Kontinuierliche Qualitätsüberwachung mit CONQAT

Florian Deißenböck, Tilman Seifert
Software & Systems Engineering
Technische Universität München
deissenb,seifert@in.tum.de

Abstract: Obwohl der entscheidende Einfluss der Software-Qualität auf die Produktivität der Wartung und Weiterentwicklung von Software-Systemen allgemein akzeptiert ist, werden Qualitätsaspekte, speziell bei der Entwicklung unter hohem Zeit- und Kostendruck, oft vernachlässigt. Zur Durchführung kosteneffizienter, kontinuierlicher Qualitätssicherungsmaßnahmen bedarf es daher der Unterstützung durch Werkzeuge zur automatisierten Überwachung vielfältiger Qualitätskriterien. Dieses Papier präsentiert das Qualitätsüberwachung-Framework CONQAT und berichtet über die Erfahrungen, die im Laufe des bisher einjährigen Einsatzes gemacht wurden.

1 Motivation

Software-Qualität hat einen entscheidenden Einfluss auf die Produktivität der Wartung und Weiterentwicklung von Software-Systemen, die bis zu 80% der Lebenszykluskosten ausmacht [Boe81]. Die hierbei relevanten Kriterien sind äußerst vielfältig und betreffen eine großer Anzahl unterschiedlichster Aspekte [BDP06]. Neben organisatorischen Aspekten wie der Qualifikation der Mitarbeiter, bietet die Sicherstellung der Software-Qualität eine vielversprechende Möglichkeit zur Verbesserung der Produktivität der Software-Wartung und damit zur substantiellen Kosteneinsparung.

Die besondere Herausforderung hierbei liegt in der Dynamik, der die Systeme unterliegen. Da die Evolution der Software typischerweise auch zu einem Verfall der Qualität führt [EGK+01, Par94, DP05], ist eine kontinuierliche Überwachung ausgewählter Qualitätsparameter notwendig, um dem fortschreitenden Verfall entgegen zu wirken. Diese Überwachung muss zeitnah und kontinuierlich durchgeführt werden, da nachträgliche Behebung von Qualitätsproblemen meist sehr aufwändig ist [Ree99]. Hinzu kommt, dass die für ein bestimmtes Projekt relevanten Qualitätskriterien zu Beginn oft noch nicht bekannt sind und sich, wie das System selbst, weiterentwickeln. Diese Entwicklung ist sowohl durch die aktuelle Projektphase, das sich entwickelnde Umfeld des Systems als auch durch die im Projekt erworbenen Erfahrungen bedingt. Da eine manuelle Überprüfung relevanter Qualitätskriterien aufgrund der Größe heutiger System nicht praktikabel ist, muss eine kontinuierliche Überwachung durch entsprechende Werkzeuge unterstützt werden.

Dieses Papier beschreibt die zentralen Eigenschaften des Qualitätsüberwachungs-Frameworks CONQAT (Continuous Quality Assessment Tool) [DPS06] und berichtet über die Erfahrungen, die im Laufe des bisher einjährigen Einsatzes gemacht wurden.

2 Anforderungen

Aus den in der Motivation genannten Punkten lassen sich direkt die zentralen Anforderungen an ein Qualitätsüberwachungs-Werkzeug ableiten (vgl. [DPS06]):

Autonomer Betrieb Um zusätzliche Kosten für die Qualitätsüberwachung zu vermeiden, muss das Werkzeug automatisiert und nicht-interaktiv arbeiten und der Qualitätssicherung direkt verwertbare Qualitätsdaten zur Verfügung stellen.

Aggregation Die bei der Qualitätsanalyse anfallenden Daten muss das Werkzeug sowohl im Detail als auch in Form aggregierter Sichten darstellen, um der Qualitätssicherung einen Überblick über den aktuelle Zustand des Systems zu verschaffen. Wichtig ist, dass der Zusammenhang zwischen der aggregierten Darstellung und der Detailinformation erhalten bleibt. Nur so kann ein Qualitätsdefekt, der in einer aggregierten Sicht identifiziert wird, zu seiner tatsächlichen Quelle zurückverfolgt werden.

