodass eine Änderung leicht durchgeführt werden kann.

Versteckte Abhängigkeiten: *Wie gut sind Abhängigkeiten zwischen den Elementen in der Notation sichtbar?* Generell gibt es wenige Abhängigkeiten zwischen den Notationselementen von ITMS. Eingesetzte Bewertungsverfahren werden bei den Gruppenwertberechnungen durch einen textlichen Bezeichner referenziert, sodass die Abhängigkeit erkennbar ist. Änderungen am Basisfilter oder an fixierten Attributen können sich auf das Verhalten der Bewertungsverfahren auswirken. Dies ist dem Anwender möglicherweise nicht direkt bewusst.

Schwierige mentale Operationen: *Welche gedanklich schwierigen Aufgaben ergeben sich auf der Ebene der Notation?* Ein Hinweis dafür kann sein, an welchen Stellen ein Anwender andere Hilfsmittel wie etwa schriftliche Notizen zu Hilfe nehmen muss. Nach unserer Erfahrung betrifft dies die beiden Bewertungsverfahren „Intervalllänge“ und „Verweildauer“. Hier muss bei der Definition einer Metrik oftmals das Bewertungsverfahren gedanklich nachvollzogen werden. Diese Aufgabe liegt allerdings teilweise auch auf der Ebene der Semantik der Metrik.

Frühzeitige Festlegung: *Ist der Anwender gezwungen, Entscheidungen zu treffen, bevor er über die eigentlichen Informationen verfügt?* Bei der Entwicklung einer Metrik ist in einigen Fällen zu Beginn nicht klar, welches Bewertungsverfahren am besten geeignet ist. Hier kann es also vorkommen, dass die erste Festlegung sich als nicht geeignet herausstellt. In diesen Fällen lassen sich definierte Filter allerdings leicht im Rahmen eines anderen Bewertungsverfahrens verwenden.

Sekundäre Notation: *Können in der Sprache zusätzliche Informationen über die offizielle Syntax hinaus annotiert werden?* Da ITMS in die Sprache XML eingebettet ist, können die gewohnten Kommentare verwendet werden.

Knappheit: *Wie viele Symbole bzw. wie viel Raum wird in Anspruch genommen, um einen bestimmten Sinngehalt auszudrücken?* Eine Metrik-Spezifikation hat typischerweise eine Länge von um die 50 Zeilen. Die Knappheit wird etwas beeinträchtigt durch die Einbettung in die Sprache XML. Abgesehen davon kann eine Metrik kaum noch kürzer dargestellt werden, ohne inhaltliche Präzision zu verlieren.

Sichtbarkeit: *Wie gut sind die einzelnen Bereiche des Programms zugänglich?* Da eine Metrik in einem einzigen Dokument spezifiziert ist, und in den allermeisten Fällen recht kurz ist, kann die Sichtbarkeit als sehr gut angesehen werden.

Konsistenz: *Wie leicht ist es, auf übrige Elemente der Notation zu schließen, wenn bereits ein Teil der Elemente erlernt wurde?* Die Filter können an allen Stellen gleich verwendet werden. Die Komposition von Ereignis- und Zustandsfiltern mit booleschen Operatoren ist analog. Weiterhin sind die Elemente zur Spezifikation der Bewertungsverfahren jeweils ähnlich, sodass insgesamt eine hohe Konsistenz vorliegt.

Fehleranfälligkeit: *Wie leicht veranlasst die Notation zu Flüchtigkeitsfehlern?* Durch die enge Domäne und den vergleichsweise kleinen Wortschatz sind domänenspezifische Sprachen generell weniger anfällig für Flüchtigkeitsfehler [PMdCH08]. Nach unserer Erfahrung betreffen Flüchtigkeitsfehler bei ITMS-Spezifikationen in der Regel das Schließen von XML-*Tags*.

Schrittweise Auswertung: *Wie leicht ist es, ein unvollständiges Programm auszuführen, um Feedback zu erhalten?* Eine unvollständige ITMS-Spezifikation kann jederzeit gegen das XML-Schema validiert werden, um Hinweise auf fehlende Notationselemente zu erhalten. Eine ITMS-Spezifikation kann jederzeit ausgewertet werden, wenn ein Grundgerüst der Metrik gegeben ist, also mindestens ein Bewertungsverfahren spezifiziert wurde.

Erkennbarkeit von Rollen: *Wie gut ist erkennbar, welche Rolle ein einzelnes Element im Programm einnimmt?* Wegen der Kürze der Metrik-Beschreibungen und den kleinen Wortschatz der Sprache ist davon auszugehen, dass diese Rollenbeziehungen gut erkennbar sind.

Abstraktionsgrad: *Was ist das minimale und maximale Abstraktionsniveau der Sprache? Können Details von Notationselementen gekapselt werden?* Wegen der Kürze der ITMS-Spezifikationen hat sich ein einziges Abstraktionsniveau als ausreichend erweisen. Aus den Erfahrungen der industriellen Fallstudie erscheint es noch wünschenswert, die Spezifikation von benutzerdefinierten Berechnungszeiträumen als ein eigenes Element zu kapseln.

8.3.2. Benutzbarkeit

Nach Hermans et al. bezieht sich der Begriff Benutzbarkeit darauf, ob die Werkzeuge und Methoden zu einer domänenspezifischen Sprache einfach und zweckdienlich benutzt werden können [HPvD09]. In Bezug auf die Werkzeugunterstützung wurde dies bereits eingehend in Abschnitt 8.2 betrachtet. Das in Kapitel 5 beschriebene Vorgehen hat sich in den Fallstudien als gut verwendbar erwiesen.

Für die Benutzbarkeit der Werkzeugunterstützung von ITMS auf längere Sicht sind die folgenden Punkte wichtig. Völter zählt diese Punkte zu bewährten Praktiken beim Entwurf domänenspezifischer Sprachen und ihrer zugehörigen Werkzeugumgebungen [Völ08].

- **Anpassbarkeit der Sprache:** ITMS kann für die auszuwertenden Software-Entwicklungsarchive unterschiedlich ausgeprägt werden (siehe Abschnitt 7.2).
- **Evolution der Sprache:** Erfahrungen aus der Weiterentwicklung von ITMS haben gezeigt, dass die Sprache mit geringem Aufwand erweitert werden kann. Neben zusätzlichen Filtern und Gruppenwertberechnungen wurden auch die Bewertungsverfahren für Intervalllänge und Verweilzeit um zusätzliche Parameter ergänzt. Dabei wurde die Semantik für bestimmte Sonderfälle präzisiert [Ric09]. Die Semantik existierender ITMS-Spezifikationen wurde dadurch nicht beeinträchtigt.
- **Trennung von Zuständigkeiten:** In der Werkzeugumgebung ist die Spezifikation der Metrik von der Spezifikation der Diagrammdarstellung getrennt.

8.3.3. Ausdrucksstärke

Grundlage bei der Entwicklung von ITMS war eine initiale Liste von relevanten Metriken. Die Sprache wurde also ausgehend von vorhandenem Domänenwissen konstruiert und in der Folge erweitert. In den geschilderten Fallstudien (siehe Abschnitt 8.1) konnten fast alle angedachten Metriken beschrieben werden. Wenn auf eine angedachte Metrik verzichtet wurde, war typischerweise die Datengrundlage problematisch und nicht die Ausdrucksstärke von ITMS. Es gab wenige Sonderfälle die nicht beschrieben werden konnten [Ric09]:

- Für die beiden Bewertungsverfahren zum Zählen von Ereignissen ist es nicht möglich, ein Startereignis anzugeben, sodass erst nach Auftreten dieses Ereignisses im Lebenslauf eines Änderungsantrags die Zählung begonnen wird.
- Beim Issue-Tracking-System Bugzilla ändert sich bei einem Zustandswechsel nach *Resolved* gleichzeitig das Attribut *Resolution*, welches die Art der Lösung beschreibt. Diese simultane Änderung der Attribute erscheint in

ITMS allerdings als zwei unterschiedliche Ereignisse. Dadurch können Ereignisfilter mit bestimmten Bedingungen an die Belegung dieser beiden Attribute nicht beschrieben werden.

- Bei der Bestimmung von Verweildauern und Intervalllängen können arbeitsfreie Tage nicht unberücksichtigt bleiben.

Prinzipiell ist es möglich, ITMS bzw. die Berechnungskomponente um Unterstützung für diese Sonderfälle zu erweitern. Der Aufwand dafür liegt in der Größenordnung von einigen Tagen. Für die gewünschten Auswertungen in den Fallstudien (siehe Abschnitt 8.1) hatten diese Erweiterungen allerdings keine besondere Relevanz, sodass bisher auf eine Implementierung verzichtet wurde.

In Bezug auf die Auswertung von Konfigurationsmanagement-Systemen wurde die Anwendung mittels Auswertungen auf den Software-Entwicklungsarchiven des ViPER-Projekts evaluiert [Lis08]. Abfragen, die mit speziellen Auswertungssystemen für Konfigurationsmanagement-Systeme möglich sind [ED, DP03], können zum überwiegenden Teil auch mit ITMS formuliert werden. Grenzen zeigen sich, wenn Anfragen sich auf Eigenschaften einer einzelnen Datei beziehen. So kann mit ITMS nicht die größte Datei oder die Datei mit den meisten Revisionen ermittelt werden. Dies liegt daran, dass bei ITMS die Änderungsanträge die zentrale Rolle spielen.

Die in den übrigen Fallstudien ausgewerteten Archive enthalten nicht die Verknüpfung zwischen Änderungsanträgen und Transaktionen im Konfigurationsmanagement-System. Somit konnten hier in Bezug auf Konfigurationsmanagement-Systeme keine zusätzlichen praktischen Erfahrungen gesammelt werden.

Weitere Erkenntnisse in Bezug auf die Ausdrucksstärke von ITMS kann der Vergleich mit anderen Werkzeugen erbringen. Von den in Abschnitt 4.2 genannten Ansätzen kommt dazu nur der OLAP-Ansatz in Frage (siehe Abschnitt 4.2.3), da dieser im Bereich der Auswertung von Issue-Tracking-Systemen angewandt wird und ausreichend ausgereift ist. Für eine Einschätzung der Ausdrucksstärke bei diesem Ansatz betrachten wir das Werkzeug *Software Quality Reports for Jira/Bugzilla* [SQR].

Die Auswertungen bei diesem Werkzeug basieren auf einem OLAP-Cube, der bestimmte Kennzahlen über die Historie der Änderungsanträge erfasst, wie die Anzahl der offenen Änderungsanträge, die Anzahl der in einem Zeitraum gelösten Änderungsanträge und das Alter der offenen Änderungsanträge. Diese Daten werden nach den Dimensionen Typ des Änderungsantrags, Reporter, Bearbeiter, Priorität, Produkt, Status, Version, Zeitpunkt der Erzeugung und Zeitpunkt der Fertigstellung aufgegliedert.

Durch diese Dimensionen sind die Möglichkeiten der Filterung und Gruppierung vorab eingeschränkt. Die mit diesem Ansatz möglichen Auswertungen lassen sich im Kontext von ITMS in die Kategorie „Zählen von Ereignissen“ einordnen. Auswertungen, die den übrigen Kategorien von Bewertungsverfahren in ITMS entsprechen, werden nicht unterstützt. Dies liegt offenbar daran, dass diese sich nicht gut in Form von ETL-Transformationen beschreiben lassen.

Zusammenfassend lässt sich also feststellen, dass ITMS in Bezug auf die möglichen Metrikberechnungen mächtiger ist. Der OLAP-Ansatz hat seine Stärken, wenn bei einem Auswertungsergebnis explorativ die Gruppierungsparameter und die zeitliche Auflösung verändert werden soll. Bei ITMS müsste in einem solchen Fall die Metrik neu berechnet werden.

8.3.4. Wiederverwendbarkeit

Prinzipiell kann mit einer domänenspezifischen Sprache die Wiederverwendung auf Modellebene unterstützt werden. ITMS bietet allerdings keine konzeptionelle Unterstützung, um Elemente aus einer ITMS-Spezifikation in einer anderen ITMS-Spezifikation wiederzuverwenden. Nach den bisherigen praktischen Erfahrungen bestand dazu bisher noch kein Bedarf (vgl. Abschnitt 8.1). Wegen der Knappheit der ITMS-Spezifikationen reicht es erfahrungsgemäß aus, eine bestehende Metrik zu kopieren und als Vorlage zu verwenden. Abhängigkeiten zwischen ITMS-Spezifikationen würden sich zudem nachteilig auf die in Abschnitt 8.3.1 genannten Aspekte Knappheit, Sichtbarkeit und versteckte Abhängigkeiten auswirken.

8.3.5. Entwicklungskosten

An dieser Stelle soll diskutiert werden, welchen Einfluss die Verwendung von ITMS auf die Entwicklungskosten für eine angedachte Metrik hat. Offensichtlich ist, dass ITMS hier deutlich weniger Aufwand erfordert als eine allgemeine Programmiersprache. Die beiden umfangreichen Fallstudien [Sch09, Ric09] wären in dieser Form nicht möglich gewesen, hätten alle angedachten Metriken händisch implementiert werden müssen.

Interessanter ist die Frage, wie der Vergleich zu anderen spezialisierten Ansätzen zur Metrikauswertung ausfällt. Wir betrachten dazu wiederum den Vergleich zum OLAP-Ansatz (vgl. Abschnitt 8.3.3), der in dem Werkzeug *Software Quality Reports for Jira/Bugzilla* [SQR] gegeben ist. Bei diesem Ansatz teilt sich die Beschreibung der Metriken in zwei Teile auf. Zum einen wird die Anzahl der möglichen Metriken durch die Transformationsskripte des ETL-Prozesses vorgegeben. Zum anderen wird eine konkrete Metrik erst bei einer Anfrage an das OLAP-System vollständig spezifiziert. Die Definition der Transformationen für den ETL-Prozess ist relativ komplex und erfordert genaue Kenntnisse über die zu Grunde liegende Datenbank. Die benutzerdefinierte Anpassung von vordefinierten Abfragen an das OLAP-System ist dagegen recht einfach, und wird durch eine intuitive Benutzeroberfläche gut unterstützt.

8.3.6. Zuverlässigkeit

Nach Hermans et. al. bezieht sich Zuverlässigkeit auf den Grad in dem eine domänenspezifischen Sprache es ermöglicht, Fehler im Endprodukt zu ver-

meiden [HPvD09]. Aus den geschilderten Fallstudien liegen hier gute Erfahrungen vor. Wie in Kapitel 5 dargestellt, können bei der Entwicklung von ITMS-Spezifikationen sehr frühzeitig bereits Validierungsschritte durchgeführt werden. Die Details in den Metrikergebnissen erleichtern es, Fehler in Metrik-Spezifikationen zu identifizieren. Veränderungen an einer Metrik können schnell umgesetzt und die Metrik wiederum validiert werden.

