ConQAT - Ein Tool zur koninuierlichen Qualitätsanalyse

Alexander Ried

ried@mytum.de

Software Engineering for Business Information Systems (sebis)
Proseminar: IT-Kennzahlen und Softwaremetriken

Technische Universität München
Institut für Informatik I19
Boltzmannstr. 3
D-85758 Garching

Abstract: Bei der Anwendungsentwicklung besteht ein direkter Zusammenhang zwischen Qualität und Produktivität von Wartung und Weiterentwicklung. Um dem Verfall der Qualität entgegenzuwirken, bietet sich eine kontinuierliche Überwachung ausgewählter Qualitätsmerkmale an. Das Programm ConQAT bietet hierfür bereits alle notwendigen Grundfunktionen vom Filtern und Einlesen der Quelldateien, über viele häufig benötigte Analysen und Aggregationen, bis hin zur Darstellung mit navigierbaren Html-Seiten und Graphiken. Durch ConQATs Flexibilität ist gewährleistet, dass es in einem Großteil der heutigen Entwicklungsumgebungen verwendet und auf spezielle Anforderungen zugeschnitten werden kann. Diese Ausarbeitung beschäftigt sich mit ConQAT in der Version 2.6 und zeigt daran exemplarisch Möglichkeiten zur Qualitätsanalyse.

1 Motivation

In Softwareprojekten ist es heute an der Tagesordnung, dass einzelne Komponenten unabhängig voneinander - zum Beispiel von verschiedenen Personen oder auf verschiedenen Kontinenten - entwickelt werden und geprägt sind von hoher Dynamik und häufigen Änderungen. Deshalb bietet es sich an, Regeln zu definieren und durch Analysen deren Einhaltung zu kontrollieren. Wird fehlerbehaftete Software weiterentwickelt, steigt im Allgemeinen der Aufwand für die Behebung eines Qualitätsdefizits. Deshalb ist es sinnvoll, die Analyse regelmäßig (zum Beispiel täglich) durchzuführen. Um durch den Analyseaufwand keine Produktivitätssenkung zu erhalten, sollte die Auswertung autonom ablaufen und die Ergebnisse so aggregieren, dass sie regelmäßig in kurzer Zeit geprüft werden können. Durch die Aggregation darf jedoch nicht die Möglichkeit verloren gehen, einen Mangel im Detail zu betrachten: eine Navigation von aggregierter Übersicht bis hin zu den Detaillinformationen muss also möglich sein. Ein Analysewerkzeug muss flexibel genug sein, um in verschiedensten Projekten einsetzbar zu sein, und um auf sich ändernde Anforderungen eingehen zu können.

Das am Lehrstuhl Software und Systems Engineering an der Technischen Universität München entwickelte Toolkit ConQAT beachtet die oben genannten Anforderungen und bringt bereits Komponenten für häufige Anwendungsfälle mit. Wird die Ausführung in einen kontinuierlichen Buildprozess integriert, reduziert sich der Arbeitsaufwand für die Analysen auf die größtenteils statische Konfiguration. ConQAT ist jedoch auch vollkommen selbstständig lauffähig.

2 Grundlagen

2.1 Pipes and Filters

Um die geforderte Flexibilität zu erreichen, implementiert ConQAT das Pipes and Filters Architekturmuster und schränkt den Typ der verarbeiteten Informationen nicht ein. Dadurch können mit ConQAT nahezu beliebige Daten und Projekte analysiert werden: unabhängig davon, ob es sich um Quelltext, Bytecode, Dokumentation oder Modellspezifikation handelt und ob diese Daten aus (Text-)Dateien stammen, aus Datenbanken oder Projektverwaltungssystemen (zum Beispiel Bugtracker) abgerufen werden.