Vielfältigkeit Qualitätskriterien sind unterschiedlicher Natur und betreffen verschiedenste Artefakte wie z. B. Quelltext, Dokumentation oder Build-Skripte. Hinzukommt, dass für verschiedene Analysen unterschiedliche Sichten auf diese Artefakte eingenommen werden, z. B. Zeilenstruktur, Tokens, abstrakter Syntaxbaum oder Abhängigkeitsgraph bei Quelltext-Artefakten. Diesem Umstand muss das Werkzeug Rechnung tragen, indem es nicht auf die Analyse eines bestimmten Kriterientyps festgelegt ist.

Flexibilität Das Werkzeug muss ein hohes Maß an Flexibilität bieten, um die Ergebnisse unterschiedlicher Analysen in Beziehung zu setzen. Insbesondere muss das Werkzeug um neue Arten von Analysen erweiterbar sein, um den sich ändernden Qualitätsansprüchen Rechnung zu tragen.

3 Verwandte Arbeiten

Unseres Wissens wurden diese Anforderungen bisher von keinem Werkzeug umgesetzt. Werkzeuge mit ähnlichen Zielsetzungen sind der *Sotograph* [BKL04], *Moose* [DLT00] und *iPlasma* [MMM+05], die jedoch einen interaktiven Charakter haben und sich daher für die kontinuierliche Überprüfung nur beschränkt nutzen lassen. Zudem basiert ihre Analyse auf einem Metamodell des zu untersuchenden Systems. Dadurch ist ihre *Vielfältigkeit* eingeschränkt; eine Integration von Analysen beispielsweise der Dokumentation oder des Versions-Management-Repositories ist nicht direkt möglich.

Daneben gibt es eine Vielzahl von hochwertigen Analyse-Werkzeugen mit einem spezialisierten Einsatzgebiet. Bekannte Vertreter sind *JDepend*[1] und *SonarJ*[2] zur Analyse von Architekturverletzungen, *CheckStyle*[3] zur Überprüfung der Einhaltung von Code-Richtlinien und *PMD*[4] zur Untersuchung diverser syntaktischer Anomalien.

Was bisher fehlt, ist die Integration dieser verschiedenen Werkzeuge sowie die Flexibilität, die Analysen zu einem späteren Zeitpunkt auf einfache Weise zu verfeinern und in Zusammenhang zu setzen.

[1] `http://www.clarkware.com/software/JDepend.html`
[2] `http://www.hello2morrow.de`
[3] `http://checkstyle.sourceforge.net/`
[4] `http://pmd.sourceforge.net`

4 CONQAT

Beim Entwurf von CONQAT wurde schnell klar, dass nur eine erweiterbare Struktur mit einer *Plug-In*-ähnlichen Schnittstelle allen Anforderungen gerecht werden kann. Wir haben uns daher dafür entschieden, CONQAT in Anlehnung an das *Pipes&Filters*-Konzept durch ein Netzwerk verschiedener *Prozessoren* zu strukturieren. Diese Prozessoren sind in Java implementiert und jeweils für eine dedizierte Analyse zuständig. Mathematisch gesehen entsprechen sie einer Funktion.

Abbildung 1 demonstriert, wie einfache Prozessoren zu komplexeren Analysen kombiniert werden können. Das dargestellte Prozessornetzwerk dient der Identifizierung von Klassen mit überdurchschnittlich langen Methoden. Hierbei handelt es sich um ein sehr einfaches Beispiel zur Demonstration. Realistischerweise übernehmen Prozessoren komplexere Aufgaben wie die Integration von *PMD* oder die Analyse von JavaDoc-Kommentaren.