8.4. Bewertung des Ansatzes zur Qualitätsmodellierung

In diesem Abschnitt wird der in Kapitel 6.3 eingeführte Ansatz zur Qualitätsmodellierung diskutiert. Die erzeugten Qualitätsmodelle und Auswertungsmodelle können ebenfalls als Modelle aus einer domänenspezifischen Sprache angesehen werden. Daher kann die Bewertung noch den gleichen Aspekten wie im vorangegangenen Abschnitt strukturiert werden. Wenn nötig wird dabei das Qualitätsmodell und das Auswertungsmodell getrennt diskutiert.

8.4.1. Erlernbarkeit

Zur Bewertung der Erlernbarkeit verwenden wir wiederum die in Abschnitt 8.3.1 vorgestellten kognitiven Dimensionen nach Green und Petre [GP96]. Da die betrachteten Metamodelle konzeptionell einfacher sind, als die Sprache ITMS kann die Diskussion kürzer gefasst werden.

Qualitätsmodell: Zunächst betrachten wir die Qualitätsmodelle. Da ihre Visualisierung dem bekannten Konzept des Qualitätenbaums nahe kommt, kann die **Nähe der Abbildung** und die **Erkennbarkeit von Rollen** der einzelnen Elemente als gut angesehen werden. Da ohnehin nur wenige Modellelemente verwendet werden, kann auch die **Konsistenz** als gut bezeichnet werden. Es sind auch keine **versteckten Abhängigkeiten** zwischen den Elementen gegeben. Auf der Ebene der Notation bestehen nach unserer Erfahrung keine **schwierigen mentalen Operationen**. Eine generelle Schwierigkeit für die Erlernbarkeit von Ansätzen für die Qualitätsbewertung ist die uneinheitliche Verwendung von Begriffen in der Literatur, sowie die Entsprechung von englischen zu deutschen Begriffen [Sch09]. Somit muss auch bei QMetric zunächst ein Verständnis über die Bedeutung und Abgrenzung der Begriffe erlangt werden.

Die **Viskosität** ist dadurch eingeschränkt, dass Veränderungen von Details der Berechnungsvorschriften über Assistentendialoge durchgeführt werden müssen. Die Berechnungsvorschriften können zudem nur in einem Eigenschaften-Dialog betrachtet werden, was die **Knappheit** und die **Sichtbarkeit** beeinträchtigt.

Diese beiden Dimensionen könnten möglicherweise mit einer rein textlichen Notation besser erfüllt werden. Eine **sekundäre Notation** ist durch Kommentarfelder als auch durch die Layout-Möglichkeiten in der grafischen Darstellung der Qualitätsmodelle gegeben.

Beeinträchtigungen durch eine **frühzeitige Festlegung** liegen nicht vor. Bei der Definition des Qualitätsmodells müssen zunächst natürlich bestimmte Annahmen für die Berechnungsvorschriften getroffen werden, die erst später validiert werden können. Dies liegt allerdings in der Domäne begründet und ist unabhängig von dem gegebenen Metamodell. Da die meisten Eingaben über Assistentendialoge erfolgen, ist die **Fehleranfälligkeit** gering.

Im Hinblick auf den **Abstraktionsgrad** ist die Ansicht als Graph das höchste Abstraktionsniveau. Jeder Qualitätsknoten kapselt bestimmte Information wie etwa die Berechnungsvorschrift. In den Fallstudien hat sich gezeigt, dass in einem Modell typischerweise Qualitätsknoten mit recht ähnlichen Berechnungsvorschriften verwendet werden (vgl. Abschnitt 8.1). Dies lässt es wünschenswert erscheinen, dazu entsprechende vordefinierte Bausteine anzubieten.

Auswertungsmodell: Viele der vorangegangenen Aussagen gelten analog für das Auswertungsmodell. Im Folgenden soll nur auf einige Besonderheiten eingegangen werden. Im Hinblick auf den **Abstraktionsgrad** gibt es zwei unterschiedliche Sichten auf das Auswertungsmodell: Zum einen die Konfiguration der Parameter der Auswertung, zum anderen die Baumansicht der Auswertungsergebnisse.

Die Möglichkeit einer **schrittweisen Auswertung** war ursprünglich nicht gegeben. Bei der Auswertung von Qualitätsmodellen auf großen Software-Entwicklungsarchiven zeigte sich dies als Nachteil, da eine Änderung an einem Teil des Qualitätsmodells immer eine komplette Berechnung des Auswertungsmodells erforderte. Das Werkzeug wurde so angepasst, dass auch Qualitätsindikatoren einzeln, sowie nur die Qualitätsmerkmale berechnet werden können.

8.4.2. Benutzbarkeit

Deissenboeck et. al beschreiben allgemeine Anforderungen an Werkzeuge zur Qualitätsbewertung und deren zu Grunde liegende Metamodelle [DJLW09]. Im Folgenden soll diskutiert werden, inwieweit diese erfüllt werden können:

- *Das Modell soll nicht auf automatisch auswertbare Metriken beschränkt sein.* In erster Linie zielt QMetric auf automatisch auswertbare Metriken ab. Es ist allerdings denkbar, dass auch auf manuell erhobene Werte mit Hilfe eines Adapters zugegriffen wird.
- *Idealerweise gibt es ein eigenes Beschreibungsmodell für Metriken.* Dies ist bei QMetric mit der Sprache ITMS gegeben. Bei Verwendung von Metriken aus anderen Werkzeugen muss im Qualitätsmodell entweder eine

präzise Beschreibung der Metrik selbst, oder eine Referenz auf die entsprechende Metrik in dem Werkzeug angegeben werden.

- *Die Aggregation von Qualitätsmerkmalen und -Indikatoren muss für den Anwender verständlich sein.* Die Umformfunktionen und Kombinationsfunktionen im Qualitätsmodell bieten ein verständliches und einfaches Schema, um eine solche Aggregation zu beschreiben. Nichtsdestotrotz müssen in einem konkreten Qualitätsmodell ausreichende Erläuterungen zu den Qualitätsmerkmalen annotiert werden.

- *Ein Bewertungsmodell sollte es ermöglichen, für unterschiedliche Teile der Software unterschiedliche Anforderungsprofile zu definieren.* Da das Werkzeug explizit die benutzerdefinierte Anpassung von Qualitätsmodellen unterstützt, können solche Anforderungsprofile durch Anpassung eines als Schablone gegebenen Qualitätsmodells erzeugt werden.

Des Weiteren wurden auch hier einige bewährte Praktiken zum Entwurf domänenspezifischer Sprachen eingesetzt [Völ08]. Die Notation orientiert sich an aus der Domäne bereits bekannten Darstellungsformen. Die Trennung in Qualitätsmodell-Editor und Auswertungswerkzeug setzt das Konzept der *Viewpoints* um [Völ08]. Analog dazu findet sich die Trennung der Zuständigkeiten bei Qualitätsmodell und Auswertungsmodell. Die Trennung könnte noch weiter getrieben werden, indem das Auswertungsmodell in die Konfiguration der Auswertung und ein Ergebnismodell aufgeteilt würde. Darauf wurde verzichtet, um dem Anwender den Überblick über die benötigten Modelle zu erleichtern.

8.4.3. Ausdrucksstärke

Bei diesem Aspekt ist das Ziel nicht eine maximale Ausdrucksstärke, sondern eine Ausdrucksstärke die den Anforderungen der Domäne am besten entspricht. Geeignete Einschränkungen der Ausdrucksstärke verringern die Komplexität aus Sicht des Anwenders.

Für die in den Fallstudien entwickelten Qualitätsmodelle haben sich die zur Verfügung stehenden Umform- und Kombinationsfunktionen als ausreichend erwiesen, um die gewünschte Aggregation der Messwerte zu beschreiben. In Abschnitt 6.3 wurde bereits diskutiert, wie eine unterschiedliche Priorisierung von Qualitätsmerkmalen, als auch die Definition von Qualitätsstufen abgebildet werden kann. Prinzipiell können die Fallstudien allerdings nur untermauern, dass die Ausdrucksstärke angemessen ist, und nicht mit einer statistischen Signifikanz zeigen [RH09].

Für eine Bewertung der Ausdrucksstärke ist außerdem ein Vergleich mit anderen werkzeuggestützten Ansätzen zur Modellierung von Qualitätsmodellen hilfreich. Von den in Abschnitt 2.2.2 vorgestellten Ansätzen kommt nur das Werkzeug ConQAT dazu in Frage [DJH+08], da es auf die Anbindung von beliebigen Messwerkzeugen ausgelegt ist.

ConQAT ermöglicht es dem Anwender, in einem grafischen Editor als Prozessoren bezeichnete Bausteine mit gerichteten Beziehungen zu verknüpfen. Prozessoren können verschiedene Aufgaben übernehmen, wie etwa die Ermittlung von Metrikergebnissen, deren Aggregation oder Visualisierung. Es gibt allgemein verwendbare Prozessoren (beispielsweise für die Zählung, Summierung, Sortierung, Filterung oder Klassifizierung von Werten) und Prozessoren für spezialisierte Aufgaben (beispielsweise für das Auffinden von dupliziertem Code). Parameter für einen Prozessor können in einem Eigenschaften-Dialog eingestellt werden. Die zwischen den Prozessoren weitergereichten Datenobjekte sind hierarchisch strukturierte Schlüssel-Wert-Paare. Der Benutzer kann ConQAT durch die Implementierung von eigenen Prozessoren erweitern.

Mit Hilfe der allgemein verwendbaren Prozessoren und gegebenenfalls einigen Erweiterungen durch eigene Prozessoren können prinzipiell die von QMetric unterstützten Qualitätsmodelle auch mit ConQAT nachgebildet werden. Der Ansatz von ConQAT ist insgesamt mächtiger als der des QMetric-Qualitätsmodell-Editors, da auch die Anbindung von unterschiedlichen Metrikwerkzeugen als auch unterschiedliche Visualisierungsformen mit Hilfe der Prozessoren modelliert werden kann. Der Qualitätsmodell-Editor von QMetric bietet allerdings einen engeren Rahmen für den vorgesehenen Einsatzzweck. Dadurch können beispielsweise globale Bedingungen an das Qualitätsmodell besser überprüft werden.

8.4.4. Wiederverwendbarkeit

Ein einmal definiertes Qualitätsmodell kann in Qualitätsauswertungen wiederverwendet werden. Es gibt keine konzeptionelle Unterstützung für die Wiederverwendung von Teilen eines Qualitätsmodells in anderen Qualitätsmodellen. Es können keine Abhängigkeiten zwischen Qualitätsmodellen definiert werden. Es hat sich bisher als ausreichend erwiesen, ein gegebenes Qualitätsmodell als Vorlage für ein neues Modell zu kopieren und anzupassen.

8.4.5. Entwicklungskosten

Die Frage ist hier, wie sich die Verwendung der Qualitätsmodelle auf den Aufwand für die Implementierung eines Auswertungsschemas für die Qualitätsbewertung auswirkt. Es ist offensichtlich, dass der Aufwand im Vergleich zur Verwendung einer allgemeinen Programmiersprache deutlich geringer ist.

Interessanter ist der Vergleich zum Einsatz von Tabellenkalkulations-Software und spezialisierten Umgebungen wie ConQAT [DJH+08].

Vermutlich kann ein erfahrener Anwender ein Auswertungsschema in vergleichbarer Zeit mit Hilfe einer Tabellenkalkulation erstellen. Die Metrikergebnisse müssten dazu geeignet importiert werden. Der Einsatz einer Tabellenkalkulation wirkt sich hingegen nachteilig auf die Aspekte Nähe der Abbildung, Viskosität, versteckte Abhängigkeiten, Sichtbarkeit und Fehleranfälligkeit aus.

Mit dem in dem Werkzeug ConQAT verfolgten Ansatz (vgl. Abschnitt 8.4.3), ließen sich vermutlich ebenfalls in vergleichbarer Zeit entsprechende Qualitätsmodelle erstellen. Dazu müssten allerdings entsprechende Prozessoren für die Erhebung von Metriken, als auch für die gewünschte Aggregation der Messwerte implementiert werden. Im Moment ist für ConQAT ein Prozessor zur Anbindung von Bugzilla verfügbar, mit dem allerdings nicht die historischen Daten zu Änderungsanträgen abgefragt werden können. Im Hinblick auf die Erlernbarkeit wäre eine Lösung mit ConQAT ähnlich zu bewerten, wie der QMetric-Qualitätsmodell-Editor. Die Dimension „Nähe der Abbildung“ wäre bei ConQAT dadurch beeinträchtigt, dass potentiell mehr Modellelemente und Möglichkeiten zu deren Komposition zur Verfügung stehen.

8.4.6. Zuverlässigkeit

Gegenüber den im vorherigen Abschnitt genannten Alternativen zur Implementierung eines Auswertungsschemas, kann die Verwendung der Qualitätsmodelle und Auswertungsmodelle gewisse konzeptionelle Fehler verhindern, da eine engere Systematik vorgegeben ist, und das zu Grunde gelegte Metamodell für die Qualitätsmodelle weniger komplex ist.

8.5. Datenqualität in Software-Entwicklungsarchiven

Grundsätzlich ist die Anwendbarkeit des vorgestellten Auswertungsansatz auf Software-Entwicklungsarchiven durch die folgenden Faktoren eingeschränkt:

1. Art der erfassten Daten
2. Qualität der erfassten Daten
3. Ausdruckstärke der Sprache zur Spezifikation von Metriken

Der erste Punkt ist offensichtlich. Je mehr Daten über den tatsächlichen Arbeitsablauf erfasst werden, desto präzisere Aussagen lassen sich aus der Auswertung gewinnen. Zur genauen Dokumentation des Arbeitsablaufs gehört beispielsweise die Erfassung von Plan- und Ist-Daten zu Aufwänden und Terminen, die beteiligten Personen sowie Informationen zur Rückverfolgbarkeit zwischen verschiedenen Artefakten.

Die Ausdruckstärke von ITMS wurde schon in Abschnitt 8.3.3 diskutiert. Im Folgenden möchten wir die Auswirkungen der Datenqualität betrachten.

Da die Daten unabhängig von den Zielen der Messung erhoben werden, sind möglicherweise nicht die Anforderungen an Vollständigkeit und Konsistenz der Daten erfüllt, die bei der Entwicklung einer Metrik angenommen werden [RH09]. Zunächst werden verwandte Arbeiten zur Problematik der Datenqualität diskutiert. In Abschnitt 8.5.2 werden eigene Erfahrungen geschildert. Schließlich folgen in Abschnitt 8.5.3 Hinweise zur Berücksichtigung der Datenqualität bei der Definition von Metriken in ITMS.

8.5.1. Verwandte Arbeiten

Das Thema der Datenqualität in Software-Entwicklungsarchiven wurde aus verschiedenen Blickwinkeln schon in anderen Studien betrachtet. Bettenburg et al. untersuchten im Rahmen einer Umfrage unter Entwicklern aus den Projekten Eclipse, Apache und Mozilla die Frage, wie brauchbar die in einem Problembericht gegebenen Informationen sind [BJS+08]. Diese Umfrage ergab, dass Entwickler hauptsächlich Informationen in der textlichen Beschreibung vermissen, beispielsweise Informationen zur Reproduzierbarkeit oder dem beobachtetem Verhalten. Außerdem kommt es vor, dass Attribute wie die Komponente, Versionsnummer oder das Betriebssystem falsch angegeben werden.