Das Pipes and Filters Muster legt fest, dass Daten in Pipes oder Transport Objects übergeben werden müssen, und von sogenannten Filtern (in ConQAT Prozessoren) verarbeitet werden [Wi10]. Der Ablauf, gemäß dem ConQAT die Daten verarbeitet, wird über Konfigurationsdateien angegeben. Der sogenannte Driver liest diese Konfiguration, instantiiert die angegebenen Prozessoren und kümmert sich darum, die Prozessoren richtig zu verbinden. Der Datenfluss zwischen den Prozessoren stellt einen gerichteten Graphen dar:

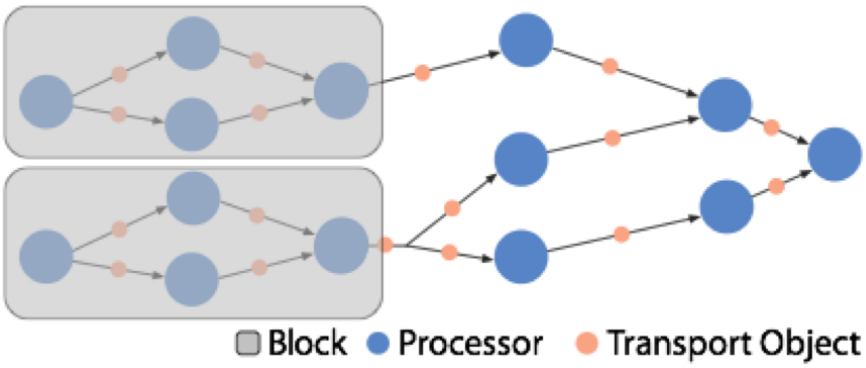

Abbildung 1: ConQAT - Analyse als Graph [Co10]

Prozessoren sind in ConQAT getypt, wodurch der Driver sicherstellen kann, dass ein Arbeitsablauf syntaktisch korrekt ist. Die Ausgabe eines Prozessors dient wiederum als Eingabe für den nächsten Prozessor, jedoch können komplexere Prozessoren auch mehrere

Ausgaben erzeugen oder Eingaben erfordern und somit mit mehreren anderen Prozessoren verbunden sein. Eine Ausnahme bilden erster (keine Eingabe) und letzter Prozessor (keine Ausgabe). Sie werden deshalb auch als Quelle und Senke bezeichnet.

Prozessoren die eine semantische Einheit bilden, können zu Blöcken zusammengefasst werden. Dadurch kann man ganze Arbeitsabläufe einfach wiederverwenden. ConQAT enthält bereits Blöcke für häufige Anwendungsfälle, sodass mit wenig Aufwand diverse Analysen durchgeführbar sind [Co10].

2.2 Treemap

ConQAT verwendet Treemaps zur übersichtlichen Darstellung der Resultate. Dieser weniger bekannte Diagrammtyp ermöglicht die Darstellung von gewichteten Baumstrukturen. Der Baum wird als geschachtelte Rechtecke dargestellt, die Gewichtung erfolgt über Farben. In ConQAT findet man Treemaps zum Beispiel bei der Clonedetection: Die äußeren Rechtecke sind Ordner oder Packages, Unterordner werden als innere Rechtecke beliebig tief geschachtelt dargestellt. Einzelne Dateien werden ebenfalls als Rechtecke innerhalb der passenden Packages gezeichnet, ihre Farbe zeigt an, wie viele Code-Duplikate sich in der jeweiligen Datei befinden. Handelt es sich nicht um Artefakte auf dem Dateisystem, lassen sich andere passende Baumstrukturen erzeugen und darstellen [Hc10]. Zum Beispiel könnte es für die Projektleitung interessant sein, wie sich Änderungen, Bugs oder andere messbare Kenngrößen auf die einzelnen Entwicklerteams und Entwickler (äußere Rechtecke stellen Teams dar, innere Rechtecke einzelne Entwickler, Gewichtung wieder über Farben) aufteilen.

3 Umfang

Um erweiterbar und flexibel bezüglich der Funktionalität zu bleiben, ist ConQAT modular aufgebaut und besteht aus einer Core-Komponente (dem sogenannten Driver), der das Framework initialisiert und die Analyse anstößt, und weiteren Plugins, die die eigentliche Funktionalität zur Verfügung stellen und in jar-Archiven mit Manifest-Dateien organisiert sind.

Abbildung 2: ConCAT - Initialisierung der Bundles [Co10]

Die Abbildung zeigt den Initialisierungsprozess der, vom Driver ausführt wird. Zuerst werden die vorhandenen Archive gesucht und die Manifest-Dateien ausgewertet. Anschlie-

ßend stellt der Driver sicher, dass alle Abhängigkeiten erfüllt sind und keine zyklischen Abhängigkeiten existieren. Sind alle Bedingungen erfüllt, werden die Klassen geladen, instantiiert und initialisiert.