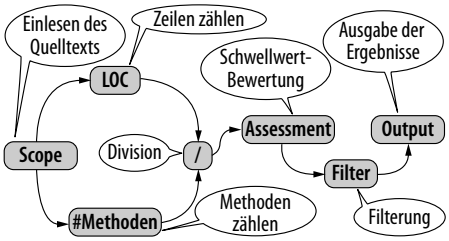

Abbildung 1: Beispielkonfiguration

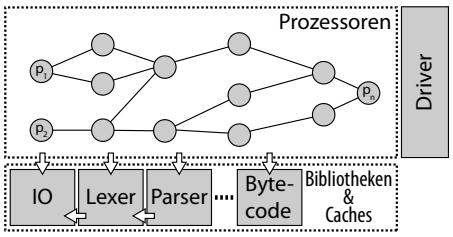

Abbildung 2: CONQAT Architektur

Zentrale Idee dieses Ansatzes ist es, die Analyse des Systems immer vor dem Hintergrund einer oder mehrerer *konkreter* Fragestellungen auszuführen. Im Gegensatz zu anderen Analysesystemen hat CONQAT daher nicht den Anspruch, das System *vollständig* in ein Analysemodell zu überführen. Nur so kann ein Maß an Flexibilität erreicht werden, das es erlaubt, unterschiedlichste Systemartefakte zu analysieren und neuartige Analysen zu integrieren.

Abbildung 2 gibt einen Überblick über die für diesen Ansatz gewählte Architektur von CONQAT. Die Aufgabe des *Drivers* ist es, das per XML-Datei beschriebene Prozessornetzwerk zu instantiieren und dabei die Verträglichkeit der Schnittstellen zu überprüfen. Dies wird durch den geschickten Einsatz von Java *Annotations* und *Reflection* ermöglicht.

Die Bibliotheken bietet den Prozessoren Standard-Funktionalität u. A. für Dateisystemzugriffe, die lexikalische und die syntaktische Analyse. Dies verhindert Redundanzen in der Implementierung der Prozessoren. Da CONQAT im Gegensatz zu vielen anderen Systemen *keine* Transformation des zu analysierenden Systems in eine durch ein Metamodell vorgegebene Standardform durchführt, muss aus Performance-Gründen sicher gestellt werden, dass die verschiedenen Prozessoren rechenintensive Aufgaben wie das Parsen des Quelltextes nicht mehrfach durchführen. CONQAT löst dieses Problem durch den Einsatz einer intelligenten Hierarchie von dynamischen *Caches*. So kann z. B. die Parser-Bibliothek, die u. A. zur Analyse von toten Code verwendet wird, auf eine bereits gecachte Tokenzerlegung des Quelltextes zurückgreifen, falls diese zuvor von einem anderen Prozessor (z. B. bei der Redundanzanalyse) angefordert wurde.

Die Ausgabe der Analyseergebnisse wird von einem Prozessor im HTML-Format erzeugt. Dies ermöglicht die einfache Bereitstellung über einen Webserver. Abbildung 3 zeigt eine durch Nutzung von Ampelfarben aggregierte Ausgabe für vier Analysen des Projekts *ClonEclipse*,[5] sowie den Ausschnitt einer detaillierten Analyse.

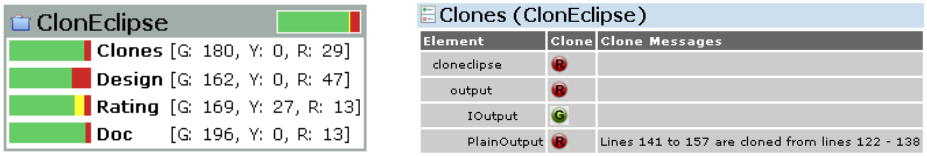

Abbildung 3: Beispielausgabe: Aggregiertes und detailliertes Analyseergebnis (Ausschnitt)

Durch die Realisierung der Ausgabe mit HTML kann der Zusammenhang zwischen der aggegrierten Sicht und der Detailinformation sehr einfach durch Querverweise hergestellt werden. So verweist z. B. ein Klick auf die *Clones*-Analyse zu einer weiteren HTML-Seite mit einer Auflistung der Quelltext-Dateien und den identifizierten Redundanzen.