Solche fehlerhaft gesetzten Attribute werden bei der Bearbeitung des Problemberichtes in der Regel korrigiert. Somit kann dies in einer ITMS Metrik-Spezifikation entsprechend berücksichtigt werden, beispielsweise durch Verwendung von fixierten Attributen (siehe Abschnitt 4.5) oder die Verwendung von entsprechenden Ereignisfiltern (siehe Abschnitt 4.4.1.3).

Herraiz et al. untersuchten die Verwendung der Attribute Priorität und Schweregrad bei Problemberichten zu Eclipse [HGGBR08]. Sie stellen fest, dass nicht alle Ersteller von Problemberichten in der Lage sind, die Klassifikation des Schweregrades richtig zu verwenden. Zudem wird das Attribut Priorität von den Entwicklern nicht konsistent verwendet.

Aranda und Venolia verglichen im Rahmen eine Feldstudie die im Issue-Tracking-System erfasste Historie von Problemberichten mit der tatsächlichen Bearbeitung, die in Form von Interviews bei den beteiligten Entwicklern abgefragt wurde [AV09]. Untersucht wurden zehn Problemberichte bei der Produktentwicklung von Microsoft. Die Autoren unterscheiden dabei die folgenden Arten von Analysen der Historie von Problemberichten. Jede Ebene erweitert die vorhergehenden Ebenen.

1. Automatisierte Analyse der im Issue-Tracking-System erfassten Informationen.
2. Zusätzliche automatisierte Analyse von Kommunikationsarchiven.
3. Analyse der in Software-Entwicklungsarchiven erfassten Informationen durch einen Experten.
4. Direkte Befragung von Personen, die bei Erstellung und Bearbeitung des Problemberichts mitgewirkt haben.

Bei einigen der untersuchten Problemberichte ergaben sich deutliche Unterschiede bei den Analysen. Darunter fallen falsch gesetzte Attribute, fehlende Verknüpfungen zu relevanten Quellcodeänderungen oder verwandten Problemberichten sowie unvollständige Dokumentation der beteiligten Personen. Abgesehen von diesen Problemen bei den Rohdaten finden sich eine ganze Reihe von Informationen, die erst auf der Ebene 4 aufgedeckt werden können, wie etwa die Begründung zu bestimmten Entscheidungen, unterschiedliche Kulturen bei

Gruppen von beteiligten Entwicklern, Abstimmungsprobleme zwischen Teams oder Abteilungen und ungeklärte Verantwortlichkeiten.

8.5.2. Erfahrungen aus den Fallstudien

Ähnliche Beobachtungen machten wir auch während der praktischen Anwendung. Beispielsweise wurde beim Issue-Tracking-System von Eclipse zu einem bestimmten Zeitpunkt das Benutzungsschema geändert. Ursprünglich war es möglich, dass ein Änderungsantrag in den Zustand *Resolved* wechselt mit einer der „Lösungen“ *Later* oder *Remind*. Da dies irreführend interpretiert werden kann, wurde von der Verwendung dieser beiden Attributwerte abgesehen. Stattdessen soll ein solcher Änderungsantrag einem Meilenstein namens *future* zugeordnet werden, mit dem Schlüsselwort *needinfo* markiert, oder seine Priorität verringert werden [SSL09].

Es zeigte sich, dass bei den einzelnen Projekten bei Eclipse und Gnome zum Teil eigene Benutzungsschema für das Issue-Tracking-System existieren. So werden beispielsweise die Bearbeitungszustände *Unconfirmed*, *Verified* und *Closed* nur von einem Teil der Projekte benutzt. Zum Teil ist die Verwendung auch innerhalb eines Projekts nicht konsistent. Es gibt zwar einheitliche Vorgaben für die Verwendung des Issue-Tracking-Systems, allerdings haben die Projekte die Freiheit davon abzuweichen. Oftmals ist es im Nachhinein schwierig nachzuvollziehen, zu welchem Zeitpunkt eine Änderung des Benutzungsschema stattfand. Häufig gibt es einen kontinuierlichen Übergang.

Auch bei der KISTERS AG hat sich die Verwendung bestimmter Attribute im Issue-Tracking-System über die Zeit geändert [Ric09]. Die Verwendung von Attributen im Issue-Tracking-System ist ebenfalls durch etwas abweichende Benutzungsschemas in den verschiedenen Arbeitsgruppen geprägt.

Zusammenfassend lässt sich also feststellen: Die in Software-Entwicklungsarchiven enthaltenen Informationen sind unvermeidbar unvollständig und inkonsistent. Es wird nur dokumentiert, was ökonomisch erfasst werden kann und im Rahmen der Bearbeitung eines Änderungsantrags gerade nötig erscheint.

8.5.3. Umgang mit mangelnder Datenqualität

Die Frage ist also, wie mit diesen Gegebenheiten bei der Entwicklung von Metriken umgegangen werden kann.

Das in Kapitel 5 vorgestellte Vorgehen zur Entwicklung von Metriken enthält explizit die stichprobenartige Überprüfung von Annahmen. Einige der zuvor geschilderten Probleme können auf diesem Weg aufgedeckt werden, beispielsweise wenn Attribute regelmäßig nur unvollständige Angaben enthalten oder inkonsistent verwendet werden. Hilfreich ist es insbesondere, Extremwerte bei den Metrikergebnissen im Detail zu überprüfen. Zudem müssen die Stichproben

aus dem gesamten Beobachtungszeitraum und aus mehreren Produkten unterschiedlichen Charakters gewählt werden.

Werden Probleme bei der Datenqualität aufgedeckt, gibt es Mittel diese zu umgehen. Teilweise können die Metriken so angepasst werden, dass unterschiedliche Benutzungsschema reflektiert werden. Beispielsweise kann bei der Zählung von gelösten Änderungsanträgen ein Ereignisfilter verwendet werden, der Lösungen mit *Later* oder *Remind* nicht berücksichtigt [SL08]. Des Weiteren kann ein zweistufiges Vorgehen angewandt werden, bei dem zunächst mit einer Metrik die Verwendung eines bestimmten Attributs bewertet wird. Detaillierte Auswertungen basierend auf diesem Attribut werden nur dann weiter durchgeführt, wenn ein Projekt dies auch in ausreichendem Maße einsetzt.

Letztlich muss dann bei der Interpretation der Ergebnisse auch immer das Wissen der Beteiligten über die Verwendung des Issue-Tracking-Systems einbezogen werden. Weiterhin können Maßnahmen eingeleitet werden, um die Datenqualität im Issue-Tracking-System zu verbessern und die Benutzungsschema zu vereinheitlichen. Im Projekt Gnome wurde beispielsweise ein eigenes Team für die Klassifizierung von eingehenden Änderungsanträgen aufgebaut. Durch verschiedene Maßnahmen, wie etwa koordinierte Treffen von freiwilligen Helfern, explizite Beschreibung von Regeln für die Klassifizierung, und eine Verbesserung der Kommunikation zu den Entwicklern, konnte die Klassifizierung der Änderungsanträge und somit auch die Qualität der erfassten Daten über die Jahre verbessert werden [SL09].

8.6. Resümee

Wichtige Aspekte der in den vorherigen Abschnitten vorgenommenen Bewertungen sind in Tabelle 8.3 zusammengefasst. Im Folgenden sollen die bisherigen Aussagen in Bezug gesetzt werden zu den in Abschnitt 3.2 formulierten Fragestellungen:

A. **Wie werden geforderte Qualitätsmerkmale des Prozesses in Bezug gesetzt zu den Metriken?** In den breit angelegten Fallstudien hat sich das vorgeschlagene Metamodell für Qualitätsmodelle als geeignet erwiesen. Insbesondere ermöglichen empirische Vergleichsdaten eine intuitive Interpretation der resultierenden Kennzahlen. Natürlich ist es nicht sicher, inwieweit sich die Erfahrungen der Fallstudien auf andere Umgebungen übertragen lassen. Letztlich liefern die Fallstudien konkretes, kontextbezogenes Erfahrungswissen, welches als solches zentral ist für Lernprozesse und menschliche Expertenarbeit [Fly06].

B. **Wie können geeignete Metriken kostengünstig entwickelt und validiert werden?** Das in Kapitel 5 vorgestellte Verfahren wurde im Rahmen der praktischen Anwendung der Sprache ITMS entwickelt. Zudem konnten Erfahrungen beim Umgang mit typischen Problemen bei der

	Vorteile	Nachteile / Einschränkungen
Werkzeug-unterstützung	Ausgereifte Metrikberech-nungskomponente Interoperabilität für unterschiedliche Software-Entwicklungsarchive GUI unterstützt Erlernen von ITMS	Effizienz beim Ressourcenverbrauch kann verbessert werden
Sprache ITMS	Ausdrucksstark Hohes Abstraktionsniveau Gute Wartbarkeit der Spezifikationen	Einarbeitungsaufwand für domänenspezifische Sprache
Ansatz zur Qualitäts-modellierung	Flexibel und einfach verständlich	Weitere Erfahrungen notwendig (Anbindung weiterer Messwerkzeuge und Anwendung in anderen Domänen)

Tabelle 8.3.: Zusammenfassende Bewertung der entwickelten Konzepte und Werkzeuge

Datenqualität gesammelt werden. Somit handelt es sich dabei um bewährte Vorgehensweisen. Die Anwendung erfordert natürlich Erfahrung mit der Sprache ITMS, sowie Kenntnisse über das Benutzungsschema der ausgewerteten Software-Entwicklungsarchive. Die Ausdrucksstärke der Sprache ITMS und die Knappheit der Metrik-Spezifikationen erleichtern das Erlernen der Sprache, und ermöglichen gleichzeitig eine schnelle iterative Entwicklung und Validierung von Metriken.

C. **Wie können Software-Entwicklungsarchive auf flexible Weise ausgewertet werden?** Mit der Referenzimplementierung der Berechnungskomponente für ITMS und der Anbindung der Issue-Tracking-Systeme Bugzilla und Mantis, sowie der Konfigurationsmanagement-Systeme CVS und Subversion, wurde eine allgemein verwendbare Werkzeugunterstützung vorgestellt. Die QMetric-Werkzeug-Suite ermöglicht eine gute Erweiterbarkeit und Anpassbarkeit der Sprache ITMS. Im Kontext der Ermittlung von Prozessmetriken aus Software-Entwicklungsarchiven wird eine vergleichbare Ausdrucksstärke bei der Definition von Metriken bei keinem anderen der diskutierten Ansätze und Werkzeuge erreicht.

Die Änderungsanträge spielen die zentrale Rolle bei diesem Ansatz. Dies

ermöglicht es, die Sprache ITMS konzeptuell einfach zu halten. Um allerdings Konfigurationsmanagement-Systeme in die Auswertungen einzubeziehen, ist eine Verknüpfung zwischen Änderungsanträgen und Transaktionen im Konfigurationsmanagement-System erforderlich. Außerdem ergeben sich Einschränkungen bei bestimmten Warenkorb-Abfragen (vgl. Abschnitt 2.4.1) in Bezug auf Konfigurationsmanagement-Systeme (siehe Abschnitt 8.3.3).

8.6.1. Offene Fragen

Es ist eine offene Frage, inwieweit andere Ansätze aus dem Bereich *Mining Software Repositories* in die gegebene Werkzeuginfrastruktur für ITMS integriert werden können. Denkbar wäre beispielsweise Änderungen im Konfigurationsmanagement-System zu klassifizieren, beispielsweise Fehler verursachende Änderungen, oder durchgeführte Refactorings. Diese Informationen könnten wiederum bei der Ermittlung von Metriken herangezogen werden.

Mit den beschriebenen Fallstudien wurde eine breite Anwendbarkeit des Ansatzes zur Qualitätsmodellierung gezeigt. Es konnte allerdings noch wenig praktische Erfahrung gesammelt werden, wie solche Auswertungen auf Basis von Qualitätsmodellen in einen kontinuierlichen Verbesserungsprozess integriert werden. Hier stellen sich beispielsweise Fragen, wie der Umgang mit einem Qualitätsmodell und die auf Basis des Modells gewonnenen Aussagen an die unterschiedlichen Anspruchsgruppen (*Stakeholder*) innerhalb einer Organisation kommuniziert werden. Es ist offen, wie eine unternehmens- oder projektspezifische Anpassung von Qualitätsmodellen unterstützt werden kann.

Weiterhin blieben in der bisherigen Diskussion ethische Aspekte der Anwendung eines solchen Auswertungsansatzes unberührt. In diesem Punkt lassen sich noch nicht vollständige Leitlinien geben. Nach Ansicht des Autors sind folgende Punkte wichtig: Die verwendeten Metrik-Spezifikationen müssen offen zugänglich sein. Vor der Verwendung im Rahmen eines Qualitätsmodells müssen Metriken sehr sorgfältig validiert, und das Feedback der Beteiligten berücksichtigt werden. Es darf keine Bewertung von individueller Arbeitsleistung auf Basis solcher Metriken erfolgen.

9. Zusammenfassung und Ausblick

> Wer sich der Praxis hingibt ohne Wissenschaft ist wie der Steuermann, der ein Schiff ohne Ruder und Kompass besteigt und nie weiß, wohin er fährt.
>
> *(Leonardo da Vinci, 1452 - 1519)*

Ausgangspunkt für diese Arbeit war, dass die bei der Softwareentwicklung routinemäßig gesammelten Daten nur unzureichend zur Bewertung, Steuerung und Verbesserung von Prozessen verwendet werden. Gängige Werkzeuge für die Metrikauswertung auf solchen Software-Entwicklungsarchiven sind zu unflexibel in Bezug auf die definierbaren Metriken und die auswertbaren Archive. Um organisationsspezifische Informationsbedürfnisse zu erfüllen, müssen deshalb eigene Metriken definiert, händisch implementiert und validiert werden.

Diese Arbeit stellt Werkzeuge und Verfahren vor, welche die Entwicklung von Metriken und ihre Verwendung zur Prozessbewertung erleichtern. Die vorgestellte deklarative Sprache ITMS bietet eine hohe Flexibilität bei der Spezifikation von Metriken, und ermöglicht gleichzeitig eine kompakte und präzise Beschreibung. Der Einsatz einer solchen Sprache ermöglicht ein schnelles, iteratives Verfahren zur Entwicklung und Validierung von Metriken. Dadurch können typische Fehler bei der Definition frühzeitig aufgedeckt oder gleich vermieden werden.

Weiterhin wurde ein Metamodell für benutzerdefinierte Qualitätsmodelle eingeführt. Dieses ermöglicht eine systematische Qualitätsbewertung auf einer höheren Abstraktionsebene.

