Glassboxtest zur Testsuite-Optimierung

Rainer Schmidberger
rainer.schmidberger@informatik.uni-stuttgart.de
Universität Stuttgart, Institut für Softwaretechnologie
Universitätsstr. 38
70569 Stuttgart

Abstract: Der Glassboxtest kann mehr liefern als nur eine Prozentangabe der Programmcode-Überdeckung. Wenn das Glassboxtest-Werkzeug Informationen zum Ablauf der Testdurchführung erhält, ist eine wirksame Hilfestellung bei der Testsuite-Optimierung möglich. In diesem Artikel werden diese Hilfestellungen vorgestellt und am Beispiel des Glassboxtest-Werkzeugs CodeCover beschrieben.

1 Einführung

Die Literatur ist sich einig, dass der Glassboxtest einen wichtigen Beitrag zum Programmtest leisten kann. Zum einen kann Programmcodeüberdeckung als objektive Vollständigkeitsmetrik – z. B. als Testendekriterium – für einen Test verwendet werden. Zum anderen kann der Glassboxtest zur Qualitätsverbesserung einer Testsuite beitragen; der Glassboxtest soll auf fehlende oder redundante Testfälle hinweisen.

Betrachtet man den Glassboxtest im Vergleich zum Blackboxtest, so ist die Herleitung der Testfälle das unterscheidende Merkmal. Beim Blackboxtest wird die Spezifikation als Grundlage zur Herleitung der Testfälle verwendet, beim Glassboxtest werden die Testfälle aus dem Programmcode abgeleitet. Blackbox- und Glassboxtest unterscheiden sich aber zudem in ihrer prinzipiellen Zielsetzung: Während das Ziel beim Blackboxtest das Finden von Fehlern ist und somit die Qualitätsverbesserung des Produkts im Vordergrund steht, ist es beim Glassboxtest zunächst die Bewertung oder Verbesserung der Testsuite. Blackboxtest und Glassboxtest bilden keine Alternativen, sondern der Glassboxtest ergänzt den Blackboxtest.

Der Blackboxtest geht davon aus, dass die Spezifikation vollständig ist. Wir wissen, dass das in vielen Projekten nicht der Fall ist. Gerade bei Änderungen gegenüber einer ursprünglichen Spezifikation oder in der Wartung wird die Spezifikation oft nicht oder nur unzureichend gepflegt. Der Glassboxtest kann auf einen nicht ausgeführten Programmcode hinweisen, aus dem der Tester dann auf solche undokumentierten Anforderungen schließen kann. Welches Sollresultat für bestimmte Eingaben erwartet wird, muss aber außerhalb des Programmcodes und beispielsweise von den betreffenden Fachexperten ermittelt werden. Hier leistet der Glassboxtest keinerlei Hilfestellung.

Eine Werkzeugunterstützung ist beim Glassboxtest Voraussetzung. Am Markt sind viele Glassboxtest-Werkzeuge für viele der kommerziell wichtigen Programmiersprachen verfügbar [Ya06]. Die Produkte sind zum Teil kommerziell, zum Teil Open-Source, und als Resultat des Glassboxtests liefern die Werkzeuge typischerweise Angaben über Anweisungs- und Zweigüberdeckung. Auswertungsmöglichkeiten zur Programmcodeüberdeckung stehen mit diesen Werkzeugen zur Verfügung. Eine Hilfestellung für den Tester zur Qualitätsverbesserung der Testsuite wird von den Werkzeugen aber nur ansatzweise angeboten. In diesem Artikel wird das Open-Source-Glassboxtest-Werkzeug CodeCover vorgestellt, das speziell die Qualitätsverbesserung der Testsuite unterstützt.

1.1 Definitionen

In diesem Artikel wird Test im Sinne der Definition von Myers [My79] verstanden als „Programmausführung in der Absicht, Fehler zu finden“. Der Glassboxtest ist ein Test, in dessen Verlauf ausgeführte Programmelemente protokolliert werden. Als Programmelement sind prinzipiell alle Programmkonstrukte möglich, deren einzelne Ausführung erhoben werden kann und die Gesamtmenge bekannt ist. In der Praxis sind dies in der Regel Anweisungen (im Sinne der Grammatik), Zweige und Bedingungsterme. Testüberdeckung ist beim Glassboxtest das Verhältnis der ausgeführten Programmeinheiten zu allen im Programm enthaltenen Einheiten. Im Übrigen wird mit dieser Definition auch deutlich, dass eine Pfadüberdeckung nicht bestimmt werden kann. Die Ausführung einzelner Pfade kann zwar erhoben werden, die Gesamtmenge aller Pfade ist aber (in der Regel) unendlich.