5 Erfahrungen

Im CCSM (Competence Center Software Maintenance der Technischen Universität München) werden verschiedene Werkzeuge zur Qualitätsbewertung von Software entwickelt: unter anderem CONQAT selbst; *CloneDetective*, ein Werkzeug zur Ermittlung von Code-Duplikaten; *ClonEclipse*, die Integration von CloneDetective in Eclipse; *IdentifierDetective* zur Analyse von verwendeten Bezeichnern (für Variablen, Klassen usw.). Alle Projekte werden aktiv weiterentwickelt. Der Umfang beträgt inzwischen ca. 100.000 Zeilen Java-Code zuzüglich Konfigurationsdaten und Build-Systeme.

CONQAT wird seit etwas über einem Jahr im CCSM für die Entwicklung dieser Werkzeuge eingesetzt. Es wird vorwiegend für die kontinuierliche Bewertung der dort entwickelten Projekte verwendet, aber auch bei der punktuellen Analyse von bestimmten Aspekten eigener und fremder Systeme (Open-Source-Systeme wie z. B. Eclipse sowie große Softwaresysteme bei Industriepartnern) konnte es erfolgreich eingesetzt werden. Hier werden die Erfahrungen mit dem Einsatz von CONQAT anhand von zwei Beispielen dargestellt. Die hier geschilderten Erfahrungen sind zum größeren Teil im akademischen Umfeld entstanden. Die Größe der betrachteten Systeme und die kontinuierliche Arbeit an den Systemen erlauben eine Übertragung der Erfahrungen auf ein industrielles Umfeld zu.

Beispiel 1: Neue Qualitätsanforderungen durch neue Sprachkonstrukte Das Werkzeug *CloneDetective* wurde ursprünglich als eigenstehende Anwendung entwickelt. Erweiterungen der Funktionalität und der Umbau der Anwendung in eine flexibler einsetzbare Bibliothek wurden mit CONQAT begleitet. Die Ziele dabei waren, bei Erweiterungen

[5] Eine vollständige Beispielausgabe der auf unseren Projekten im Rahmen des *Nightly-Builds* durchgeführten Analysen ist unter `http://www4.in.tum.de/~ccsm/conqat-demo/` verfügbar

die Qualität des Systems zu erhalten und neben den Erweiterungsprojekten auch Konsolidierungsprojekte mit klar definierten und überprüfbaren Zielen durchführen zu können.

Mit Hilfe von CONQAT konnten Qualitätsprobleme im Code identifiziert und priorisiert werden. So konnten zielgerichtete Verbesserungen im Code durchgeführt werden, die die Architekturkonzepte klarer zum Ausdruck bringen. Zu den durchgeführten Qualitätsüberprüfungen und -verbesserungen zählen u. A. die Ermittlung und Entfernung von Redundanzen und weiteren Code-Anomalien, vor allem die Entfernung unbenutzten Codes, die Ermittlung von Attributen, die als *final* bzw. von Klassen, die als *immutable* deklariert sein könnten, sowie die Vervollständigung der Inline-Dokumentation mit JavaDoc-Kommentaren, um die Dokumentation kontinuierlich auf dem aktuellsten Stand zu halten.

Während der Weiterentwicklung von Systemen entstehen neue Fragestellungen bezüglich der Qualität. Ein Beispiel dafür ist die Migration des CloneDetective von Java 4 nach Java 5. Der Code soll soweit möglich die Typsicherheit durch generische Typen nutzen. Dafür wurde ein Prozessor entwickelt, der die Verwendung von nicht-generischen Typen analysiert und den zu verwendenden generischen Typ vorschlägt.[6] Seit der abgeschlossenen Umstellung gehört diese Analyse zu den regelmäßigen Überprüfungen; so kann sichergestellt werden, dass der Code dauerhaft die verlangten Qualitätskriterien erfüllt.