Die genannten Sprachen und Methoden werden durch die Werkzeug-Suite QMetric unterstützt. Zu dieser Werkzeug-Suite gehört eine Referenzimplementierung für die Berechnung von Metriken in der Sprache ITMS. Weiterhin gibt es jeweils Werkzeuge für das Erstellen und Editieren von Qualitätsmodellen, und für automatisierte durchgeführte Qualitätsbewertungen. Wichtige Erfolgsfaktoren bei der Entwicklung der Werkzeug-Suite waren zum einen die langjährige Zusammenarbeit mit der KISTERS AG als industriellem Kooperationspartner, und zum anderen die Bereitstellung als Open Source Projekt.

Die vorgestellten Ansätze wurden in mehreren Fallstudien im industriellen Umfeld und im Open Source Bereich angewandt. Die Ausdrucksstärke von ITMS hat sich dabei als angemessen erwiesen. Es konnten neue Einblicke in Bezug auf die Qualität der Prozesse gewonnen werden, die bisher nur mit weit größerem Aufwand oder gar nicht möglich waren. Bei einem Teil der Fallstudien konnten auch die Werkzeuge zur Qualitätsmodellierung schon erprobt werden. Allerdings müssen hier noch weitere Erfahrungen darüber gesammelt werden,

wie eine solche Qualitätsbewertung für die Steuerung und Verbesserung der Entwicklungsprozesse eingesetzt werden kann.

Ausblick

Ein explizites Ziel beim Entwurf der QMetric-Werkzeug-Suite war eine gute Erweiterbarkeit. Diese konnte im Rahmen der Anbindung von Konfigurationsmanagement-Systemen bestätigt werden. Eine weitere nahe liegende Erweiterung ist eine Auswertung von Daten aus Testfallmanagementsystemen. Hier stehen bereits Systeme zur Verfügung, die eine Verknüpfung zwischen Änderungsanträgen und zugehörigen Testfällen ermöglichen [Tesa, Tesb].

Die Erfassung von zusätzlichen Daten im Issue-Tracking-System ermöglicht ebenfalls weitere aussagekräftige Auswertungen. Ein Beispiel ist der Ansatz der *Orthogonal Defect Classification*, bei dem gefundene Fehler zusätzlich nach Fehlerart und Fehlerursache kategorisiert werden [CBC+92]. Die Werteverteilung dieser beiden Attribute kann dann mit erwarteten Verteilungen für einzelne Phasen des Entwicklungsprozesses verglichen werden. Dies erleichtert es, die Qualitätsentwicklung eines Produkts zu bewerten, und Fehlerursachen im Entwicklungsprozess zu identifizieren.

Weiterhin könnte die QMetric-Werkzeug-Suite hinsichtlich der Aspekte Problembehandlung und Metrik-Handling erweitert werden. Funktionen zum Metrik-Handling ermöglichen die Annotation von Bewertungen bzw. Entscheidungen zu konkreten Messwerten [RW05]. Problembehandlung bedeutet, dass das Werkzeug auch Handlungsanweisungen für den Umgang mit identifizierten Problemen liefert. Um dies zu realisieren, ist allerdings noch deutlich mehr Erfahrung in der Anwendung von Qualitätsmodellen und zugehörigen Metriken notwendig.

Ein breites Anwendungsfeld für den vorgestellten Ansatz ergibt sich im Bereich der Bewertung von Open Source Projekten. Bei Reifegradmodellen für Open Source Softwareentwicklung kommen bisher nur recht einfach zu ermittelnde Metriken zum Einsatz (vgl. Abschnitt 2.3.4). Die hier entwickelten Techniken ermöglichen deutlich aussagekräftigere Messungen (vgl. Abschnitt 8.1). Die Werkzeuginfrastruktur trägt zudem zu einer besseren Wiederholbarkeit und Reproduzierbarkeit der Messungen bei, was momentan eine deutliche Schwäche der empirischen Forschung im Bereich des Software Engineerings ist [GS09b].

Über das Anwendungsfeld der Bewertung von Software-Prozessen hinaus könnte der vorgestellte Ansatz auch in anderen Bereichen angewandt werden. Viele Software-Systeme, mit denen der Arbeitsablauf zur Bearbeitung eines betriebswirtschaftlichen Objekts erfasst wird, haben Ähnlichkeiten zu Issue-Tracking-Systemen. Es stellt sich also die Frage, inwieweit durch die Sprache ITMS typische Auswertungen auf solchen Systemen unterstützt werden.

Grundsätzlich ermöglicht der hier vorgestellte Ansatz mit geringem Aufwand eine höhere Transparenz des Entwicklungsprozesses. Es werden nur Daten benutzt, die ohnehin routinemäßig erhoben werden. Es ist also kein zusätzlicher

Aufwand für die Datenerfassung nötig. Weiterhin können auch Daten aus der Vergangenheit in die Auswertung mit einbezogen werden.

Die höhere Prozesstransparenz bietet eine bessere Basis, um Entscheidungen zu treffen und die richtigen Prioritäten zu setzen. Die größte Herausforderung beim Einsatz in der industriellen Praxis bleibt allerdings, Sinn und Nutzen der Metriken zu vermitteln [Osb97]. Dies ist eine Voraussetzung dafür, dass die Verwendung von Metriken und angemessene Reaktionen auf Abweichungen ein integraler Bestandteil der Arbeitsabläufe werden.

A. Anhang

A.1. Beispiel-Metriken zur Prozessqualität in Eclipse

Listing A.1: Metrik „Änderungsanträge ohne Reaktion in 2 Tagen“

```
<metric>
 <baseFilter>
  <none/>
 </baseFilter>
 <groupingParameters>
  <fieldGrouping>product</fieldGrouping>
 </groupingParameters>
 <groupEvaluations>
  <calculation name="No reaction within 2 days">
    <divide>
     <multiply>
      <constant>100.0</constant>
      <sum valueCalculator="default" />
     </multiply>
     <count valueCalculator="default" />
    </divide>
  </calculation>
 </groupEvaluations>
 <valueCalculators>
  <intervalLength id="default">
   <from>
    <create />
   </from>
   <to>
    <or>
     <commentAdded />
     <transition field="status" />
     <transition field="assignee" />
     <transition field="priority" />
     <transition field="component" />
     <transition field="targetMilestone" />
     <transition field="summary" />
     <transition field="product" />
    </or>
   </to>
   <threshold thresholdInDays="2" useThresholdWeight="true" />
  </intervalLength>
 </valueCalculators>
 <evaluationTimePeriod>
  <timePeriod>
```

```
    <start>2003-07-01</start>
    <end>2008-06-30</end>
   </timePeriod>
  </evaluationTimePeriod>
  <timePeriodGranularity>
   <customGranularity>
    <aggregateAt>2004-06-30</aggregateAt>
    ...
    <aggregateAt>2008-06-30</aggregateAt>
   </customGranularity>
  </timePeriodGranularity>
  <fixedFields />
</metric>
```

Listing A.2: Metrik „Falsch negativ klassifizierte Änderungsanträge“

```
<metric>
 <baseFilter>
  <none />
 </baseFilter>
 <groupingParameters>
  <fieldGrouping>product</fieldGrouping>
 </groupingParameters>
 <groupEvaluations>
  <calculation name="Reopened rate of rejected CRs">
   <divide>
    <multiply>
     <constant>100.0</constant>
     <countUnique valueCalculator="Rejected_and_Reopened" />
    </multiply>
    <countUnique valueCalculator="Rejected" />
   </divide>
  </calculation>
 </groupEvaluations>
 <valueCalculators>
  <countEvents id="Rejected_and_Reopened">
   <event>
    <and>
     <transition field="resolution">
      <from>INVALID</from>
      <from>WONTFIX</from>
      <from>DUPLICATE</from>
      <from>WORKSFORME</from>
      <from>MOVED</from>
      <from>NOT_ECLIPSE</from>
     </transition>
     <stateFilter>
      <value field="status">REOPENED</value>
     </stateFilter>
    </and>
   </event>
   <weight>
    <default />
```

```
    </weight>
   </countEvents>
   <countEvents id="Rejected">
    <event>
     <and>
      <transition field="resolution">
       <to>INVALID</to>
       <to>WONTFIX</to>
       <to>DUPLICATE</to>
       <to>WORKSFORME</to>
       <to>MOVED</to>
       <to>NOT_ECLIPSE</to>
      </transition>
      <stateFilter>
       <value field="status">RESOLVED</value>
      </stateFilter>
     </and>
    </event>
    <weight>
     <default />
    </weight>
   </countEvents>
  </valueCalculators>
  <evaluationTimePeriod>
   <timePeriod>
    <start>2003-07-01</start>
    <end>2008-06-30</end>
   </timePeriod>
  </evaluationTimePeriod>
  <timePeriodGranularity>
   <customGranularity>
    <aggregateAt>2004-06-30</aggregateAt>
    ...
    <aggregateAt>2008-06-30</aggregateAt>
   </customGranularity>
  </timePeriodGranularity>
  </fixedFields>
</metric>
```

Listing A.3: Metrik „Priorität von Problemberichten mit hohem Schweregrad“

```
<metric>
 <baseFilter>
  <none />
 </baseFilter>
 <groupingParameters>
  <none />
 </groupingParameters>
 <groupEvaluations>
  <calculation name="Average priority of severe CRs">
   <divide>
     <sum valueCalculator="Severe_CRs_weighted_by_priority" />
     <count valueCalculator="Severe_CRs_weighted_by_priority" />
```

```
    </divide>
   </calculation>
  </groupEvaluations>
  <valueCalculators>
   <countEvents id="Severe_CRs_weighted_by_priority">
    <event>
     <and>
      <transition field="status">
       <to>RESOLVED</to>
      </transition>
      <stateFilter>
       <or>
         <value field="severity">blocker</value>
         <value field="severity">critical</value>
       </or>
      </stateFilter>
     </and>
    </event>
    <weight>
     <mapping field="priority">
      <map from="P1" to="1" />
      <map from="P2" to="2" />
      <map from="P3" to="3" />
      <map from="P4" to="4" />
      <map from="P5" to="5" />
     </mapping>
    </weight>
   </countEvents>
  </valueCalculators>
  <evaluationTimePeriod>
   <timePeriod>
    <start>2003-07-01</start>
    <end>2008-06-30</end>
   </timePeriod>
  </evaluationTimePeriod>
  <timePeriodGranularity>
   <customGranularity>
    <aggregateAt>2004-06-30</aggregateAt>
    ...
    <aggregateAt>2008-06-30</aggregateAt>
   </customGranularity>
  </timePeriodGranularity>
  <fixedFields />
</metric>
```

A.2. Beispiel eines Qualitätsmodells in der industriellen Softwareentwicklung

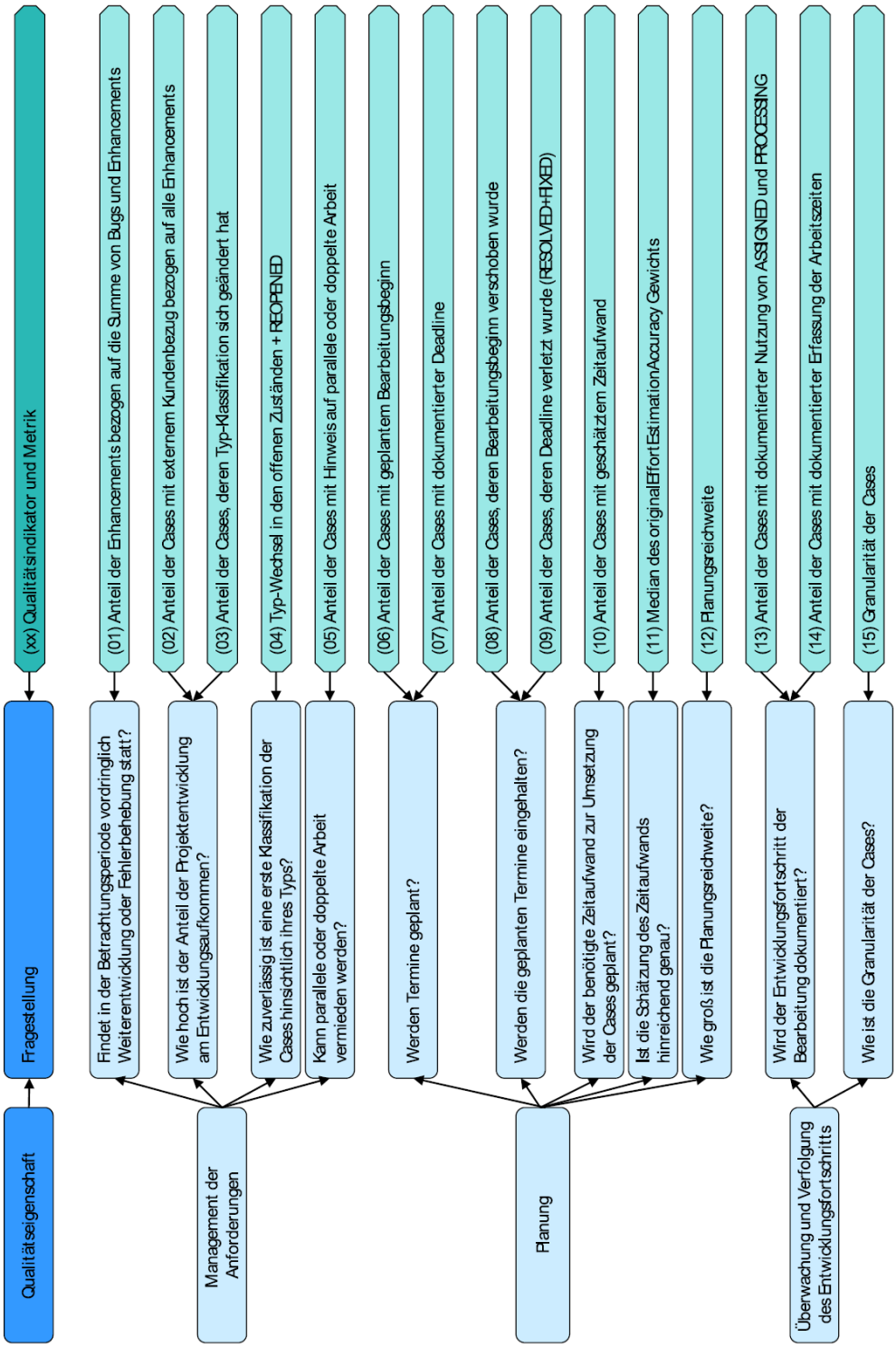


Abbildung A.1.: Qualitätsmodell zur Bewertung eines industriellen Software-Entwicklungsprozesses (Teil 1) [Ric09]

