Überdeckungsmessung von Java-Programmen

Beitrag für den 20. *TAV-Workshop* der GI-Fachgruppe "Test, Analyse und Verifikation von Software"
16. und 17. Oktober 2003, method park, Erlangen

Mario Winter[1], Hans-Gerd Wefels[2]
[1]Fachhochschule Köln, Fakultät 10 — Informatik und Ing.-Wiss.
Am Sandberg 1, 51643 Gummersbach, winter@gm.fh-koeln.de, www.gm.fh-koeln.de/~winter
[2]Geologischer Dienst NRW, Fachbereich 21 - Informationstechnologie, De-Greiff-Strasse 195, D-47803 Krefeld

Zusammenfassung: Zur Zeit ist kein Werkzeug zum automatisierten Bedingungsüberdeckungstest für Java erhältlich. Daher wurde Anfang 2001 an der FernUniversität Hagen und der FH Köln ein Forschungsprojekt initiiert, das sich die Entwicklung geeigneter Werkzeuge in diesem Bereich zum Ziel gesetzt hat. Als erstes Produkt ist ein datenbankgestützter Überdeckungsanalysator für Java entstanden, mit dem die beim dynamischen Test erreichte Bedingungsüberdeckung gemessen werden kann. Der Überdeckungsanalysator ermöglicht die Messung der C_0-, C_1- und C_2-Überdeckung, der geworfenen Ausnahmen sowie der minimalen Mehrfach- Bedingungsüberdeckung. Zusätzlich sind einige „objektorientierte“ Überdeckungsmaße verfügbar.

1 Einführung

Viele industrielle Software-Entwicklungsprojekte basieren auf objektorientierten Techniken. Insbesondere in kommerziellen Anwendungsbereichen hat sich in den letzten Jahren für die Implementation die plattform-unabhängige Programmiersprache Java als Stand der Technik etabliert und wird dort zunehmend auch für geschäftskritische Anwendungen eingesetzt.

Mit wachsender Kritikalität der zu entwickelnden Software rückt die Qualitätssicherung gleichberechtigt neben die „konstruktiven“ Entwicklungstätigkeiten, wobei der Test nicht nur in kommerziellen Anwendungsbereichen eine zentrale Rolle spielt. Bekannterweise gestaltet sich das Testen objektorientierter Software schwieriger als für solche, die in einem „herkömmlichen“ (imperativen/prozeduralen) Paradigma erstellt ist. Die daraufhin entwickelten speziellen Testverfahren für Objektorientierte Software gehören noch nicht zum Rüstzeug der meisten Entwickler bzw. Tester [SW02].

Glücklicherweise sind herkömmliche Testverfahren in gewissem Maße auch für objektorientierte Software einsetzbar, wobei hier in erster Linie Black-box Verfahren wie z.B. die Äquivalenzklassenbildung oder der Zustandstest, die ohne Kenntnis der inneren Struktur des Testobjekts arbeiten, in Frage kommen. Um die „Güte“ der durchgeführten Tests zu dokumentieren werden jedoch oft bestimmte Überdeckungen des Programmcodes vertraglich vorgeschrieben.

Oft verlangte Überdeckungskriterien sind die C_0- und C_1-Überdeckung (Anweisungs- und Programmzweigüberdeckung) sowie die C_2- Überdeckung (Entscheidungsüberdeckung). Zur Messung solcher Überdeckungen ist in der Praxis eine Werkzeugunterstützung notwendig, mit denen der Programmcode so instrumentiert wird, dass die bei der Testausführung erreichte Überdeckung ermittelt und dauerhaft abgelegt und dokumentiert wird.

In diesem Beitrag wird ein entsprechendes Instrumentierungs- und Auswertungswerkzeug für die Überdeckungsmessung von Java Programmen vorgestellt.