Beispiel 2: Logische Architektur und physisches Code-Layout In einer Wachstumsphase von CONQAT mussten die Architektur und die Paketstruktur neu diskutiert werden. Hierfür wurde ein Prozessor implementiert, der die Soll-Architektur mit der Paketstruktur vergleicht. Als Analyseergebnis wird einerseits ein Graph mit aggregierten Nutzungsbeziehungen zwischen Klassen bzw. Paketen erzeugt. Andererseits werden die Klassen aufgelistet, die die vorgesehenen Nutzungsbeziehungen verletzen; mit Hilfe dieser Information können Verletzungen aufgelöst werden. Die Umstrukturierung des Codes wurde wesentlich erleichtert. Ein interessantes Ergebnis der Umstrukturierung war, dass die *logische Architektur* nicht verändert wurde; lediglich das *physische Code-Layout* wurde umgestellt, um die Architekturkonzepte besser im Code widerzuspiegeln.

5.1 Evolution der Qualitätsanalysen

Beide Beispiele verdeutlichen ein wichtiges Konzept der kontinuierlichen Qualitätsanalyse mit CONQAT: Die überprüften Qualitätskriterien werden parallel mit dem System weiterentwickelt. Bestimmte Qualitätsaspekte rücken erst mit steigender Systemgröße in den Fokus des Interesses. Genau zu dem Zeitpunkt, ab dem eine Qualitätsanforderung *konkret* benannt werden kann, ist eine Analyse für die konkrete Fragestellung formulierbar, die ab dann kontinuierlich verfolgt wird. Sie kann entweder aus den existierenden CONQAT-Prozessoren zusammengesetzt werden oder es ist mit geringem Aufwand in Form eines geeigneten Prozessor implementierbar.

Die Implementierung der benötigten Analyse war dank der CONQAT- Infrastruktur sehr schnell zu erledigen: Die genannte Java-5-Überprüfung konnte in einer Studentenarbeit in-

[6] Zu dieser Zeit war hierfür noch kein automatisches Refactoring in den gängigen IDEs verfügbar.

nerhalb einiger Wochen implementiert werden; eine einfache Abhängigkeits-Analyse entstand innerhalb weniger Tage. Dies ermöglichte es uns, innerhalb eines Jahres mehr als 100 Prozessoren bereitzustellen, die ein ein breites Spektrum von Analysen abdecken. Beispiele hierfür sind neben einfachen Prozessoren zur Bestimmung von bekannten Metriken wie LOC oder *Depth of Inheritance Tree (DIT)* auch die Integration bekannter, leistungsfähiger Werkzeuge wie *PMD*, *FindBugs*, *JUnit* oder des Test-Abdeckungs-Analysewerkzeugs *Cobertura*.

Bei der Integration neuer Analysen hat sich die Entscheidung, auf ein explizites Analyse-Meta-Modell zu verzichten, sehr bewährt. So war es möglich auch Analysen zu entwerfen, die neben dem Quelltext auch andere System-Artefakte bzw. Datenquellen wie Build-Skripte oder das Change-Managements-Systems integrieren. Im Rahmen eines Industrieprojekts wird CONQAT derzeit zur Analyse eines Systems vorbereitet, das sowohl mit C-Code als auch MatLab-Simulink-Modellen realisiert ist. Ziel hierbei ist nicht nur die getrennte Analyse der unterschiedlichen Artefakttypen sondern auch die Sicherstellung der Konsistenz zwischen Modellen und Code. Spätestens diese Aufgabe wäre mit einem Analyse-Meta-Modell nicht zu bewerkstelligen gewesen bzw. hätte einen enormen Aufwand zur Integration der Meta-Modelle der sehr unterschiedlich strukturierten Artefakttypen erfordert.