Auch zu erwähnen sind die ConQAT-Eclipse-Plugins, die zwar für die eigentliche Analyse nicht erforderlich sind, jedoch für das Erstellen, Bearbeiten und Auswerten von vorhandenen Analysekonfigurationen eine erhebliche Erleichterung darstellen, und gemeinsam mit ConQAT von der ConQAT-Webseite bezogen werden können.

4 Funktionalität

Die Grundfunktionalität von ConQAT beschränkt sich zwar auf den Driver, jedoch werden viele kleinere, aber auch einige umfangreichere Analysen bereits unterstützt. Im Rahmen dieses Papers sollen Architekturkonformitätsanalyse sowie Duplikatenerkennung genauer betrachtet werden.

4.1 Architecture Conformance Analysis

Die Architektur einer Software beschreibt eine Zerlegung des gesamten Systems in einzelne, logisch voneinander unabhängige Teile. Diese Komponenten können Schnittstellen definieren und mit anderen Komponenten über deren öffentliche Schnittstellen kommunizieren. Welche Komponenten miteinander interagieren dürfen, ist ebenfalls durch die Architektur festgelegt.

Die Vorteile einer Architektur sind offensichtlich: Komponenten können unabhängig voneinander entwickelt werden, die Architekturbeschreibung dient als grobe Übersicht über das System und kann so den Einstieg für neue Entwickler erleichtern, einzelne Komponenten können ersetzt oder weitergegeben beziehungsweise verkauft werden, etc.

Unter dem Verfall der Architektur versteht man ein Divergieren zwischen Ist- und Soll-Zustand. Der Ist-Zustand kann zum Beispiel aus Quellen, Bytecode oder auch Changemanagementsystemen geparst werden. Um ConQAT auch den Soll-Zustand bekannt zu machen, müssen Konfigurationsdateien angelegt werden: Hier wird beschrieben welche Dateien und Packages einzelne Komponenten bilden und wie diese semantischen Einheiten in Zusammenhang stehen. Es ist möglich Zugriffe zwischen den Komponenten zu erlauben oder verbieten, oder zu tolerieren (tolerierte Beziehungen können zum Beispiel verwendet werden, um in der Auswertung gewollte Architekturverletzungen nicht als Fehler darzustellen, weil die Komponenten nicht angepasst werden können oder um den Testklassen Zugriff auf interne Klassen zu gestatten) [Dh10, De10].

ConQAT erstellt zur Analyse je einen Komponenten-Graphen aus Ist- und Soll-Zustand und vergleicht diese beiden Graphen miteinander. Abweichungen zwischen den beiden Graphen stellen Architekturverletzungen dar und werden dem Benutzer zur Überprüfung präsentiert. Zu beachten ist, dass nicht jede gefundene Architekturverletzung einem Fehler

im Programm entspricht. Ebenso kann es sein, dass die Architekturspezifikation falsch oder unvollständig ist, und verbessert werden muss.

4.2 Clone Detection

Eine weitere in ConQAT integrierte Funktion ist die Suche nach dupliziertem Code (Clone Detection). Duplikate können sich negativ auf Wartungskosten auswirken, da Änderungen wie Bugfixes immer an mehreren Stellen durchgeführt werden müssen. Wird dies nicht beachtet, divergieren die duplizierten Stellen und es kann zu unerwarteten Nebeneffekten kommen (als behoben angenommene Fehler treten in anderen Situationen noch auf, etc.). Es gibt jedoch auch Situationen, in denen ähnlicher Code gewollt, durch Programmiersprache oder Artefakttyp unumgänglich oder sogar notwendig ist. Deshalb muss eine Abwägung von Fall zu Fall manuell durchgeführt werden. Beispielsweise lässt sich Fehlerbehandlung oftmals nicht generalisieren oder verwendete Bibliotheken bieten zur Konfiguration keine ausrechend mächtige Syntax um Duplikate zu vermeiden. Durch andere Programmierpraktiken wie zum Beispiel aspektorientierte Programmierung existiert oftmals zwar eine Möglichkeit zur Verbesserung, jedoch können gerade für kleinere Projekte der Einarbeitungsaufwand und die zusätzlich eingeführte Komplexität entscheidende Gründe für den einfacheren Weg (duplizierter Code) sein.