Ein Testfall besteht aus Ausführungsbedingungen, einer Folge an Eingabedaten und einem Sollresultat. Während der Testdurchführung werden je Testfall die Eingabedaten in das zu testende System eingegeben und nach Abschluss der Eingabe wird das Sollresultat geprüft. Der Beginn der Eingabe wird im folgenden als Testfallbeginn, die Sollresultatsprüfung als Testfallende bezeichnet. Eine Testsuite umfasst mehrere Testfälle.

1.2 Verwandte Arbeiten

Lyu, Horgan und London beschreiben in [Ly93] das Glassboxtest-Werkzeug ATAC (Automatic Test Analysis for C). ATAC erhebt Datenflussüberdeckung und enthält Funktionen, um die Ausführung einzelner Testfälle zu vergleichen. Allerdings kann Testfallbeginn und -ende nicht während eines Programmlaufs bestimmt werden, sondern jeder Programmlauf – von Programmstart bis Programmende – entspricht der Ausführung für einen Testfall. Für den Test einzelner kleiner Funktionen ist diese Einschränkung akzeptabel, beim Test großer Systeme aber unpraktisch.

Eine selektive Verkleinerung einer Testsuite schlagen Jeffrey und Gupta [Je05] vor. Sie beschreiben ein Verfahren, bei dem die Testsuite zwar verkleinert, der Verlust an Testgüte aber möglichst gering gehalten werden soll. Sie führen ihre Untersuchungen an relativ kleinen Programmen durch und verwenden dabei das Werkzeug ATAC zur Datenerhebung. James und Harrold [Ja03] untersuchen die Testsuite-Verkleinerung auf der Basis der Bedingungsüberdeckung der „Modified Condition/Decision Coverage“. Sie betrachten speziell sicherheitskritische Software mit einer sehr hohen Zahl an Testfällen.

Größere Systeme untersuchen Piwowarski et al. [Pi94] und Kim [Ki03]. Sie berichten von praktischen Erfahrungen beim Einsatz des Glassboxtests bei großen Systemen, ohne dabei aber auf Testsuite-Verbesserung einzugehen. Sie stellen fest, dass die Tester selbst nach einem (soliden) Blackboxtest in vielen Fällen eine zu hohe Selbsteinschätzung der Programmüberdeckung haben.

Der Glassboxtest birgt auch Risiken. Zu dieser Erkenntnis kommen Lawrance et al. [La05] in ihrem Experiment zur Wirkung von Visualisierung im Glassboxtest. Sie untersuchen die Wirkung von grafischer, durch Markierung im Programmcode angezeigter Überdeckung. Sie stellen fest, dass Entwickler, denen die Überdeckung im Programmcode grafisch angezeigt wird, sich eher mit weniger Testfällen zufrieden geben als solche Entwickler, die entweder über keinerlei Glassboxtest-Werkzeug verfügen oder denen die grafische Anzeige nicht zur Verfügung steht. Das Glassboxtest-Werkzeug erzeugt hier offensichtlich eine „trügerische Sicherheit“ und signalisiert durch entsprechende Einfärbung, dass ein Programmcode „ausreichend“ getestet ist. Das Experiment zeigt, dass eine Teststrategie „nur-Glassboxtest“, also Glassboxtest ohne einen vorab durchgeführten Blackboxtest, i. A. nicht als sinnvolles Vorgehen angesehen werden kann. Auf eine „nur-Glassboxtest“-Strategie wird im folgenden auch nicht weiter eingegangen.