2 Verfügbare Werkzeuge

Der TAV-Arbeitskreis "Testen objektorientierter Programme" hat 1999 eine Umfrage zu Testwerkzeugen für objektorientierte Software durchgeführt [JW99]. Von den berücksichtigten kommerziell angebotenen Werkzeugen war nur eines in der Lage, die C_0-, C_1- und die C_2- Überdeckung von Java-Programmen zu vermessen — dessen Entwicklung wurde allerdings Anfang 2000 leider eingestellt. Unsere aktuellere Marktuntersuchung förderte weder im kommerziellen noch im Open Source-Bereich ein entsprechend leistungsfähiges Instrumentierungswerkzeug für Java-Programme zutage [Wefels03].

3 DoiT - Datenbankgestützte Überdeckungsmessung

Der entwickelte Instrumentierer *DoiT* (Dokumentation von instrumentierten Testläufen) ist als Datenbankanwendung implementiert und aufgrund der verkapselten Datenbankschnittstelle leicht an jedes relationale Datenbankmanagementsystem anpassbar. Mit *DoiT* ist es möglich, viele Testläufe verschiedener Prüflinge in verschiedenen Versionen in einer oder mehreren Datenbanken dauerhaft zu dokumentieren. *DoiT* ist in der Lage, Messungen zur Ermittlung verschiedener Überdeckungsmetriken an Java-Programmen, bis hin zur minimalen Mehrfachüberdeckung, vorzunehmen. Darüber hinaus ist eine Visualisierungskomponente enthalten, mit deren Hilfe Daten aus durchgeführten Testläufen aus der Datenbank ausgelesen und zusammen mit dem zugehörigen Programm-Quelltext angezeigt werden können.

Mit *DoiT* können Java- Klassen, -Dateien oder ganze Java-Pakete schnell und einfach für Überdeckungstests instrumentiert werden. Beim Instrumentieren der Java-Quellen werden die Instrumentierungspunkte zur Block-, Entscheidungs-, Schleifen-, Selektions-, Klassen-, Methoden- und Ausnahme-Instrumentierung zusammen mit einer Schlüsselnummer der Instrumentierung für dieses Paket in einer relationalen Datenbank abgelegt. Bei der Ausführung der instrumentierten Programme schreiben die Instrumentierungsanweisungen Messwerte über Besuchshäufigkeiten und -reihenfolge in die Datenbank.

Besonders vorteilhaft ist dabei der Einsatz einer relationalen Datenbank, um während der Instrumentierung die ermittelten strukturellen Daten eines Prüflings redundanzfrei abzulegen bzw. zur Laufzeit des instrumentierten Prüflings die an diesen Punkten anfallenden, bewertenden Daten aufzunehmen. Bei der Evaluierung fällt es dann leicht, die so archivierten Daten anhand der zusammengesetzten Schlüssel aus der Datenbank zu selektieren und dem Tester zu präsentieren.

Mit Hilfe eines besonderen Algorithmus zur Zerlegung von zusammengesetzten logischen Ausdrücken aus Konditionalanweisungen und Schleifensteuerungen ist es gelungen, in *DoiT* auch die Instrumentierung und seiteneffektfreie Bewertung logischer Ausdrücke zu realisieren. Die zur Laufzeit ermittelten Bewertungen und Besuchshäufigkeiten werden ebenfalls in der Datenbank abgelegt.

Ein weitere Besonderheit von *DoiT* ist die Fähigkeit, nicht nur Methodenaufrufe innerhalb einer Klasse zu dokumentieren, sondern die verschiedenen Objektkontexte der dynamisch gebundenen Aufrufe von überschriebenen Methoden auch aus abgeleiteten Klassen sichtbar zu machen. Auf diese Weise kann getestet werden, ob und wie oft Methoden einer Oberklassen im Kontext von Instanzen der davon abgeleiteten Unterklassen aufgerufen werden.