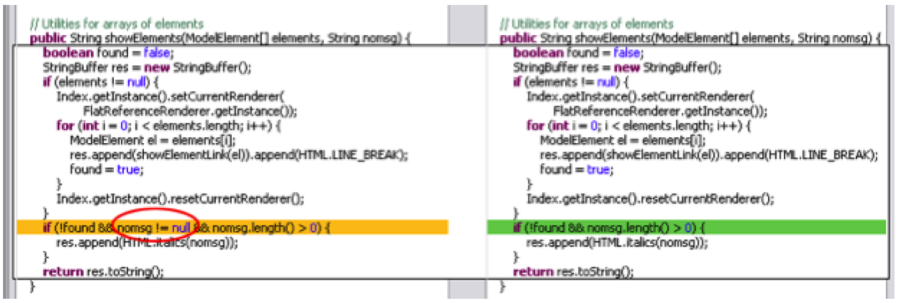

Abbildung 3: ConQAT - Inkonsistente Änderung Nullcheck [De10]

Bei der Erkennung von Duplikaten genügt es jedoch nicht, Textpassagen syntaktisch zu vergleichen: Zum Beispiel ändern Variablennamen, eingefügte Kommentare, oder angepasste Abfragen und Klammerungen an der Semantik eines Code-Fragments nichts, können jedoch nicht gefunden werden, wenn der Quelltext nicht gemäß seiner Bedeutung analysiert wird.

ConQAT bietet zur Duplikatensuche die folgenden Methoden: Text clone detection (findet identische Textpassagen) auf Wort- und Zeilen-Basis und source code clone detection (analysiert die Semantik des Codes und erfordert somit spezifische Parser die semantische Identität erkennen). Außerdem können gewollte Duplikate mit Hilfe von Blacklist-Filtern anhand ihres Hashwertes von der Analyse ausgeschlossen werden. Die Menge von Vor-

kommen eines Duplikats wird in ConQAT als Clonegroup bezeichnet.

Eine wissenschaftliche Auswertung zum Thema Code Clones anhand von Beispielprojekten beziffert die Quote Bugs pro gefundener Clonegruppe auf über 5% [Ju09].

5 Schlusswort

ConQAT bietet ein robustes Framework um Analysen in jedem Anwendungsbereich durchzuführen. Für gängige Sprachen und Analysetypen bringt ConQAT bereits Funktionen mit, die mit wenig Aufwand auf vorhandene Softwaresysteme zugeschnitten werden können. Im Gegensatz zu anderen Tools, die Codequalität über automatisierte Analysen erhöhen sollen, verfolgt ConQAT den Ansatz, fest definierte Fragestellungen zu beantworten statt eine vollständige Analyse durchzuführen. Jedoch sind auch allgemeine Analysen (zum Beispiel für Java das Suchen nach fehlenden @Override-Annotationen, nicht typparametrisierte Verwendung von generischen Klassen oder fehlende Javadoc-Kommentare) mit ConQAT möglich.

In Verbindung mit einem kontinuierlichen Buildsystem kann zudem die Ausführung der Analyse automatisiert werden (zum Beispiel mit Hudson oder Cruise control) [Co10].

Das ConQAT-Buch bietet einen guten Überblick über ConQAT, wodurch es auch für Einsteiger möglich ist, mit wenig Lernaufwand eine funktionierende Analyse einzurichten. ConQAT wird zur Analyse seiner eigenen Quellen verwendet. Dadurch eignen sie sich gut als Referenz.

ConQAT ist unter der Apache License, Version 2.0 [Ap04] veröffentlicht und damit kompatibel zur GPL Version 3. Dadurch ist es möglich ConQAT nach Belieben anzupassen, weiterzuentwickeln und gemäß der Lizenz zu verbreiten.

Literatur

[Ap04] http://www.apache.org/licenses/LICENSE-2.0, last accessed 25.07.2010

[Co10] *ConCAT Source (inkl. Samples)*

[De10] Dießenböck F., Feilkas M., Heinemann L., Hummel B., Jürgens E.: *ConQAT Book*, 2010

[DH10] Dießenboeck F., Heinemann L., Hummel B., Juergens E.: *Flexible Architecture Conformance Assessment with ConQAT*, 2010

[Hc10] http://www.cs.umd.edu/hcil/treemap-history/, last accessed 20.07.2010

[Ju09] Elmar Juergens, Florian Deissenboeck, Benjamin Hummel, Stefan Wagner: *Do Code Clones Matter?*, ISACA 2009

[Wi10] http://de.wikipedia.org/wiki/Pipes_und_Filter, last accessed 20.07.2010