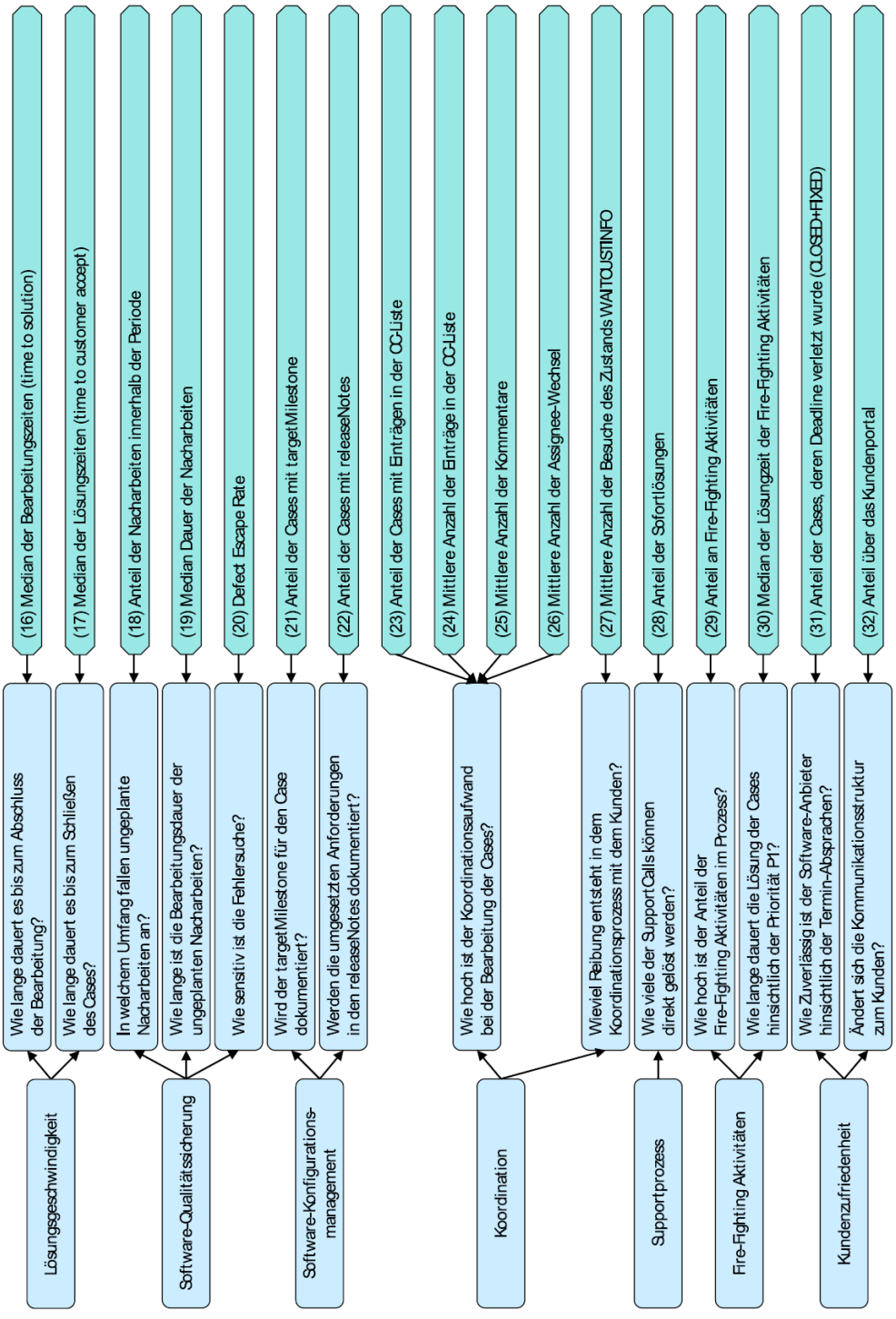


Abbildung A.2.: Qualitätsmodell zur Bewertung eines industriellen Software-Entwicklungsprozesses (Teil 2) [Ric09]

Literaturverzeichnis

[AA08] April, Alain und Alain Abran: *Software Maintenance Management: Evaluation and Continuous Improvement.* IEEE Computer Society, Wiley, Hoboken, NJ, 2008.

[AA09] April, Alain und Alain Abran: *A Software Maintenance Maturity Model (S3M): Measurement Practices at Maturity Levels 3 and 4.* Electron. Notes Theor. Comput. Sci., 233:73–87, 2009.

[ABD+04] Abran, Alain, Pierre Bourque, Robert Dupuis, James W. Moore und Leonard L. Tripp (Herausgeber): *Guide to the Software Engineering Body of Knowledge - SWEBOK.* IEEE Press, Piscataway, NJ, USA, 2004.

[AH06] Askari, Mina und Richard C. Holt: *Information Theoretic Evaluation of Change Prediction Models for Large-Scale Software.* In: Diehl, Stephan et al. [DGH06], Seiten 126–132.

[AHAD05] April, Alain, Jane Huffman Hayes, Alain Abran und Reiner R. Dumke: *Software Maintenance Maturity Model (SM^{mm}): The Software Maintenance Process Model.* Journal of Software Maintenance, 17(3):197–223, 2005.

[AHK+01] Aoki, Atsushi, Kaoru Hayashi, Kouichi Kishida, Kumiyo Nakakoji, Yoshiyuki Nishinaka, Brent Reeves, Akio Takashima und Yasuhiro Yamamoto: *A Case Study of the Evolution of Jun: An Object-Oriented Open-Source 3D Multimedia Library.* In: *Proceedings of the 23rd International Conference on Software Engineering, ICSE 2001, 12-19 May 2001, Toronto, Ontario, Canada*, Seiten 524–533. IEEE Computer Society, 2001.

[Alt] *Altova XMLSpy, Schema Tools.* http://www.altova.com/xmlspy/schema-tools.html. (Zugriff: 4. Januar 2010).

[AMPS74] Abernathy, David H., John S. Mancino, Charls R. Pearson und Dona C. Swiger: *Survey of Design Goals for Operating Systems.* SIGOPS Oper. Syst. Rev., 8(1):25–35, 1974.

[ATL] *ATL (ATLAS Transformation Language).* http://www.eclipse.org/m2m/atl/. (Zugriff: 4. Januar 2010).

[Ato06] Atos Origin: *Method for Qualification and Selection of Open Source software (QSOS) version 1.6.* http://qsos.org, April 2006. (Zugriff: 4. Januar 2010).

[AV09] Aranda, Jorge und Gina Venolia: *The secret life of bugs: Going past the errors and omissions in software repositories.* In: *ICSE* [ICS09], Seiten 298–308.

[Bal98] Balzert, Helmut: *Lehrbuch der Software-Technik: Software Management, Software-Qualitätssicherung und Unternehmensmodellierung.* Spektrum, Akad. Verlag, 1998.

[Bas89] Basili, Victor R.: *Software Development: a Paradigm for the Future.* In: *Computer Software and Applications Conference, 1989. COMPSAC 89., Proceedings of the 13th Annual International*, Seiten 471–485, Sep 1989.

[BBD07] Büren, Günter, Manfred Bundschuh und Reiner Dumke (Herausgeber): *Tagungsband des DASMA Software Metrik Kongresses MetriKon 2007, 15.-16. November 2007, Kaiserslautern*, Magdeburger Schriften zum Empirischen Software Engineering. Shaker, 2007.

[BBDA09] Breuker, Dennis, Jacob Brunekreef, Jan Derriks und Ahmed Nait Aicha: *Reliability of Software Metric Tools.* In: Abran, Alain, René Braungarten, Reiner R. Dumke, Juan Jose Cuadrado-Gallego und Jacob Brunekreef (Herausgeber): *Software Process and Product Measurement, International Conferences: IWSM 2008 and Mensura 2008, Amsterdam, The Netherlands, November 5-6, 2009. Industrial Track Papers*, Seiten 10–22. Hogeschool van Amsterdam, 2009.

[BBL76] Boehm, Barry W., John R. Brown und Myron Lipow: *Quantitative Evaluation of Software Quality.* In: *ICSE '76: Proceedings of the 2nd International Conference on Software Engineering*, Seiten 592–605, Los Alamitos, CA, USA, 1976. IEEE Computer Society Press.

[BCBP06] Baroni, Aline Lucia, Coral Calero, Fernando Brito e Abreu und Mario Piattini: *Object-Relational Database Metrics Formalization.* International Conference on Quality Software, 0:30–37, 2006.

[BCM+92] Basili, Victor, Gianluigi Caldiera, Frank McGarry, Rose Pajerski, Gerald Page und Sharon Waligora: *The Software Engineering Laboratory: an Operational Software Experience Factory.* In: *ICSE '92: Proceedings of the 14th International Conference on Software Engineering*, Seiten 370–381, New York, NY, USA, 1992. ACM.

[BCR84] Basili, V. R., G. Caldiera und H. D. Rombach: *The Experience Factory.* In: Marciniak, John J. (Herausgeber): *Encyclopedia of Software Engineering*, Seiten 469–476, Chichester, 1984. Wiley.

[BCR94] Basili, Victor R., Gianluigi Caldiera und H. Dieter Rombach: *The Goal Question Metric Paradigm.* In: Marciniak, John J. (Herausgeber): *Encyclopedia of Software Engineering*, Seiten 528–532, Chichester, 1994. Wiley.

[BDP06] Broy, Manfred, Florian Deissenboeck und Markus Pizka: *Demystifying maintainability.* In: *WoSQ '06: Proceedings of the 2006 International Workshop on Software Quality*, Seiten 21–26, New York, NY, USA, 2006. ACM.

[BDR96] Briand, Lionel C., Christiane M. Differding und H. Dieter Rombach: *Practical Guidelines for Measurement-Based Process Improvement.* Software Process Improvement and Practice, 2(4):253–280, 1996.

[BGD+06] Bird, Christian, Alex Gourley, Prem Devanbu, Michael Gertz und Anand Swaminathan: *Mining Email Social Networks.* In: Diehl, Stephan et al. [DGH06], Seiten 137–143.

[BHL+07] Basili, Victor R., Jens Heidrich, Mikael Lindvall, Jürgen Münch, Myrna Regardie, Dieter Rombach, Carolyn Seaman und Adam Trendowicz: *GQM+Strategies®: A Comprehensive Methodology for Aligning Business Strategies with Software Measurement.* In: Büren, Günter et al. [BBD07], Seiten 253–266.

[BJS+08] Bettenburg, Nicolas, Sascha Just, Adrian Schröter, Cathrin Weiss, Rahul Premraj und Thomas Zimmermann: *What Makes a Good Bug Report?* In: Harrold, Mary Jean und Gail C. Murphy (Herausgeber): *SIGSOFT FSE*, Seiten 308–318. ACM, 2008.

[BMB02] Briand, Lionel C., Sandro Morasca und Victor R. Basili: *An Operational Process for Goal-Driven Definition of Measures.* IEEE Trans. Softw. Eng., 28(12):1106–1125, 2002.

[BR88] Basili, Victor R. und H. Dieter Rombach: *The TAME Project: Towards Improvement-Oriented Software Environments.* IEEE Trans. Software Eng., 14(6):758–773, 1988.

[Bru09] Brunèliere, Hugo: *MoDisco Use Case Example „Bugzilla Metrics“.* http://www.eclipse.org/gmt/modisco/useCases/BugzillaMetrics, 2009. (Zugriff: 4. Januar 2010).

[BSM+03] Budinsky, Frank, Dave Steinberg, Ed Merks, Ray Ellersick und Timothy J. Grose: *Eclipse Modeling Framework.* Addison-Wesley, 2003.

[Bug09a] *The Bugzilla Guide – 3.4.1 Release.* http://www.bugzilla.org/docs/3.4/en/pdf/Bugzilla-Guide.pdf, 2009. (Zugriff: 4. Januar 2010).

[Bug09b] *BugzillaMetrics Guide*. http://bugzillametrics.org, 2009. (Zugriff: 4. Januar 2010).

[Buh04] Buhl, Axel: *Grundkurs Software-Projektmanagement*. Carl Hanser Verlag, München Wien, 2004.

[BW84] Basili, Victor R. und David M. Weiss: *A Methodology for Collecting Valid Software Engineering Data*. IEEE Trans. Software Eng., 10(6):728–738, 1984.

[BZL06] Breu, Silvia, Thomas Zimmermann und Christian Lindig: *HAM: Cross-Cutting Concerns in Eclipse*. In: Burke, Michael G., Alessandro Orso und Martin P. Robillard (Herausgeber): *Proceedings of the 2006 OOPSLA Workshop on Eclipse Technology eXchange, ETX 2006, Portland, Oregon, USA, October 22-23, 2006*, Seiten 21–24. ACM, 2006.

[CB99] Chulani, Sunita und Barry W. Boehm: *Modeling Software Defect Introduction and Removal: COQUALMO (COnstructive QUALity MOdel)*. Technischer Bericht USC-CSE-99-510, University of Southern California, Center for Software Engineering, 1999.

[CBC+92] Chillarege, Ram, Inderpal S. Bhandari, Jarir K. Chaar, Michael J. Halliday, Diane S. Moebus, Bonnie K. Ray und Man-Yuen Wong: *Orthogonal Defect Classification - A Concept for In-Process Measurements*. IEEE Trans. Software Eng., 18(11):943–956, 1992.

[CC05] Canfora, Gerardo und Luigi Cerulo: *Impact Analysis by Mining Software and Change Request Repositories*. In: *IEEE METRICS*, Seite 29. IEEE Computer Society, 2005.

[CFGQ04] Carvallo, Juan Pablo, Xavier Franch, Gemma Grau und Carme Quer: *QM: A Tool for Building Software Quality Models*. In: *12th IEEE International Conference on Requirements Engineering (RE 2004), 6-10 September 2004, Kyoto, Japan*, Seiten 358–359. IEEE Computer Society, 2004.

[CH96] Cogan, Boris I. und Robin B. Hunter: *Language-based Approaches to Software Measurement*. In: *IEEE METRICS*, Seiten 3–9. IEEE Computer Society, 1996.

[CK94] Chidamber, Shyam R. und Chris F. Kemerer: *A Metrics Suite for Object Oriented Design*. IEEE Trans. Software Eng., 20(6):476–493, 1994.

[CLO95] Coleman, Don M., Bruce Lowther und Paul W. Oman: *The Application of Software Maintainability Models in Industrial Software Systems*. Journal of Systems and Software, 29(1):3–16, 1995.

[CM78] Cavano, Joseph P. und James A. McCall: *A Framework for the Measurement of Software Quality.* In: *Proceedings of the Software Quality Assurance Workshop on Functional and Performance Issues*, Seiten 133–139, 1978.

[CMM06] *CMMI for Development, Version 1.2.* Technischer Bericht, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, 2006.

[CMSB05] Cubranic, Davor, Gail C. Murphy, Janice Singer und Kellogg S. Booth: *Hipikat: A Project Memory for Software Development.* IEEE Trans. Software Eng., 31(6):446–465, 2005.

[CN02] Clements, Paul und Linda Northrop: *Software Product Lines – Practices and Patterns.* Addison Wesley, 2002.

[Cov] *Code Cover, Version 1.0.0.1.* http://www.codecover.org. (Zugriff: 4. Januar 2010).

[Cro79] Crosby, Philip B.: *Quality Is Free: The Art of Making Quality Certain.* McGraw-Hill, 1979.