1.3 Werkzeuglandschaft

Yang et al. [Ya06] untersuchen und vergleichen 17 Glassboxtest-Werkzeuge für Java und C sowie C++ und geben einen guten Überblick über die aktuelle Marktsituation. Die Werkzeuge unterstützen weitestgehend die gleichen Überdeckungsmaße wie Anweisungs-, Zweig- und Methodenüberdeckung. Datenflussüberdeckung wird von keinem der Werkzeuge unterstützt, was die Autoren auf die Schwierigkeiten bei der Interpretation der Ergebnisse zurückführen. Bei der Anweisungsüberdeckung basieren die Werkzeuge teils auf Programmblöcken, auf Anweisungen im Sinne der Grammatik oder auf Programmzeilen. Einzelne Werkzeuge unterstützen auch Bedingungsüberdeckungen. Unterschiede zwischen den Werkzeugen bestehen im Wesentlichen in der Integrationsmöglichkeit in einen bestehenden Build-Prozess und bei den Reportfunktionen. Insgesamt müsste man aus der Zahl der Anbieter von Glassboxtest-Werkzeugen auf eine lebhafte Nachfrage schließen. In der Untersuchung von Müller et al. [Mü98] zeigt sich allerdings, dass kontrollflussorientierte Testverfahren keine hohe Verbreitung haben. Selbst Anweisungsüberdeckung praktizieren nur 12 % der befragten Firmen (wobei in dieser Untersuchung auch 12 % der Firmen Pfadüberdeckung einsetzen, was zeigt, dass besonders beim Test eine erhebliche Begriffsverwirrung bei den Praktikern herrscht).

1.4 Nutzen des Glassboxtests

Der Nutzen des Glassboxtests wird in der Literatur zusammenfassend wie folgt beschrieben:

- **Codeüberdeckungsmaße als Alternativ-Metrik zur Testgüte:** Der Glassboxtest liefert für die Durchführung einer Testsuite die ausgeführten Programmbereiche (z. B. Anweisungen oder Programmzweige), die ins Verhältnis zum gesamten Programm gesetzt werden. So ergibt sich für eine Testsuite ein objektives Vollständigkeitskriterium, das beispielsweise als Testendekriterium verwendet werden kann. Im Wesentlichen besteht Einvernehmen darüber, dass Codeüberdeckung und Testgüte korrelieren, wobei eine hohe Codeüberdeckung ein nötiges, aber kein hinreichendes Merkmal für eine hohe Testgüte darstellt [Lu07][Sp05].
- **Testsuite-Erweiterung:** Piwowarski et al. berichten in [Pi93], dass Tester beim Systemtest die Qualität einer Testsuite oftmals überschätzen und durch den Glassboxtest von der Menge an ungetestetem Programmcode überrascht werden. Der Glassboxtest zeigt diesen ungetesteten Programmcode. Allerdings ist es gerade bei Systemen mit viel Programmcode und großer Testsuite unter Verwendung heute verbreiteter Glassboxtest-Werkzeuge für einen Tester ausgesprochen schwierig, auf konkrete fehlenden Testfälle zu schließen.
- **Testsuite-Reduktion:** Das Ziel ist die Kostensenkung beim Test, speziell beim Regressionstest, durch eine Verkleinerung der Testsuite, ohne dabei aber (wesentlich) Testgüte einzubüßen. Die Grundüberlegung basiert darauf, dass Testfällen mit gleicher oder sehr ähnlicher Überdeckung eine geringere Chance auf das Auffinden von Fehlern eingeräumt wird als Testfällen mit deutlich unterschiedlicher Überdeckung. Testfälle, die auf diese Weise als redundant erkannt werden, können entweder ganz aus der Testsuite entnommen oder in der Priorität weiter nach hinten versetzt werden. Damit kommen sie abhängig von der angestrebten Testgüte zum Einsatz [Je05] [Jo03].
- **Grundlage für selektiven Regressionstest:** Zur Kostenreduktion im Regressionstest sollen anstelle der „rerun-all"-Strategie nur einzelne, ausgewählte Testfälle ausgeführt werden. Die Grundüberlegung ist, dass nach einer (kleinen und abgegrenzten) Programmänderung nur die Testfälle ausgeführt werden, die vor der Programmänderung die betreffenden Programmstellen auch ausgeführt hatten [Sn04].

Bei allen genanten Einsatzgebieten des Glassboxtests ist es erforderlich oder sinnvoll die Programmausführung testfallselektiv zu erheben. D.h. die vom Glassboxtest-Werkzeug gespeicherte Programmausführung wird in einzelne Testfälle, jeweils von Testfallbeginn bis –ende, untergliedert.