Mit Hilfe der ermittelten Messwerte, lassen sich Kennzahlen für Überdeckungsmaße wie C_0-, C_1-, C_2-, minimale-Mehrfachbedingungs-, Klassen-, Methoden- und Ausnahmeüberdeckung angeben (Abb. 1).

```
// Package-Name: junit.framework, Version: 3.8.1
// Instrumentierung Nr. 2, instrumentiert am 29.12.2002
// Ergebnisse des instrumentierten Testlaufes vom 29.12.2002 18:58
// lfd. Test-Nr. dieser Inst-Nr: 2
// Bemerkung zur Instrumentierung: Testen von DoiT mittels JUNit
//
// Globale Messergebnisse für Package junit.framework :
// Klassenüberdeckung                          : 100 %
// Methodenüberdeckung                         : 77 %
// Zweigüberdeckung                            : 74 %
// Bedingungsüberdeckung                       : 77 %
// Minimale Mehrfachbedingungsüberdeckung: 77 %
// Schleifenüberdeckung                        : 100 %
// Ausnahmeüberdeckung                         : 41 %
```

Abb. 1 DoiT Überdeckungs-Report

Zusätzlich zu den Überdeckungs-Reports für ganze Pakete oder Klassen kann aus dem Quelltext zusammen mit den während der Testläufe ermittelten Werten und Kontexten ein umfassender HTML-Report generiert werden. Die vom Test unberührten Anweisungen des Prüflings werden dabei farblich hervorgehoben.

Bei diversen Tests zeigte ‚DoiT' ein gut beherrschbares Verhalten. Ein Test gegen den Quelltext seiner eigenen GUI mit 3000 Zeilen erledigte das Werkzeug in vertretbarer Zeit und mit nur geringem Performanzverlust des instrumentierten Prüflings.

4 Ausblick

Für zukünftige Erweiterungen von DoiT sind folgende Features wünschenswert und angedacht:

- Überdeckungsmetrik für den ‚modified condition/decision coverage Test' der z.B. in der Avionik für sicherheitskritische Software in Flugzeugen gefordert wird. Dabei muss u.a. jede atomare Teilentscheidung einer zusammengesetzten Logik ihren Einfluss auf das Gesamtergebnis einer logischen Kette ‚ceteris paribus' — also unter Beibehaltung der Werte der übrigen Glieder — nachweisen.
- Berücksichtigung der unvollständigen Evaluierung (short-circuit-evaluation) von zusammengesetzten logischen Ausdrücken.
- Verbesserung des Laufzeitverhaltens auch bei I/O-intensiven Testläufen durch asynchrones Schreiben der Datenbank-Updates. Dies kann beispielsweise durch einen SQL-Puffer und einen Wächter-Prozess bzw. -Thread erreicht werden, der erst beim Erreichen einer gewissen Anzahl von DB-Updates asynchron aus dem Puffer in die Datenbank überträgt.
- Realisierung einer „schlanken“ Version mit datenbankunabhängigen Instrumentierungsanweisungen z. B. für eingebettete Systeme (Java MicroEdition, JavaCard).
- Erkennung und Berücksichtigung von Vererbungshierarchien und auf Wunsch Instrumentierung entsprechender Oberklassen (falls diese im Quelltext vorliegen).
- Erweiterung um Sensitivierungs- und Testdatenerzeugungskomponenten z.B. für den Schnittstellentest [Winter01].
- Integration in die Open Source Testwerkzeuge aus dem XP-Umfeld (vgl. z.B. [LF02]).

Danksagung. Dr. Boris Bokowski hat das zur Implementierung verwendete Parser-Rahmenwerk Barat auf unsere Wünsche hin erweitert und auch sonst jederzeit hilfreich und kritisch die Anwendung von Barat begleitet.

5 Literatur und Verweise

[JW99] Stefan Jungmayr, Mario Winter: Testwerkzeuge - Wunsch oder Wirklichkeit. TAV 13, 4.-5.2.1999, Siemens AG, München

[LF02] Johannes Link, Peter Fröhlich: Unit Tests mit Java, dpunkt-Verlag, 2002

[SW02] Harry Sneed, Mario Winter: Testen objektorientierter Software. Carl Hanser Verlag, München, 2002

[Wefels03] Hans-Gerd Wefels: Konzeption und Realisierung eines datenbankgestützten Testwerkzeugs zur Überdeckungsanalyse von Java-Programmen. Diplomarbeit, FernUniversität Hagen, 2003

[Winter01] Mario Winter: Testfallermittlung aus Komponentenschnittstellen. Beitrag zum Imbus QS-Tag 01, Nürnberg, 2001