[CS08] Ciolkowski, Marcus und Martín Soto: *Towards a Comprehensive Approach for Assessing Open Source Projects.* In: Dumke, Reiner R. et al. [DBB+08], Seiten 316–330.

[CST07] Case, Gary, George Spalding und Sharon Taylor: *Continual Service Improvement.* TSO (The Stationery Office), 2007. Published for the Office of Government Commerce (OGC).

[CVW98] Cook, Jonathan E., Lawrence G. Votta und Alexander L. Wolf: *Cost-Effective Analysis of In-Place Software Processes.* IEEE Trans. Software Eng., 24(8):650–663, 1998.

[CWT07] Cannon, David, David Wheeldon und Sharon Taylor: *Service Operation.* TSO (The Stationery Office), 2007. Published for the Office of Government Commerce (OGC).

[DBB+08] Dumke, Reiner R., René Braungarten, Günter Büren, Alain Abran und Juan Jose Cuadrado-Gallego (Herausgeber): *Software Process and Product Measurement, International Conferences: IWSM 2008, Metrikon 2008, and Mensura 2008, Munich, Germany, November 18-19, 2008. Proceedings*, Band 5338 der Reihe *Lecture Notes in Computer Science.* Springer, 2008.

[Dek99] Dekkers, Carol A.: *The Secrets of Highly Successful Measurement Programs.* Cutter IT Journal, 12(4):29–35, 1999.

[DGH06] Diehl, Stephan, Harald Gall und Ahmed E. Hassan (Herausgeber): *Proceedings of the 2006 International Workshop on Mining Software Repositories, MSR 2006, Shanghai, China, May 22-23, 2006.* ACM, 2006.

[DGPH06] Diehl, Stephan, Harald Gall, Martin Pinzger und Ahmed E. Hassan: *Introduction to MSR 2006*. In: Diehl, Stephan et al. [DGH06], Seiten 1–2.

[DIN09] *DIN 69 901 – Projektmanagement – Projektmanagementsysteme*. DIN Deutsches Institut für Normung e.V., Berlin, 2009.

[DJH+08] Deissenboeck, Florian, Elmar Jürgens, Benjamin Hummel, Stefan Wagner, Benedikt Mas y Parareda und Markus Pizka: *Tool Support for Continuous Quality Control*. IEEE Software, 25(5):60–67, 2008.

[DJLW09] Deissenboeck, Florian, Elmar Juergens, Klaus Lochmann und Stefan Wagner: *Software Quality Models: Purposes, Usage Scenarios and Requirements*. In: *Proceedings of the 7th international workshop on Software Quality (WoSQ 09)*. IEEE CS Press, 2009.

[DP03] Draheim, Dirk und Lukasz Pekacki: *Process-Centric Analytical Processing of Version Control Data*. In: *6th International Workshop on Principles of Software Evolution (IWPSE 2003), 1-2 September 2003, Helsinki, Finland*, Seiten 131–. IEEE Computer Society, 2003.

[Dro95] Dromey, R. Geoff: *A Model for Software Product Quality*. IEEE Trans. Software Eng., 21(2):146–162, 1995.

[EBU] *Eclipse Bugzilla Usage*. http://wiki.eclipse.org/Development_Resources/HOWTO/Bugzilla_Use. (Zugriff: 4. Januar 2010).

[ED] *Eclipse Dash*. http://www.eclipse.org/dash/. (Zugriff: 4. Januar 2010).

[ED07] Ebert, Christof und Reiner Dumke: *Software Measurement*. dpunkt Verlag, Heidelberg, 2007.

[EEBF05] El Wakil, M., A. El Bastawissi, M. Boshra und A. Fahmy: *A Novel Approach to Formalize and Collect Object-Oriented Design-Metrics*. In: *Proceedings of the 9th International Conference on Empirical Assessment in Software Engineering*, 2005.

[EGM+06] Eichberg, Michael, Daniel Germanus, Mira Mezini, Lukas Mrokon und Thorsten Schäfer: *QScope: an Open, Extensible Framework for Measuring Software Projects*. In: *10th European Conference on Software Maintenance and Reengineering (CSMR 2006), 22-24 March 2006, Bari, Italy*, Seiten 113–122. IEEE Computer Society, 2006.

[EHLP07] Edgar, Nick, Kevin Haaland, Jin Li und Kimberley Peter: *Eclipse User Interface Guidelines*. http://wiki.eclipse.org/User_Interface_Guidelines, November 2007. (Zugriff: 4. Januar 2010).

[Fen94] Fenton, Norman: *Software Measurement: A Necessary Scientific Basis.* IEEE Trans. Softw. Eng., 20(3):199–206, 1994.

[FFB02] Fey, Dániel, Róbert Fajta und András Boros: *Feature Modeling: A Meta-Model to Enhance Usability and Usefulness.* In: Chastek, Gary J. (Herausgeber): *SPLC*, Band 2379 der Reihe *Lecture Notes in Computer Science*, Seiten 198–216. Springer, 2002.

[FGR+97] Foreman, John, Jon Gross, Robert Rosenstein, David A. Fisher und Kimberly Brune: *C4 Software Technology Reference Guide - A Prototype.* Technischer Bericht CMU/SEI-97-HB-001, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, 1997.

[Fly06] Flyvbjerg, Bent: *Five Misunderstandings About Case-Study Research.* Qualitative Inquiry, 12(2):219–245, April 2006.

[FM08] Francalanci, Chiara und Francesco Merlo: *Empirical Analysis of the Bug Fixing Process in Open Source Projects.* In: Russo, Barbara et al. [RDH+08], Seiten 187–196.

[FP97] Fenton, Norman E. und Shari Lawrence Pfleeger: *Software Metrics - A Rigorous and Practical Approach.* IEEE Press, 2 Auflage, 1997.

[Gam05] Gamma, Erich: *Agile, Open Source, Distributed, and On-Time – Inside the Eclipse Development Process.* Keynote Talk, International Conference of Software Engineering (ICSE), St.Louis, Missouri, 2005.

[GC87] Grady, Robert B. und Deborah L. Caswell: *Software Metrics: Establishing a Company-Wide Program.* Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1987.

[GEF] *Graphical Editing Framework.* http://www.eclipse.org/gef/. (Zugriff: 4. Januar 2010).

[Gli05] Glinz, Martin: *Einführung in die Modelltheorie.* Vorlesungsskript Informatik II, Universität Zürich, 2005.

[GLZ07] Gall, Harald, Michele Lanza und Thomas Zimmermann (Herausgeber): *Fourth International Workshop on Mining Software Repositories, MSR 2007 (ICSE Workshop), Minneapolis, MN, USA, May 19-20, 2007, Proceedings.* IEEE Computer Society, 2007.

[GM97] Gray, Andrew und Stephen G. Macdonell: *GQM++ A Full Life Cycle Framework for the Development and Implementation of Software Metric Programs.* In: *Australian Software Measurement Conference, Canberra, November 1997*, Seiten 22–35, 1997.

[GMF] *Graphical Modeling Framework.* http://www.eclipse.org/gmf/. (Zugriff: 4. Januar 2010).

[Gol04] Golden, Bernard: *Succeeding with Open Source (Addison-Wesley Information Technology Series).* Addison-Wesley Professional, 2004.

[GP96] Green, Thomas R. G. und Marian Petre: *Usability Analysis of Visual Programming Environments: A 'Cognitive Dimensions' Framework.* J. Vis. Lang. Comput., 7(2):131–174, 1996.

[GR03] Gasser, Les und Gabriel Ripoche: *Distributed Collective Practices and F/OSS Problem Management: Perspective and Methods.* In: *Conference on Cooperation, Innovation & Technologie, (CITE2003).* University de Technologie de Troyes, France, December 2003.

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

[GS09a] Gousios, Georgios und Diomidis Spinellis: *Alitheia Core: An Extensible Software Quality Monitoring Platform.* In: *ICSE* [ICS09], Seiten 579–582.

[GS09b] Gousios, Georgios und Diomidis Spinellis: *A Platform for Software Engineering Research.* In: Godfrey, Michael W. und Jim Whitehead [GW09], Seiten 31–40.

[GSCL+07] García, Francisco, Manuel A. Serrano, José A. Cruz-Lemus, Francisco Ruiz und Mario Piattini: *Managing Software Process Measurement: A Metamodel-Based Approach.* Inf. Sci., 177(12):2570–2586, 2007.

[GSL07a] Grammel, Lars, Holger Schackmann und Horst Lichter: *BugzillaMetrics - Design of an Adaptable Tool for Evaluating User-Defined Metric Specification on Change Requests.* In: Büren, Günter et al. [BBD07], Seiten 201–218.

[GSL07b] Grammel, Lars, Holger Schackmann und Horst Lichter: *BugzillaMetrics: An Adaptable Tool for Evaluating Metric Specifications on Change Requests.* In: Penta, Massimiliano Di und Michele Lanza (Herausgeber): *9th International Workshop on Principles of Software Evolution (IWPSE 2007), in Conjunction with the 6th ESEC/FSE Joint Meeting, Dubrovnik, Croatia, September 3-4, 2007*, Seiten 35–38. ACM, 2007.

[GST+] Grammel, Lars, Adrian Schröter, Christoph Treude, Holger Schackmann und Margaret-Anne Storey: *Attracting the Community's Many Eyes: An Exploration of Non-Developer Involvement in Issue Tracking in Open Source and Open Commercial Development.* to appear.

[GT00] Godfrey, Michael W. und Qiang Tu: *Evolution in Open Source Software: A Case Study.* In: *ICSM '00: Proceedings of the International Conference on Software Maintenance (ICSM'00)*, Seite 131, Washington, DC, USA, 2000. IEEE Computer Society.

[GW09] Godfrey, Michael W. und Jim Whitehead (Herausgeber): *6th IEEE Intl. Working Conference on Mining Software Repositories (MSR-09), Vancouver, Canada, May 16-17, 2009, Proceedings.* IEEE, 2009.

[GWT] *Google Web Toolkit.* http://code.google.com/webtoolkit/. (Zugriff: 4. Januar 2010).

[HALS08] Habra, Naji, Alain Abran, Miguel Lopez und Asma Sellami: *A Framework for the Design and Verification of Software Measurement Methods.* Journal of Systems and Software, 81(5):633 – 648, 2008. Software Process and Product Measurement.

[HDHM06] Hörmann, Klaus, Lars Dittmann, Bernd Hindel und Markus Müller: *SPICE in der Praxis.* dpunkt Verlag, Heidelberg, 2006.

[HF97] Hall, Tracy und Norman Fenton: *Implementing Effective Software Metrics Programs.* IEEE Software, 14(2):55–65, 1997.

[HG98] Herbsleb, James D. und Rebecca E. Grinter: *Conceptual Simplicity Meets Organizational Complexity: Case Study of a Corporate Metrics Program.* In: *Proceedings of the 1998 International Conference on Software Engineering, ICSE 98, April 19-25, 1998, Kyoto, Japan.*, Seiten 271–280, 1998.

[HGBR07] Herraiz, Israel, Jesús M. González-Barahona und Gregorio Robles: *Forecasting the Number of Changes in Eclipse Using Time Series Analysis.* In: Gall, Harald et al. [GLZ07], Seite 32.

[HGGBR08] Herraiz, Israel, Daniel M. Germán, Jesús M. González-Barahona und Gregorio Robles: *Towards a Simplification of the Bug Report Form in Eclipse.* In: Hassan, Ahmed E. et al. [HLG08], Seiten 145–148.

[HHMS09] Hindel, Bernd, Klaus Hörmann, Markus Müller und Jürgen Schmied: *Basiswissen Software-Projektmanagement.* dpunkt Verlag, Heidelberg, 2009.

[HK06] Han, Jiawei und Micheline Kamber: *Data Mining – Concepts and Techniques.* Morgan Kaufmann, San Francisco, CA, 2006.

[HLG08] Hassan, Ahmed E., Michele Lanza und Michael W. Godfrey (Herausgeber): *Fifth International Workshop on Mining Software Repositories, MSR 2008 (ICSE Workshop), Leipzig, Germany, May 10-11, 2008, Proceedings.* ACM, 2008.

[HMO08] Harjumaa, Lasse, Jouni Markkula und Markku Oivo: *How Does a Measurement Programme Evolve in Software Organizations?* In: *PROFES '08: Proceedings of the 9th International Conference on Product-Focused Software Process Improvement*, Seiten 230–243, Berlin, Heidelberg, 2008. Springer-Verlag.

[Hor03] Horváth, Peter: *Controlling*. Vahlen, München, 2003.

[Hou07] Hou, Daqing: *Studying the Evolution of the Eclipse Java Editor*. In: Cheng, Li-Te, Alessandro Orso und Martin P. Robillard (Herausgeber): *Proceedings of the 2007 OOPSLA Workshop on Eclipse Technology eXchange, ETX 2007, Montreal, Quebec, Canada, October 21, 2007*, Seiten 65–69. ACM, 2007.

[HPvD09] Hermans, Felienne, Martin Pinzger und Arie van Deursen: *Domain-Specific Languages in Practice: A User Study on the Success Factors*. In: *Proceedings ACM/IEEE 12th International Conference on Model Driven Engineering Languages and Systems (MODELS) — Empirical Track*, Lecture Notes in Computer Science. Springer-Verlag, 2009.

[HS87] Humphrey, Watts S. und William L. Sweet: *A Method for Assessing the Software Engineering Capability of Contractors*. Technical Report CMU/SEI-87-TR-23, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, 1987.

[Hub96] Hubert, Kupper: *Zur Kunst der Projektsteuerung*. Oldenburg Wissenschaftsverlag, München, 1996.

[IBM08] IBM Cooperation: *Telelogic Logiscope – Basic Concepts – Version 6.5*, November 2008.

[ICS08] *First International Conference on Software Testing, Verification, and Validation, ICST 2008, Lillehammer, Norway, April 9-11, 2008*. IEEE Computer Society, 2008.

[ICS09] *31st International Conference on Software Engineering, ICSE 2009, May 16-24, 2009, Vancouver, Canada, Proceedings*. IEEE, 2009.

[IEE90] IEEE: *Standard 610 – Glossary of Software Engineering Terminology (IEEE Std 610.12-1990)*. IEEE Press, New York http://standards.ieee.org/reading/ieee/std/se/610.12-1990.pdf, 1990. (Zugriff: 4. Januar 2010).

[IEE98] *IEEE Standard for a Software Quality Metrics Methodology*. IEEE Std 1061-1998, Dec 1998.

[IGC02] IGC International Group of Controlling: *Controller Leitbild der IGC International Group of Controlling*. http://www.controllerverein.de, 2002. (Zugriff: 4. Januar 2010).

[IM03] Iversen, Jakob und Lars Mathiassen: *Cultivation and Engineering of a Software Metrics Program.* Inf. Syst. J., 13(1):3–20, 2003.

[INT07] Iqbal, Majid, Michael Nieves und Sharon Taylor: *Service Strategy.* TSO (The Stationery Office), 2007. Published for the Office of Government Commerce (OGC).