2 CodeCover

Das Werkzeug CodeCover wurde im Rahmen eines Studienprojekts am Institut für Softwaretechnologie der Universität Stuttgart entwickelt. CodeCover bietet zunächst die gemäß Yang et al. für Glassboxtest-Werkzeuge üblichen Funktionen: Integration in eine Build-Umgebung, Code-Instrumentierung, Laufzeit-Protokollierung und Auswertung. CodeCover unterstützt derzeit die Sprachen Java und COBOL und besitzt eine Schnittstelle, die eine Erweiterung um weitere Programmiersprachen ermöglicht. Im folgenden werden die Funktionen beschrieben, die direkt von der testfallselektiven Ausführungserhebung betroffen sind.

2.1 Testfallselektiver Glassboxtest

Ähnlich wie beim Glassboxtest-Werkzeug ATAC kennt CodeCover die einzelnen Testfälle der Testsuite und kann einzeln für jeden Testfall die Programmcodeüberdeckung angeben. Bei CodeCover wird der Testfall aber nicht durch Programmstart und –ende festgelegt, sondern der Tester gibt explizit Beginn und Ende eines Testfalls an. Die Anwendung muss hierzu nicht beendet werden. Die einzelnen Testfälle müssen auch nicht unmittelbar aufeinander folgen. Es ist denkbar, dass ein Testfall endet und der Tester eine Reihe an Funktionen des zu testenden Systems verwendet (z.B. Rücksprung auf einen Ausgangspunkt), bevor der dann folgende Testfall beginnt. Für die Praxistauglichkeit hat es sich als wichtig erwiesen, dass der Tester sehr einfach Beginn und Ende eines Testfalls bestimmen kann. Beim Test von Java-Anwendungen verwendet CodeCover die Java Management Extension (JMX) zur Übermittlung von Testbeginn und –ende. Dies hat große praktische Vorteile, weil die Anwendungen unabhängig von ihrer Architektur gleich behandelt werden können und auch eine Steuerung über das Netzwerk kein Problem darstellt. Über definierte JMX-Anweisungen werden dem zu testenden System Beginn und Ende eines Testfalls übermittelt. Diese JMX-Anweisungen können über eine beliebige JMX-Konsole oder auch über einen einfachen Eingabedialog, der Teil des eclipse-Plugin von CodeCover ist, ausgelöst werden. Die im zu testenden System erforderliche JMX-Server-Funktion wird bei der Instrumentierung automatisch eingerichtet.

Abbildung 1: Eingabe von Testfallbeginn und -ende

Bei COBOL-Anwendungen wurden bestimmte Kommentarstile festgelegt, die vom CodeCover-Instrumentierer in Prozeduraufrufe transformiert werden. Für Batch-Systeme ist ein Benutzereingriff ohnehin schlecht möglich.

2.2 Redundanzanalyse

Ausgangspunkt der Redundanzanalyse bildet die testfallselektive Erhebung der ausgeführten Programmeinheiten. Aus dem paarweisen Vergleich können solche Testfallpaare angezeigt werden, die eine gleiche oder nahezu gleiche Ausführung haben. CodeCover visualisiert für ausgewählte Testfälle die Resultate dieses paarweisen Vergleichs in einer Korrelationsmatrix. Der Grauton gibt die Übereinstimmung zweier Testfälle an.

