Glassbox-Test zur Äquivalenzklassenbildung von Produktionsdaten

Rainer Schmidberger
Abteilung Software-Engineering, Institut für Softwaretechnologie
Universität Stuttgart
rainer.schmidberger@informatik.uni-stuttgart.de

Kurzfassung
Produktionsdatenbestände bilden einen guten Fundus an Eingabedaten für Testfälle. Allerdings ist die Zahl dieser Eingabedaten oft unpraktikabel hoch, und immer gleiche Eingaben machen den Test aufwändig, finden aber keine weiteren Fehler. In diesem Artikel wird ein Verfahren vorgestellt, das durch das Glassbox-Test-Werkzeug CodeCover Produktionsdaten auf ein praktikablen Umfang reduziert, ohne die Testgüte (wesentlich) zu beeinträchtigen.

1 Einführung

In der Wartung besteht für den Test oft der Vorteil, dass Praxisdaten in großer Menge zur Verfügung stehen. Gerade bei Systemen mit Batch-Verarbeitung, bei denen Eingabedatensätze nacheinander bearbeitet werden, kann jeder einzelne Datensatz als Eingabe eines Testfalls betrachtet werden. Diese Testfälle haben den Vorteil, dass sie praxistypisch und in großer Menge vorhanden sind.
Allerdings nützt die große Zahl an Testfällen nicht viel, da sich Eingabedaten oftmals wiederholen und so den Test aufwändiger machen, ohne zu einer Verbesserung der Testgüte beizutragen. Die große Anzahl an Testfällen hat zwei entscheidende Nachteile: Erstens führt die Ausführung von vielen Testfällen zu langer Laufzeit und zu einer hohen Ressourcenbelastung, und zum zweiten müssen für alle Eingabedaten die Soll-Resultate ermittelt und mit den Ist-Resultaten verglichen werden. Damit wird die Verwendung der Produktionsdaten als Testfälle unwirtschaftlich.
Helfen würde eine Äquivalenzklassenbildung der Produktionsdaten: Die Reduktion auf jeweils einen Testfall einer Testfallgruppe, deren Elemente hinsichtlich ihrer Fehleraufdeckungsmöglichkeiten als gleich angenommen werden können. Um diese Reduktion einer großen Menge an Testfällen, ohne dabei (wesentlich) die Testgüte zu reduzieren, geht es in diesem Artikel.

2 Definitionen

In diesem Artikel wird Test im Sinne der Definition von Myers verstanden als „Programmausführung in der Absicht, Fehler zu finden“. Der Glassbox-Test ist ein Test, in dessen Verlauf ausgeführte Programm-Entitäten protokolliert werden. Als Programm-Entität sind prinzipiell alle Programmcode-Konstrukte möglich, deren einzelne Ausführung festgestellt und deren Gesamtmenge statisch bestimmt werden kann. In der Praxis sind dies in der Regel Anweisungen, Zweige, Schleifen und Bedingungsterme.

Ein Testfall besteht aus Ausführungsbedingungen, einer Folge von Eingabedaten und einem Soll-Resultat. Während der Testdurchführung werden je Testfall die Eingabedaten in das zu testende System eingegeben, und nach Abschluss der Eingabe wird das Resultat geprüft. Der Beginn der Eingabe wird im Folgenden als Testfallbeginn bezeichnet. Sind die Aktionen, die durch die Eingabedaten ausgeführt werden, vollständig abgearbeitet, ist das Testfallende erreicht. Es folgt dann die Resultatsprüfung.
Das Glassbox-Test-Protokoll für einen Testfall ist die Menge der für den Testfall – von dessen Beginn bis Ende – ausgeführten Programm-Entitäten. Zwei Testfälle gelten als stark äquivalent, wenn sie bei der Programmausführung hinsichtlich ihrer Fehleraufdeckung als gleich anzusehen sind [3]. D. h. bei zwei stark äquivalenten Testfällen hat die Ausführung des zweiten Testfalls keine Chance, weitere Fehler aufzudecken. Auf diese Ausführung kann also ohne Minderung der Testgüte verzichtet werden.
Ein Glassbox-Test-Kriterium reduziert das Glassbox-Test-Protokoll auf die Programm-Entitäten eines bestimmten Typs: z. B. Anweisungen oder Zweige. Zwei Testfälle sind, bezogen auf ein Kriterium *K*, Glassbox-äquivalent, wenn für beide Testfälle das Glassbox-Test-Protokoll, bezogen auf die Programm-Entitäten aus *K*, gleich ist. Die Grundlage der weiteren Betrachtung bildet die Annahme, dass Glassbox-Äquivalenz als Ersatz für die starke Äquivalenz verwendet werden kann.

3 Codecover

Das Glassbox-Test-Werkzeug CodeCover [1, 4] wurde 2007 als Studienprojekt entwickelt und wird nun am Institut für Softwaretechnologie der Universität Stuttgart als Open Source-Projekt weiter entwickelt. CodeCover bietet die für Glassbox-Test-Werkzeuge üblichen Funktionen: Integration in eine Build-Umgebung, Code-Instru-mentierung, Laufzeit-Protokollierung und Auswertung. CodeCover unterstützt derzeit die Sprachen Java und COBOL und besitzt eine Schnittstelle, die eine Erweiterung um weitere Programmiersprachen ermöglicht. Codecover kann die Anweisungs-, Zweig, Schleifen- und Termüberdeckung erheben. Auch hier sind Erweiterungen vorgesehen.
Für das zu untersuchende Java-Programm bildet CodeCover je Anweisung genau eine Programm-Entität, ebenso für Zweige (if-else, case, catch, finally). Bei Schleifen entstehen drei Programm-Entitäten: Eine für keine Ausführung, eine für genau einen Durchlauf

und eine für mehrere Durchläufe. Bei den Prädikaten wird je atomarem Boole'schen Term eine Programm-Entität gebildet und eine für das Gesamtresultat.
Im Unterschied zu den meisten anderen Glassbox-Test-Werkzeugen kann Codecover das Ausführungsprotokoll testfallselektiv führen, d. h. es können die ausgeführten Programm-Entitäten je einzelnem Testfall ausgewertet werden.