[ISO95] *ISO/IEC 12207 – Information Technology - Software Life Cycle Processes.* International Organization for Standardization (ISO) / International Electrotechnical Commision (IEC), Genf, 1995.

[ISO01] *ISO/IEC 9126 – Software Engineering – Product Quality; Parts 1–4.* International Organization for Standardization (ISO) / International Electrotechnical Commision (IEC), Genf, 2001.

[ISO04] *ISO/IEC 15504 – Information Technology – Process Assessment.* International Organization for Standardization (ISO) / International Electrotechnical Commision (IEC), Genf, 2004.

[ISO05a] *ISO/IEC 20000 – Information Technology – Service Management.* International Organization for Standardization (ISO) / International Electrotechnical Commision (IEC), Genf, 2005.

[ISO05b] *ISO/IEC 25000 – Software Engineering – Software product Quality Requirements and Evaluation (SQuaRE) – Guide to SQuaRE.* International Organization for Standardization (ISO) / International Electrotechnical Commision (IEC), Genf, 2005.

[ISO07a] *ISO/IEC 14143 – Information technology – Software measurement – Functional size measurement.* International Organization for Standardization (ISO) / International Electrotechnical Commision (IEC), Genf, 2007.

[ISO07b] *ISO/IEC 15939 – Systems and Software Engineering – Measurement Process.* International Organization for Standardization (ISO) / International Electrotechnical Commision (IEC), Genf, 2007.

[ISO07c] *ISO/IEC Guide 99:2007-12 – International vocabulary of metrology - Basic and general concepts and associated terms (VIM).* International Organization for Standardization (ISO) / International Electrotechnical Commision (IEC), Genf, 2007.

[IT 03] IT Governance Institute: *IT Governance für Geschäftsführer und Vorstände, 2.Ausgabe.* http://www.isaca.org, 2003. (Zugriff: 4. Januar 2010).

[IT 07] IT Governance Institute: *COBIT 4.1.* http://www.isaca.org, 2007. (Zugriff: 4. Januar 2010).

[JA98] Jacquet, Jean-Philippe und Alain Abran: *Metric Validation Proposals: A Structured Analysis.* In: *8th International Workshop on Software Measurement*, 1998.

[Jan09] Jansen, Martin: *Entwicklung eines generischen Qualitätsmodell-Editors und Auswertungswerkzeuges.* Diplomarbeit, RWTH Aachen, Januar 2009.

[Jon08] Jones, Capers: *Applied Software Measurement: Global Analysis of Productivity and Quality.* Mcgraw-Hill Professional, 2008.

[KA08] Kanat-Alexander, Max: *The Bugzilla Survery.* http://wiki.mozilla.org/Bugzilla:Survey, August 2008. (Zugriff: 4. Januar 2010).

[KCH+90] Kang, Kyo, Sholom Cohen, James Hess, William Novak und A. S. Peterson: *Feature-Oriented Domain Analysis (FODA) Feasibility Study.* Technical Report CMU/SEI-90-TR-21, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, 1990.

[KCM07] Kagdi, Huzefa H., Michael L. Collard und Jonathan I. Maletic: *A Survey and Taxonomy of Approaches for Mining Software Repositories in the Context of Software Evolution.* Journal of Software Maintenance, 19(2):77–131, 2007.

[KG07] Kidane, Yared und Peter Gloor: *Correlating Temporal Communication Patterns of the Eclipse Open Source Community with Performance and Creativity.* Computational & Mathematical Organization Theory, 13(1):17 – 27, March 2007.

[KH05] Khaddaj, Souheil und Gerard Horgan: *A Proposed Adaptable Quality Model for Software Quality Assurance.* Journal of Computer Sciences 1 (4), Seiten 482–487, 2005.

[KHL01] Kitchenham, Barbara A., Robert T. Hughes und Stephen G. Linkman: *Modeling Software Measurement Data.* IEEE Trans. Softw. Eng., 27(9):788–804, 2001.

[KKJC06] Kitchenham, Barbara, Cat Kutay, D. Ross Jeffery und Colin Connaughton: *Lessons Learnt from the Analysis of Large-Scale Corporate Databases.* In: Osterweil, Leon J., H. Dieter Rombach und Mary Lou Soffa (Herausgeber): *28th International Conference on Software Engineering (ICSE 2006), Shanghai, China, May 20-28, 2006*, Seiten 439–444. ACM, 2006.

[KKL+98] Kang, Kyo Chul, Sajoong Kim, Jaejoon Lee, Kijoo Kim, Euiseob Shin und Moonhang Huh: *FORM: A Feature-Oriented Reuse Method with Domain-Specific Reference Architectures.* Ann. Software Eng., 5:143–168, 1998.

[Kle08] Kleppe, Anneke: *Software Language Engineering: Creating Domain-Specific Languages Using Metamodels.* Addison-Wesley Longman, Amsterdam, 2008.

[KLPN97] Kitchenham, Barbara, Steve Linkman, Alberto Pasquini und Vincenzo Nanni: *The SQUID Approach to Defining a Quality Model.* Software Quality Control, 6(3):211–233, 1997.

[KM02] Kajko-Mattsson, M.: *Corrective Maintenance Maturity Model: Problem Management.* Software Maintenance, IEEE International Conference on, 0:0486, 2002.

[Knu68] Knuth, Donald E.: *Semantics of Context-Free Languages.* Theory of Computing Systems, 2(2):127–145, June 1968.

[Kop06] Koponen, Timo: *RaSOSS - Remote Analysis System for Open Source Software.* In: *Proceedings of the International Conference on Software Engineering Advances (ICSEA 2006), October 28 - November 2, 2006, Papeete, Tahiti, French Polynesia*, Seiten 54–59. IEEE Computer Society, 2006.

[KPF95] Kitchenham, Barbara, Shari Lawrence Pfleeger und Norman E. Fenton: *Towards a Framework for Software Measurement Validation.* IEEE Trans. Software Eng., 21(12):929–943, 1995.

[Kri88] Kriz, Jürgen: *Facts and Artefacts in Social Science: An Ephistemological and Methodological Analysis of Empirical Social Science.* McGraw Hill Research, 1988.

[KS02] Koch, Stefan und Georg Schneider: *Effort, Co-operation and Co-ordination in an Open Source Software Project: GNOME.* Information Systems Journal, 12(1):27–42, 2002.

[Küt05] Kütz, Martin: *IT-Controlling für die Praxis.* dpunkt Verlag, Heidelberg, 2005.

[Küt06] Kütz, Martin: *Kennzahlen in der IT.* dpunkt Verlag, Heidelberg, 2006.

[Lis08] Lischkowitz, Christoph: *Erweiterung von BugzillaMetrics zur Auswertung von Daten aus Versionskontrollsystemen.* Diplomarbeit, RWTH Aachen, December 2008.

[LK03] Lawler, Jim und Barbara Kitchenham: *Measurement Modeling Technology.* IEEE Softw., 20(3):68–75, 2003.

[LL07] Ludewig, Jochen und Horst Lichter: *Software Engineering.* dpunkt Verlag, Heidelberg, 2007.

[LLL08] Lincke, Rüdiger, Jonas Lundberg und Welf Löwe: *Comparing Software Metric Tools.* In: *ISSTA '08: Proceedings of the 2008 International Symposium on Software Testing and Analysis*, Seiten 131–142, New York, NY, USA, 2008. ACM.

[LMT07] Lacy, Shirley, Ivor MacFarlane und Sharon Taylor: *Service Transition.* TSO (The Stationery Office), 2007. Published for the Office of Government Commerce (OGC).

[LP76] Lehman, Meir M. und F. N. Parr: *Program Evolution and its Impact on Software Engineering.* In: *ICSE '76: Proceedings of the 2nd International Conference on Software Engineering*, Seiten 350–357, Los Alamitos, CA, USA, 1976. IEEE Computer Society Press.

[LRB+07] Linstead, Erik, Paul Rigor, Sushil Krishna Bajracharya, Cristina Videira Lopes und Pierre Baldi: *Mining Eclipse Developer Contributions via Author-Topic Models.* In: Gall, Harald et al. [GLZ07], Seite 30.

[LRT07] Lloyd, Vernon, Colin Rudd und Sharon Taylor: *Service Design.* TSO (The Stationery Office), 2007. Published for the Office of Government Commerce (OGC).

[Lyu96] Lyu, Michael R. (Herausgeber): *Handbook of Software Reliability Engineering.* IEEE Computer Society Press and McGraw-Hill, 1996.

[Mac93] Macdonell, Stephen: *Deriving Relevant Functional Measures for Automated Development Projects.* Information and Software Technology, 35:499–512, 1993.

[Mar94] Martin, Robert: *OO Design Quality Metrics - An Analysis of Dependencies.* http://www.objectmentor.com/resources/articles/oodmetrc.pdf, 1994. (Zugriff: 4. Januar 2010).

[Mar06] Martin, Robert C.: *Agile Software Development: Principles, Patterns, and Practices in C#.* Prentice Hall, 2006.

[MBW+97] Morasca, Sandro, Lionel C. Briand, Elaine J. Weyuker, Victor R. Basili und Marvin V. Zelkowitz: *Comments on „Towards a Framework for Software Measurement Validation“.* IEEE Trans. Softw. Eng., 23(3):187–188, 1997.

[McC76] McCabe, Thomas J.: *A Complexity Measure.* IEEE Trans. Software Eng., 2(4):308–320, 1976.

[MD96] Morris, Pam und Jules Desharnais: *Function Point Analysis. Validating the Results.* In: *IFPUG Spring Conference*, April 1996.

[Mes07] Meszaros, Gerard: *XUnit Test Patterns: Refactoring Test Code.* Addison-Wesley, 2007.

[MFH02] Mockus, Audris, Roy T. Fielding und James D. Herbsleb: *Two Case Studies of Open Source Software Development: Apache and Mozilla.* ACM Trans. Softw. Eng. Methodol., 11(3):309–346, 2002.

[MIS95] MISRA: *Software Metrics.* Technischer Bericht 5, The Motor Industry Software Reliability Association, Februar 1995.

[MJCH08] Monperrus, Martin, Jean-Marc Jézéquel, Joël Champeau und Brigitte Hoeltzener: *A Model-Driven Measurement Approach.* In: *MoDELS '08: Proceedings of the 11th International Conference on Model Driven Engineering Languages and Systems*, Seiten 505–519, Berlin, Heidelberg, 2008. Springer-Verlag.

[ML02] Mens, Tom und Michele Lanza: *A Graph-Based Metamodel for Object-Oriented Software Metrics.* Electr. Notes Theor. Comput. Sci., 72(2), 2002.

[MMG05] Marinescu, Christina, Radu Marinescu und Tudor Girba: *Towards a Simplified Implementation of Object-Oriented Design Metrics.* In: *Software Metrics, 2005. 11th IEEE International Symposium*, Seiten 10 pp.–11, Sept. 2005.

[MoD] *MoDisco.* http://www.eclipse.org/gmt/modisco. (Zugriff: 4. Januar 2010).

[MP08] McQuillan, Jacqueline und James Power: *A Metamodel for the Measurement of Object-Oriented Systems: An Analysis using Alloy.* International Conference on Software Testing, Verification, and Validation 2008, 0:288–297, 2008.

[MR05] Makris, Kristis und Kyung Dong Ryu: *Scmbug: Policy-based Integration of Software Configuration Management with Bug-tracking.* In: *USENIX Annual Technical Conference, FREENIX Track*, Seiten 11–22. USENIX, 2005.

[MS03] Messerschmidt, Hartmut und Kai Schweinsberg: *OLAP mit dem SQL-Server.* dpunkt Verlag, Heidelberg, 2003.

[MS06] Michlmayr, Martin und Anthony Senyard: *A Statistical Analysis of Defects in Debian and Strategies for Improving Quality in Free Software Projects.* In: Bitzer, Jürgen und Philipp J. H. Schröder (Herausgeber): *The Economics of Open Source Software Development*, Seiten 131–148, Amsterdam, The Netherlands, 2006. Elsevier.

[MV00] Mockus, Audris und Lawrence G. Votta: *Identifying Reasons for Software Changes using Historic Databases.* In: *International Conference on Software Maintenance (ICSM'00), 11-14 October 2000, San Jose, California*, Seiten 120–130. IEEE Computer Society, 2000.

[NAS] *NASA Metrics Data Program.* http://mdp.ivv.nasa.gov/complexity_metrics.html#Cyclo. (Zugriff: 4. Januar 2010).

[NAS95] NASA Software Program: *Software Measurement Guidebook.* Technischer Bericht, National Aeronautics and Space Administration, 1995.

[Nor83] Norman, Donald A.: *Some Observations on Mental Models.* In: Gentner, D. und A. L. Stevens (Herausgeber): *Mental Models*, Seiten 7–14. Hillsdale, N J., 1983.

[NvV01] Niessink, Frank und Hans van Vliet: *Measurement Program Success Factors Revisited.* Information & Software Technology, 43(10):617–628, 2001.

[Nyß09] Nyssen, Alexander: *Model-Based Construction of Embedded & Real-Time Software - A Methodology for Small Devices.* Dissertation, RWTH Aachen. Aachener Informatik Berichte, AIB 2009-03, 2009.

[OAH03] Orso, Alessandro, Taweesup Apiwattanapong und Mary Jean Harrold: *Leveraging Field Data for Impact Analysis and Regression Testing.* In: *ESEC/FSE-11: Proceedings of the 9th European Software Engineering Conference held jointly with 11th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, Seiten 128–137, New York, NY, USA, 2003. ACM.

[Oba07] Obaid, Malek: *Development of a Query and Reporting Interface for a Change Request Analysis Tool.* Masterarbeit, RWTH Aachen, December 2007.

[OMG06] OMG: *Object Constraint Language Specification, version 2.0.* http://www.omg.org/technology/documents/formal/ocl.htm, 2006. (Zugriff: 4. Januar 2010).

[OMG07] OMG: *MOF 2.0 / XMI Mapping Specification, v2.1.1.* http://www.omg.org/technology/documents/formal/xmi.htm, 2007. (Zugriff: 4. Januar 2010).

[OMG09] OMG: *Architecture-Driven Modernization (ADM): Software Metrics Meta-Model (SMM), FTF-Beta1.* http://www.omg.org/docs/ptc/09-03-03.pdf, 2009. (Zugriff: 4. Januar 2010).

[Ope05] OpenBRR.org: *Business Readiness Rating for Open Source: A Proposed Open Standard to Facilitate Assessment and Adoption of Open Source Software.* http://www.openbrr.org, 2005. (Zugriff: 4. Januar 2010).

[Osb97] Osborne, M.: *Mix or Mash the Dashboard Metaphor.* American Programmer, 10(11), November 1997.

[OW07] Ostrand, Thomas J. und Elaine J. Weyuker: *An Industrial Research Program in Software Fault Prediction.* In: Bleek, Wolf-Gideon, Henning Schwentner und Heinz Züllighoven (Herausgeber): *Software Engineering (Workshops)*, Band 106 der Reihe *LNI*, Seiten 21–28. GI, 2007.