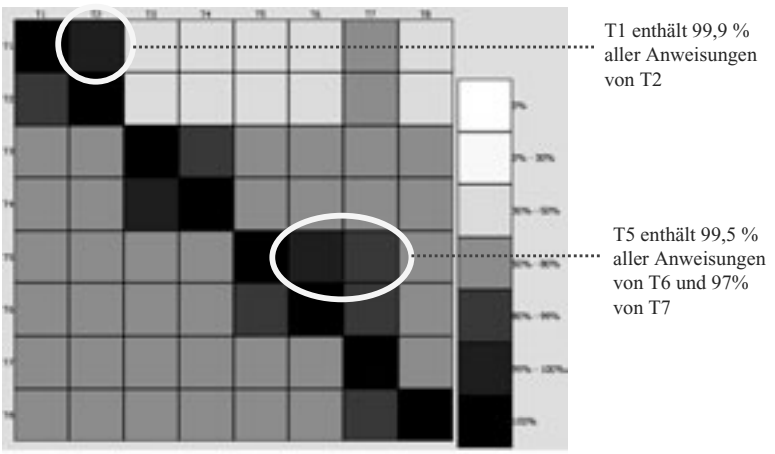

Abbildung 2: Beispiel einer Korrelationsmatrix

Im Beispiel von Abbildung 2 wurde der Testfall T1 wiederholt und als Testfall T2 erneut ausgeführt. Die Übereinstimmung von T1 und T2 ist damit nahe 100 %. Tatsächlich wurden bei T1 und T2 etwa 3000 Anweisungen gleich und 5 verschieden ausgeführt. Der auf den ersten Blick überraschende Unterschied liegt an unterschiedlich ausgeführten Singletons. Im Unterschied zu den Arbeiten von Jones und Herrold [J003] sowie Jeffrey und Gupta [Je05] sollen die Auswertungen lediglich Hinweise auf Redundanzen liefern und damit eine Inspektion der betroffenen Testfälle empfehlen. Eine automatische Testsuite-Reduktion ist nicht vorgesehen.

Da beim Einsatz mit großen Testsuites die Visualisierung als Matrix unübersichtlich wird, können die Datenwerte auch als csv-Datei exportiert und außerhalb weiter ausgewertet werden.

2.3 Testfallselektive Visualisierung der Programmausführung

Glassboxtest-Werkzeuge visualisieren ausgeführte Anweisungen in der Regel durch entsprechende Markierungen im Programmcode. Dort, wo Anweisungen ausgeführt wurden, wird der Programmcode grafisch hervorgehoben. Die Auswertung und Visualisierung bezieht sich auf die gesamte Testsuite. In CodeCover können wahlweise einzelne Testfälle in die Auswertung einbezogen oder davon ausgeschlossen werden. Im Beispiel von Abbildung 3 sind die Testfälle der Testsuiten 1 und 3 einbezogen, die Testfälle der Testsuite 2 nicht. Die Visualisierung wie auch die Überdeckungsberechnung erfolgt dann nur für die ausgewählten Testfälle. Umgekehrt werden zu einer im Programmcode selektierten Anweisung die Testfälle angezeigt, die zur jeweiligen Ausführung geführt haben. Im Beispiel von Abbildung 3 wird zur selektierten Anweisung der Testfall T6 angegeben.