Eine besondere Rolle nimmt ein Prozessor zur Datenbankunterstützung ein, mit dessen Hilfe ausgewählte Qualitätsattribute über die Zeit betrachtet werden können. Eine graphische Darstellung der zeitlichen Entwicklung dieser Attribute erlaubt es, Tendenzen leicht zu erkennen. Auch hierbei wird die zentrale Idee von CONQAT umgesetzt, indem nur die Entwicklung spezifischer Daten, wie z. B. die Größe des Systems im Verhältnis zur Anzahl der Klone, untersucht wird. Daher findet keine generische Persistierung von Analyseergebnissen statt. Falls zu einem späteren Zeitpunkt neue Trendanalysen eingefügt werden, deren rückwirkende Erhebung erwünscht ist, so wird dies über eine automatisierte Analyse der im Versionsmanagement verwalteten Quelltextversionen erreicht.

5.2 Sichten auf die Analysen

Für die *Identifikation und Bearbeitung* von Qualitätsproblemen ist es notwendig, die betroffene Stelle im System präzise zu lokalisieren sowie das identifizierte Problem detailliert zu beschreiben. Für eine *Bewertung des Qualitätsstandards* des Systems ist es jedoch interessant, die Informationen zu einer knappen und aussagekräftigen Darstellung zu verdichten. Entscheidendes Merkmal von CONQAT ist, dass es die Navigation von der verdichteten zu detaillierten Darstellung erlaubt.

Selbst wenn eine Reihe von Qualitätsproblemen in einem System bekannt sind, bleibt als dritte Herausforderung die *Priorisierung* der Qualitätsprobleme. Bei der Überarbeitung des CloneDetective konnte aufgrund dieser Darstellung sofort entschieden werden, welche Komponenten zuerst zu überarbeiten sind. Im Beispiel des Code-Restrukturierung von CONQAT war die Unterscheidung in „wesentliche“ und „marginale“ Probleme mit Hilfe der aggregierten CONQAT-Darstellung offensichtlich.

5.3 Performance

Perfomanceüberlegungen stellen mit dem gewählen Ansatz keine Einschränkungen dar. Beim Einsatz im Nightly Build können umfassende Analysen durchgeführt werden. Hierbei ist die Dauer der Analysen nicht relevant, solange sie im Bereich von einigen Minuten bleibt und sichergestellt werden kann, dass am Morgen die aktuellen Analysen vorliegen.

Im CCSM werden jede Nacht ca. 60 Analysen für sieben Projekte mit insgesamt 65.000 Zeilen Code durchgeführt. Hierfür werden zwei Minuten benötigt. Eine punktuelle Analyse des Eclipse-Systems mit ca. 2,5 Mio LoC in ca. 11.500 Java-Klassen, die aufwändige Analysen wie z. B. die Berechnung diverser Metriken, eine Redundanzanalyse (Suche von Code-Duplikaten), Ermittlung unbenutzter Codefragmente sowie weitere Bytecode-Analysen umfasst, kann innerhalb von weniger als 20 Minuten durchgeführt werden. Daher sind mit CONQAT auch für große Systeme Qualitätsüberprüfungen mit ausgewählten Analysen sogar in einem interaktiven Arbeitsmodus möglich. Interessant ist hierbei die Beobachtung, dass die Ein-/Ausgabeoperationen mit Abstand den größten Teil der Laufzeiten ausmachen.

Aufgrund dieser Ergebnisse hat sich das Werkzeug auch in Industrieprojekten bewährt.

Erfolgsfaktor sind hierfür sicherlich die eingesetzten Caches sowie die Tatsache, dass mit aktueller Hardware auch die Quellen von sehr großen Systemen vollständig in den Hauptspeicher geladen werden können.