4 Vorgehen

Ziel des Vorgehens ist es, aus vorliegenden Produktionsdaten eine effiziente Testsuite zu entwickeln. In einem ersten Schritt wird festgelegt, welche Daten der Produktion als Eingabedaten im Sinne eines Tests zu verstehen sind. Betrachtet man eine Batch-Verarbeitung, bei der Datensätze einzeln eingelesen und verarbeitet werden, könnte jeder Datensatz als Eingabe aufgefasst werden. Wichtig ist, dass sich die Eingabedaten voneinander abgrenzen lassen, d. h. es soll eindeutig feststellbar sein, an welcher Stelle im Programmcode die Daten des Datensatzes *n* vollständig eingelesen und bearbeitet sind, und wo die Bearbeitung des Datensatzes *n+1* beginnt. Der Programmierer setzt nun an diesen Programmcodestellen spezielle Markierungen:

```
// hier beginnt die Bearbeitung
// eines Datensatzes
// startTestCase("Testfall_1");

// hier endet die Bearbeitung
// des Datensatzes
// endTestCase();

// hier endet die Protokollierung
// finishTestSession();
```

Die fett gedruckten Zeilen werden vom CodeCover Instrumentierer erkannt und in spezielle CodeCover-Aufrufe transformiert. Anstelle der Angabe „Testfall_1" wird ein eindeutiges Kennzeichen des Eingabedatensatzes (z. B. ein Primärschlüssel o. ä.) angegeben. Beim Betrieb des instrumentierten Programms werden die ausgeführten Programm-Entitäten (einschl. der Ausführungsanzahl) zusammen mit der eindeutigen Kennzeichnung der verarbeiteten Eingabedaten protokolliert. Zur Auswertung werden die Protokolle für alle Eingabedaten paarweise verglichen. Für ein festgelegtes Kriterium *K* können so Eingabedaten zu Äquivalenzklassen zusammengefasst werden. Im Unterschied zu ähnlichen Verfahren der Literatur (z. B. [2]) werden aber nur Ausführungsprotokolle zusammengefasst, die genau gleich sind. Ein Subsumieren führt nicht zu Äquivalenz. Die Äquivalenz im Sinne dieses Artikels ist symmetrisch. Die Testsuite wird durch die Wahl eines Repräsentanten aus jeder Äquivalenzklasse gebildet. CodeCover wählt hier jeweils den ersten Eingabedatensatz aus. Neben der Ermittlung der Äquivalenzklassen liefert CodeCover auch die Anzahl der Elemente der Äquivalenzklassen, und die zwischen den Äquivalenzklassen verschieden ausgeführten Programm-Entitäten in einer HTML-Tabelle. So kann man leicht nachvollziehen, wo Unterschiede zwischen den Äquivalenzklassen liegen. Zudem sind Äquivalenzklassen mit vielen Elementen als Standardfälle anzusehen, während Äquivalenzklassen mit wenigen Elementen als Ausnahmefälle anzusehen sind.

5 Praxisanwendung

Das beschriebene Verfahren wird für einen Import-Filter eines kommerziellen Informationssystems angewendet. Die Import-Datei, ein XML-Dokument, umfasst etwas über 700 Datensätze. Die Laufzeit des Imports beträgt etwa 30 Minuten. Für einen Test müssen die Soll-Resultate „von Hand" aus der Spezifikation ermittelt und „von Hand" über das System geprüft werden. Das ist für wenige ausgewählte Datensätze möglich; für alle Datensätze der Importdatei wäre der Aufwand zu groß. Die folgenden Reduktionen haben sich für die verschiedenen Kriterien ergeben:

Kriterium	**Äquivalenzklassen**
Ohne Einschränkung	720
Anweisung	32
Zweig	32
Zweig und Schleife	40

Damit entstehen für das Kriterium Zweig- und Schleifen-Gleichheit nur 40 Äquivalenzklassen; eine Reduktion um fast 95%. Die Unterschiede zwischen den Äquivalenzklassen waren überwiegend nur einzelne (kleine) Zweige. Die befragten Fachexperten konnten diese Unterschiede aber gut nachvollziehen.

Zusammenfassung

Produktionsdaten bilden einen guten Fundus an Eingabedaten für Testfälle. In diesem Artikel wird hierzu ein Verfahren beschrieben, wie eine Verdichtung dieser Daten durch Äquivalenzklassenbildung mit dem Glassbox-Test-Werkzeug CodeCover vorgenommen werden kann. In einem Java-Projekt wurde das Verfahren angewendet, die Anwendung in einem COBOL-Projekt ist in Planung.

Literatur

[1] CodeCover Homepage: www.codecover.org

[2] Harrold, M. J., et al.: "A methodology for controlling the size of a test suite", ACM Trans. Softw. Eng. Methodol. 2, 3 (Jul. 1993),

[3] Ludewig, J., Lichter, H.: "Software Engineering", Seite 466-468. dpunkt Verlag 2007

[4] Schmidberger, R.: „Glassboxtest zur Testsuite-Optimierung", Software Engineering 2008 - Beiträge zu den Workshops, GI-Edition, Lecture Notes in Informatics, ISBN 978-3-88579-215-4, März 2008.