[OW08] Oestereich, Bernd und Christian Weiss: *APM – Agiles Projektmanagement.* dpunkt Verlag, Heidelberg, 2008.

[OWB05] Ostrand, Thomas J., Elaine J. Weyuker und Robert M. Bell: *Predicting the Location and Number of Faults in Large Software Systems.* IEEE Trans. Software Eng., 31(4):340–355, 2005.

[Oxy] *SyncRO Soft <oXygen/>, Schema Editor.* http://www.oxygenxml.com/xml_schema_editor.html. (Zugriff: 4. Januar 2010).

[Pan07] Panjer, Lucas D.: *Predicting Eclipse Bug Lifetimes.* In: Gall, Harald et al. [GLZ07], Seite 29.

[PCC91] Paulk, Mark C., Bill Curtis und Mary Beth Chrissis: *Capability Maturity Model for Software.* Technical Report CMU/SEI-91-TR-24, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, 1991.

[Pfl93] Pfleeger, Shari Lawrence: *Lessons Learned in Building a Corporate Metrics Program.* IEEE Software, 10(3):67–74, 1993.

[PGF96] Park, Robert E., Wolfhart B. Goethert und William A. Florac: *Goal-Driven Software Measurement - A Guidebook.* Technischer Bericht CMU/SEI-96-HB-002, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, 1996.

[PGK+09] Plösch, Reinhold, Harald Gruber, Christian Körner, Gustav Pomberger und Stefan Schiffer: *A Proposal for a Quality Model Based on a Technical Topic Classification.* In: *2. Workshop zur Software-Qualitätsmodellierung und -bewertung (SQMB '09)*, Seiten 10–18, 2009.

[PGP+08] Plösch, Reinhold, Harald Gruber, Gustav Pomberger, Matthias Saft und Stefan Schiffer: *Tool Support for Expert-Centred Code Assessments.* In: *ICST* [ICS08], Seiten 258–267.

[PMdCH08] Pereira, Maria João Varanda, Marjan Mernik, Daniela da Cruz und Pedro Rangel Henriques: *Program Comprehension for Domain-Specific Languages.* ComSIS – Computer Science an Information Systems Journal, Special Issue on Compilers, Related Technologies and Applications, 5(2):1–17, Dec 2008.

[Pop80] Popper, Karl: *The Logic of Scientific Discovery.* Hutchison, 1980.

[QMT09] *QMetric Quality Model Tools Guide.* http://qmetric.org, 2009. (Zugriff: 4. Januar 2010).

[RDH+08] Russo, Barbara, Ernesto Damiani, Scott A. Hissam, Björn Lundell und Giancarlo Succi (Herausgeber): *Open Source Development, Communities and Quality, IFIP 20th World Computer Congress, Working Group 2.3 on Open Source Software, OSS 2008, September 7-10, 2008, Milano, Italy*, Band 275 der Reihe *IFIP*. Springer, 2008.

[Rei01] Reissing, Ralf: *Towards a Model for Object-Oriented Design Measurement.* In: *In ECOOP Workshop on Quantative Approaches in Object-Oriented Software Engineering*, Seiten 71–84, 2001.

[RH68] Rubey, Raymond J. und R. Dean Hartwick: *Quantitative Measurement of Program Quality.* In: *Proceedings of the 1968 23rd ACM National Conference*, Seiten 671–677, New York, NY, USA, 1968. ACM.

[RH09] Runeson, Per und Martin Höst: *Guidelines for Conducting and Reporting Case Study Research in Software Engineering.* Empirical Software Engineering, 14(2):131–164, 2009.

[RHMP85] Radice, Ronald A., John T. Harding, Paul E. Munnis und Richard W. Phillips: *A Programming Process Study.* IBM Systems Journal, 24(2):91–101, 1985.

[Ric09] Richlofsky, Georg: *Analyse von Softwareentwicklungs- und Wartungsprozessen auf Basis von Change Request Daten.* Diplomarbeit, RWTH Aachen, Juni 2009.

[Roc94] Roche, John M.: *Software Metrics and Measurement Principles.* SIGSOFT Softw. Eng. Notes, 19(1):77–85, 1994.

[RSG99] Rosenberg, Linda, Ruth Stapko und Albert Gallo: *Applying Object-Oriented Metrics.* In: *Sixth International Symposium on Sofware Metrics, Boca Raton, Florida, November 4-6*, 1999.

[RW05] Rech, Joerg und Sebastian Weber: *Automatische Software-Qualitätsanalyse: Freie Werkzeuge zur Ermittlung von Software Produktmetriken und Qualitätsdefekten.* Technischer Bericht 108.05/D, Fraunhofer IESE, 2005.

[Sch04] Schackmann, Holger: *Entwicklung eines grafischen Feature-Editors für das Produktlinien-Toolset RequiLine.* Diplomarbeit, RWTH Aachen, Juni 2004.

[Sch05] Schuh, Günther: *Produktkomplexität managen, Strategien – Methoden – Tools.* Carl Hanser, 2005.

[Sch07] Schneider, Kurt: *Abenteuer Softwarequalität – Grundlagen und Verfahren für Qualitätssicherung und Qualitätsmanagement.* dpunkt Verlag, 2007.

[Sch09] Schaefer, Henning: *Bewertung der Prozessqualität von Open Source-Entwicklungsprozessen anhand von Software-Prozessdaten.* Diplomarbeit, RWTH Aachen, Januar 2009.

[SE96] Stienen, Hans und Franz Engelmann: *Die BOOTSTRAP-Methode zur Bewertung und Verbesserung der Software Entwicklung.* Wirtschaftsinformatik, 6:609–624, 1996.

[SGR04] Sandusky, Robert J., Les Gasser und Gabriel Ripoche: *Bug Report Networks: Varieties, Strategies, and Impacts in an OSS Development Community.* In: *Proceedings of the ICSE Workshop on Mining Software Repositories*, Seiten 80–84. ACM, 2004.

[SGSS08] Samoladas, Ioannis, Georgios Gousios, Diomidis Spinellis und Ioannis Stamelos: *The SQO-OSS Quality Model: Measurement Based Open Source Software Evaluation.* In: Russo, Barbara et al. [RDH+08], Seiten 237–248.

[SHT05] Sneed, Harry M., Martin Hasitschka und Maria-Therese Teichmann: *Software-Produktmanagement.* dpunkt Verlag, Heidelberg, 2005.

[SJH09] Shihab, Emad, Zhen Ming Jiang und Ahmed E. Hassan: *On the Use of IRC Channels by Developers of the GNOME GTK+ Open Source Project.* In: Godfrey, Michael W. und Jim Whitehead [GW09], Seiten 107–110.

[SL97] Simon, Frank und Claus Lewerentz: *Integration of an Object-Oriented Metrics Tool into SNiFF+.* Technischer Bericht I-22/1997, Brandenburgische Technische Universität Cottbus, 1997.

[SL08] Schackmann, Holger und Horst Lichter: *Comparison of Process Quality Characteristics Based on Change Request Data.* In: Dumke, Reiner R. et al. [DBB+08], Seiten 127–140.

[SL09] Schackmann, Holger und Horst Lichter: *Evaluating Process Quality in GNOME based on Change Request Data.* In: Godfrey, Michael W. und Jim Whitehead [GW09], Seiten 95 – 98.

[SMB08] *Guide to the Software Engineering Body of Knowledge – Supplemental Knowledge Areas – Software Measurement (Draft)*, Februar 2008.

[SMN09] Staron, Miroslaw, Wilhelm Meding und Christer Nilsson: *A Framework for Developing Measurement Systems and its Industrial Evaluation.* Inf. Softw. Technol., 51(4):721–737, 2009.

[Som07] Sommerville, Ian: *Software Engineering.* Pearson Education, 8 Auflage, 2007.

[Son] *SonarJ Architecture Management Solution for Java, Version 4.1.0 [Build 327].* http://www.hello2morrow.com/products/sonarj. (Zugriff: 4. Januar 2010).

[SQR] *Software Quality Reports for Jira/Bugzilla, Version 0.8.25.* https://sourceforge.net/projects/qareports/. (Zugriff: 4. Januar 2010).

[SS01] Schlittgen, Rainer und Bernd Streitberg: *Zeitreihenanalyse.* Oldenburg, 9 Auflage, 2001.

[SSL09] Schackmann, Holger, Henning Schaefer und Horst Lichter: *Evaluating Process Quality Based on Change Request Data - An Empirical Study of the Eclipse Project.* In: Abran, Alain, René Braungarten, Reiner R. Dumke, Juan Jose Cuadrado-Gallego und Jacob Brunekreef (Herausgeber): *IWSM/Mensura*, Band 5891 der Reihe *Lecture Notes in Computer Science*, Seiten 227–241. Springer, 2009.

[SSM06] Simon, Frank, Olaf Seng und Thomas Mohaupt: *Code-Quality-Management – Technische Qualität industrieller Softwaresysteme transparent und vergleichbar gemacht.* dpunkt Verlag, Heidelberg, 2006.

[SSSV04] Scotto, Marco, Alberto Sillitti, Giancarlo Succi und Tullio Vernazza: *A Relational Approach to Software Metrics.* In: *SAC '04: Proceedings of the 2004 ACM symposium on Applied computing*, Seiten 1536–1540, New York, NY, USA, 2004. ACM.

[Sta73] Stachowiak, Herbert: *Allgemeine Modelltheorie.* Springer, 1973.

[Sty] *Stylus Studio, XML Schema Editor.* http://www.stylusstudio.com/xml_schema_editor.html. (Zugriff: 4. Januar 2010).

[Sun99] Sun Microsystems: *Code Conventions for the Java Programming Language.* http://java.sun.com/docs/codeconv/, 1999. (Zugriff: 4. Januar 2010).

[SvGB05] Svahnberg, Mikael, Jilles van Gurp und Jan Bosch: *A Taxonomy of Variability Realization Techniques: Research Articles.* Softw. Pract. Exper., 35(8):705–754, 2005.

[SZZ05] Sliwerski, Jacek, Thomas Zimmermann und Andreas Zeller: *When Do Changes Induce Fixes?* In: *Proceedings of the 2005 International Workshop on Mining Software Repositories, MSR 2005, Saint Louis, Missouri, USA, May 17, 2005.* ACM, 2005.

[Tesa] *TestLink.* http://www.teamst.org. (Zugriff: 4. Januar 2010).

[Tesb] *Testopia.* http://www.mozilla.org/projects/testopia. (Zugriff: 4. Januar 2010).

[Tid05] Tidwell, Jenifer: *Designing Interfaces.* O'Reilly, 2005.

[TMK+06] Tsunoda, Masateru, Akito Monden, Takeshi Kakimoto, Yasutaka Kamei und Ken-ichi Matsumoto: *Analyzing OSS Developers' Working Time Using Mailing Lists Archives.* In: *MSR '06: Proceedings of the 2006 International Workshop on Mining Software Repositories*, Seiten 181–182, New York, NY, USA, 2006. ACM.

[TMY+06] Tsunoda, Masateru, Akito Monden, Hiroshi Yadohisa, Nahomi Kikuchi und Ken-ichi Matsumoto: *Productivity Analysis of Japanese Enterprise Software Development Projects.* In: *MSR '06: Proceedings of the 2006 International Workshop on Mining Software Repositories*, Seiten 14–17, New York, NY, USA, 2006. ACM.

[vB05] Bon, Jan van: *IT Service Management basierend auf ITIL.* Van Haren Publishing, 2005.

[vdM07] Massen, Thomas von der: *Feature-basierte Modellierung und Analyse von Variabilität in Produktlinienanforderungen.* Dissertation, RWTH Aachen. Shaker Verlag, 2007.

[ViP] *ViPER - Visual Tooling Platform for Model-Based Engineering.* http://www.viper.sc. (Zugriff: 4. Januar 2010).

[Völ08] Völter, Markus: *MD* Best Practices.* http://www.voelter.de, Dezember 2008. (Zugriff: 4. Januar 2010).

[vSB99] Solingen, Rini van und Egon Berghout: *The Goal/Question/Metric Method.* McGraw-Hill Education, 1999.

[VXT08] *V-Modell XT, Version 1.3.* http://www.v-modell-xt.de, 2008. (Zugriff: 4. Januar 2010).

[W3C07] W3C: *XQuery 1.0: An XML Query Language – W3C Recommendation 23 January 2007.* http://www.w3.org/TR/xquery, 2007. (Zugriff: 4. Januar 2010).

[WD07] Wagner, Stefan und Florian Deissenboeck: *An Integrated Approach to Quality Modelling.* In: *WoSQ '07: Proceedings of the 5th International Workshop on Software Quality*, Seite 1, Washington, DC, USA, 2007. IEEE Computer Society.

[Wey88] Weyuker, Elaine J.: *Evaluating Software Complexity Measures.* IEEE Trans. Softw. Eng., 14(9):1357–1365, 1988.

[Wit02] Witt, Franz: *Controlling Lexikon.* Dtv; Beck Juristischer Verlag, München, 2002.

[WNF+07] Washizaki, Hironori, Rieko Namiki, Tomoyuki Fukuoka, Yoko Harada und Hiroyuki Watanabe: *A Framework for Measuring and Evaluating Program Source Code Quality.*

In: Münch, Jürgen und Pekka Abrahamsson (Herausgeber): *PROFES*, Band 4589 der Reihe *Lecture Notes in Computer Science*, Seiten 284–299. Springer, 2007.

[Wul73] Wulf, William A.: *Programming Methodology.* In: Goldberg, J. (Herausgeber): *Proceedings of a Symposium on the High Cost of Software*. Stanford Research Institute, September 1973.

[WY08] Wermelinger, Michel und Yijun Yu: *Analyzing the Evolution of Eclipse Plugins.* In: Hassan, Ahmed E. et al. [HLG08], Seiten 133–136.

[WZP05] Whitehorn, Mark, Robert Zare und Mosha Pasumansky: *Fast Track to MDX.* Springer, London, 2005.

[Zel07] Zeller, Andreas: *Empirical Software Engineering 2.0.* Keynote Talk, Fourth International Workshop on Mining Software Repositories, Minneapolis, Minnesota. http://msr.uwaterloo.ca/msr2007/Empirical-SE-2.0-Zeller.pdf, 2007. (Zugriff: 4. Januar 2010).

[Zha08] Zhang, Hongyu: *An Initial Study of the Growth of Eclipse Defects.* In: *MSR '08: Proceedings of the 2008 International Working Conference on Mining Software Repositories*, Seiten 141–144, New York, NY, USA, 2008. ACM.

[ZRDvD08] Zaidman, Andy, Bart Van Rompaey, Serge Demeyer und Arie van Deursen: *Mining Software Repositories to Study Co-Evolution of Production & Test Code.* In: *ICST* [ICS08], Seiten 220–229.

[Zus97] Zuse, Horst: *A Framework of Software Measurement.* De Gruyter, Berlin, 1997.

Stichwortverzeichnis