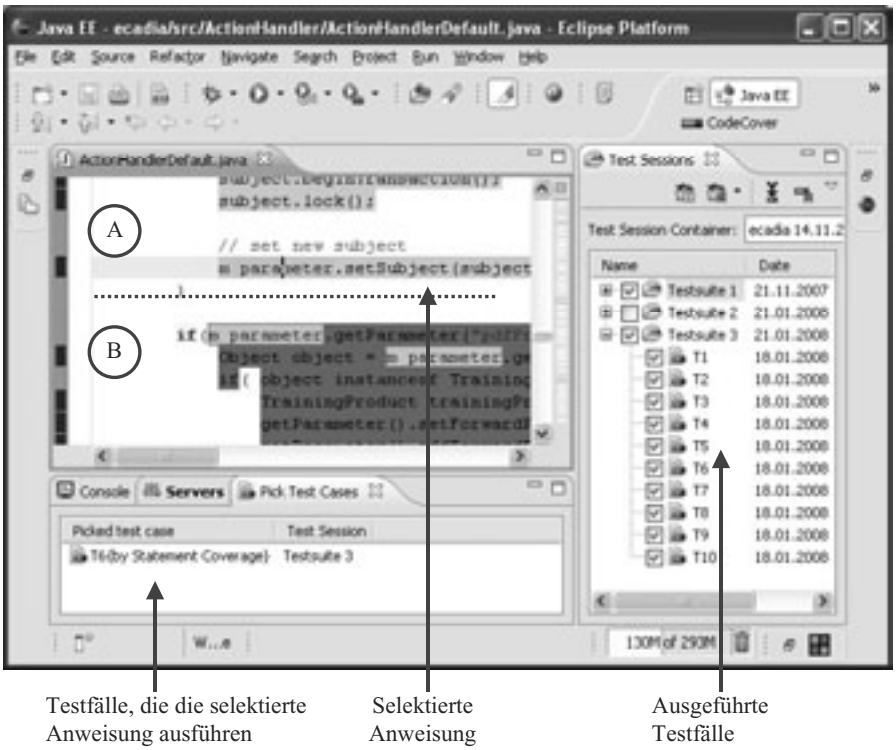

Abbildung 3: CodeCover-Benutzungsschnittstelle

Diese Testfallanzeige kann dazu dienen, fehlende Testfälle leichter zu finden. So besteht die Möglichkeit, einen fehlenden Testfall von einem bestehenden Testfall aus zu beschreiben. Im Beispiel wäre der bestehende Testfall T6, der den Programmcode im Bereich A ausführt. Der Programmcode im Bereich B wird von keinem Testfall ausgeführt. Es ist nun praktisch, den Testfall T6 als Ausgangspunkt zu verwenden und für den fehlenden Testfall das Prädikat der abweisenden Verzweigung einzubeziehen.

Die Testfallanzeige lässt sich auch als Grundlage für den selektiven Regressionstest verwenden. Im Beispiel würden Änderungen im Programmbereich A eine Testdurchführung nur von Testfall T6 unmittelbar erfordern. Bei Programmcode, der nur wenigen Änderungen unterliegt, ist auf diese Weise eine deutliche Kostenersparnis beim Regressionstest möglich.

3 Zusammenfassung

Der Glassboxtest kann einen wichtigen Beitrag zum Programmtest leisten. Das Werkzeug CodeCover stellt hierzu durch eine testfallselektive Erhebung der Programmausführung eine Reihe an Funktionen bereit. CodeCover steht unter EPL-Lizenz und ist unter www.codecover.org frei verfügbar.

Literatur

[Je05] D. Jeffrey and N. Gupta, “Test Suite Reduction with Selective Redundancy,” Proc. 21st IEEE Int'l Conf. Software Maintenance, pp.549-558, 2005.

[Jo03] Jones, J., Harrold, M.: „Test-Suite Reduction an Prioritization for Modified Condition/decistion Coverage”, IEEE Transactions on Software Engineering, Vol. 29, No. 3, March 2003

[Ki03] Kim, Y. W. “Efficient use of code coverage in large-scale software development”, Conference of the Centre For Advanced Studies on Collaborative Research (Toronto, Ontario, Canada, October 06 - 09, 2003). IBM Press, 145-155.

[La05] Lawrance, J., Clarke, S., Burnett, M., Rothermel, G., “How Well Do Professional Developers Test with Code Coverage Visualizations? An Empirical Study”, Proceedings of the IEEE Symposium on Visual Languages and Human-Centric Computing, 2005

[Lu07] Ludewig, J., Lichter, H.: “Software Engineering”. dpunkt Verlag 2007

[Ly93] Lyu, M.R., J.R. Horgan, S. London, "A Coverage Analysis Tool for the Effectiveness of Software Testing," Proceedings of ISSRE’93, Denver, CO, pp. 25-34, November 1993.

[Mü98] Uwe Müller, Thomas Wiegmann: “State of the practice“ der Prüf- und Testprozesse in der Softwareentwicklung — Ergebnisse einer empirischen Untersuchung bei Softwareunternehmen in Deutschland“ Technischer Bericht, Universität Köln, März 1998

[My79] Myers, G. J.: “Art of Software Testing”, John Wiley & Sons, Inc., New York, 1979

[Pi93] Piwowarski, P.; Ohba, M.; Caruso, J.: “Coverage Measurement Experience During Function Test”, Proceedings of the 15th International Conference on Software Engineering, ICSE'93, 1993

[Sn04] Sneed, H.M.: “Reverse engineering of test cases for selective regression testing”, Proceedings of CSMR 2004, pp 69-74, 24-26 March.

[Sp05] Spillner, A., Linz, T.: “Basiswissen Softwaretest”. 3. Auflage, dpunkt Verlag 2005

[Ya06] Yang, Q., Li, J. J., and Weiss, D. 2006. A survey of coverage based testing tools. In Proceedings of the 2006 International Workshop on Automation of Software Test (Shanghai, China, May 23 - 23, 2006). AST '06. ACM, New York, NY, 99-103.