6 Zusammenfassung und Ausblick

CONQAT unterstützt die kontinuierliche Überprüfung von Qualitätsparametern. Es ist flexibel erweiterbar, um sich ändernde und vielfältige Qualitätsattribute untersuchen zu können. Die Erweiterbarkeit beruht auf dem Ansatz, spezialisierte Analysen zu entwickeln und zu kombinieren, und auf eine Transformation in eine gemeinsame, durch ein Metamodell vorgegebene Struktur zu verzichten. Auf diese Weise ist eine reichhaltige Bibliothek von Prozessoren entstanden; sie umfasst sowohl Prozessoren, die sehr allgemein verwendbar sind und daher einen hohen Wiederverwendungsgrad aufweisen, als auch Prozessoren, die hochspezialisierte Analysen durchführen. Die Entwicklung neuer Prozessoren kann aufgrund der verfügbaren Infrastruktur mit geringem Aufwand erfolgen; dank der Caches wird eine hohe Performanz erreicht.

CONQAT ist in das Standard-Vorgehen in der Softwareentwicklung im CCSM integriert, da es sich sowohl in Studenten- als auch in Forschungsprojekten bewährt hat. Von Industriepartnern wurde großes Interesse am schlanken Ansatz von CONQAT signalisiert.

Zu den nächsten Zielen gehört die Nutzung der Datenbankunterstützung, um die Entwicklung von Trends bestimmter Qualitätsparameter zu verfolgen. Interessant ist die Frage, ob Schwellwerte für bestimmte Qualitätsparameter angegeben werden können, um ein quantitatives Verständnis für Qualitätsmerkmale zu erarbeiten und damit Aufwände für Qualitätsverbesserungen noch gezielter einschätzen und bewerten zu können.

Literatur

[BDP06] Manfred Broy, Florian Deißenböck und Markus Pizka. Demystifying Maintainability. In *Proceedings of the 4th Workshop on Software Quality*. ACM Press, 2006.

[BKL04] Walter R. Bischofberger, Jan Kühl und Silvio Löffler. Sotograph - A Pragmatic Approach to Source Code Architecture Conformance Checking. In *EWSA 2004*, Seiten 1–9. Springer, 2004.

[Boe81] Barry W. Boehm. *Software Engineering Economics*. Prentice-Hall, Englewood Cliffs, N.J., 1981.

[DLT00] Stéphane Ducasse, Michele Lanza und Sander Tichelaar. Moose: an Extensible Language-Independent Environment for Reengineering Object-Oriented Systems. In *Proceedings of CoSET '00 (2nd International Symposium on Constructing Software Engineering Tools)*, 2000.

[DP05] Florian Deißenböck und Markus Pizka. Concise and Consistent Naming. In *IWPC '05: Proceedings of the 13th International Workshop on Program Comprehension*, Seiten 97–106, Washington, DC, USA, 2005. IEEE Computer Society.

[DPS06] Florian Deißenböck, Markus Pizka und Tilman Seifert. Tool Support for Continuous Quality Assessment. In *Workshop on Software Technology and Engineering Practice (STEP)*. IEEE Computer Society, 2006. *to appear*.

[EGK+01] Stephen G. Eick, Todd L. Graves, Alan F. Karr, J. S. Marron und Audris Mockus. Does Code Decay? Assessing the Evidence from Change Management Data. *IEEE Trans. Softw. Eng.*, 27(1):1–12, 2001.

[MMM+05] Cristina Marinescu, Radu Marinescu, Petru Florin Mihancea, Daniel Ratiu und R. Wettel. iPlasma: An Integrated Platform for Quality Assessment of Object-Oriented Design. In *ICSM (Industrial and Tool Volume)*, Seiten 77–80, 2005.

[Par94] David Lorge Parnas. Software aging. In *Proceedings of the 16th international conference on Software engineering*, Seiten 279–287. IEEE CS Press, 1994.

[Ree99] John S. Reel. Critical success factors in software projects. *IEEE Software*, 16(3):18–23, 1999.