Selective Regression Testing of Large Application Systems

von Harry M. Sneed
Software Test Engineering Consultant
Budapest, Hungary
h.sneed@axelero.hu

Abstract: Dieser Beitrag beschreibt einen Ansatz zum selektiven Regressionstest bei der Wartung und Weiterentwicklung eines großen Anwendungssystems. Die Basis für den Ansatz ist eine System Repository in der die relevanten Beziehungen der Konzepte, Codeelemente, Datenbanktabellen und Testfälle gespeichert sind. Neben der Impactanalyse der Fachkonzept- und Codeelemente wird auch die Reverse Engineering der Testfälle durch statische und dynamische Analyse beschrieben. Das Ziel ist die Reduktion der Regressionstestkosten, die den größten Teil der Lebenszykluskosten ausmachen.

1. Selektives Regressionstesten als Gegenstand der Software Testforschung

Der Regressionstest ist nach der IEEE Standard für Software Engineering Terminlogie, ein erneuter Test einer bereits getesteter Software nach deren Modifikation mit dem Ziel nachzuweisen, daß durch die vorgenommene Änderung keine neuen Defekte eingebaut oder bisher maskierte Fehlerzustände freigelegt werden [1]. Die Software die hier erneut getestet wird, könnte in Abhängigkeit vom Wirkungsbereich der Änderung, eine einzelne Klasse, eine Komponente, ein Teilsystem oder ein ganzes Anwendungssystem sein. Je größer das Testobjekt, desto größer der Ausmaß des Regressionstests. Im ungünstigsten Fall, muß das ganzes System mit sämtlichen Testfällen erneut getestet werden. Deshalb wurde schon immer angestrebt, den Regressionstest auf das kleinste mögliche Element, z.B. die Klasse oder die Prozedur, und dort auf die kleinste Untermenge der Testfälle zu beschränken. Die Reduzierung des Regressionstests auf die aller notwendigsten Testfälle wird als „Selective Regression Testing“ bezeichnet und ist seit vielen Jahren der Gegenstand zahlreicher Forschungsarbeiten [2].

Selektives Regression Testen ist aber nur möglich wenn sicher ist, daß die selektierten Testfälle auch alle Nebenwirkungen der Änderungen mit testen. Diese Sicherheit läßt sich nur durch eine Impactanalyse gewinnen, denn erst durch die Impaktanalyse werden die Nebenwirkungen aufgezeigt.

Nach dem Prinzipien des Change Managements gilt jedes Codeelement als verseucht in dem nur eine Codezeile geändert, gelöscht oder hinzugefügt wurde. Potentiell verseucht sind auch alle Elemente die mit dem verseuchten Element in Berührung kommen. Dazu zählen alle Elemente die das verseuchte Element verwenden, die davon Daten übernehmen oder die darin aufgenommen werden, sei es statisch wie im Falle von Include Members oder dynamisch wie im Falle geerbter Klassen. Die Impactanalyse zielt darauf alle solche Elemente zu identifizieren. Das sind, z.B. Methoden, die verseuchten Methoden aufrufen, Methoden die auf Daten zugreifen, die von verseuchten Methoden erzeugt wurden, Klassen die von verseuchten Klassen erben, und Module die verseuchten Include Members beinhalten. Die Impactanalyse ist ein Versuch alle abhängigen Softwareelemente herauszufinden damit diese in den Regressionstest einbezogen werden können [3].

Ein Schlüsselfaktor zur Reduzierung des Regressionstestaufwandes ist der Grad zu dem Komponente von anderen Komponenten abhängig sind. Je höher dieser Abhängigkeitsgrad, desto höher der Testaufwand. Es müßte daher ein Entwurfsprinzip sein, möglichst „self contained“ Komponente zu schaffen, auch wenn dies auf Kosten der Redundanz geschieht. Es wird wohl nötig sein, fremde Methoden und globale Klassen zu duplizieren, um die Zahl der Asssoziationen und Vererbungen abzubauen. Denn nur so wird es möglich Komponente unabhängig von einander zu testen. In dieser Hinsicht hat die Objektorientierung zu einer verminderten Testbarkeit geführt [4].

2. Regressionstesten als Kostentreiber der Software Wartungspraxis

Falls es nicht gelingt, die betroffenen Elemente über die Impaktanalyse zu identifizieren, bleibt nichts anders übrig als alles zu testen was nur im geringsten mit dem verseuchten Baustein zu tun hat. Je nachdem wie die Software strukturiert ist, könnte dies zu einem erheblichen Testaufwand führen. Im Falle des GEOS Systems, das ein sehr hohes Maß an Interaktion zwischen den einzelnen Komponenten aufweist, müßte wegen der Änderung einer einzelnen Komponente oft ganze Teilsysteme mit sämtlicher Testfälle erneut getestet werden. Da jeder Testfall der zusätzlich durchgeführt und ausgewertet wird, Aufwand verursacht, führte dies bei häufigen Änderungen zu unvertretbaren Kosten für den Regressionstest.

Um den Testaufwand zu reduzieren, wurde ein Projekt angelegt mit zwei Zielen:

a) die von den geänderten Elemente abhängiger Komponente zu erkennen und
b) die Testfälle zu den geänderten Komponenten und den abhängigen Komponente zu selektieren.

Für das erste Ziel wurde eine Impaktanalyse eingeführt. Für das zweite Ziel wurden die Testfälle einer statischen und einer dynamischen Analyse unterworfen. Das Ergebnis sollte eine zuverlässige Untermenge der Testfälle für einen gezielten Regressionstest sein.

3. Impactanalyse mit der GEOS System Repository

Zur Durchführung der Impaktanalyse wird die GEOS Repository herangezogen. Diese wird durch die statischen Analyse der Konzeptdokumente, des Sourcecodes, der Datenbankschemen und der Testfälle aufgebaut. Auf der Konzeptebene werden u.a. die hierarchische Ordnung der Fachfunktionen und deren Assoziationen unter einander festgehalten. Hinzu kommen die „Uses“ Beziehungen zwischen Fachfunktionen und Fachobjekte. Auf der Codeebene werden, u.a. folgende Beziehungen festgehalten:

- Methode zu Methode (Assoziationen)
- Klassen zu Klassen (Vererbungshierarchien)
- Klassen zu Schnittstellen (Datenverwendungsbeziehung)
- Klassen zu Datenbanktabellen (Zugriffsbeziehungen)
- Klassen zu Komponenten (lokale Verpackung)
- Komponente zu Teilsysteme (globale Verpackung)

Bei der Impactanalyse geht es lediglich darum von der geänderten, bzw. verseuchten, Methode, Schnittstelle oder Datenbanktabelle auszugehen und, über einen rekursiven Algorithmus, die Beziehungen zu den verwandten Entitäten zu verfolgen. Da manche Beziehungspfade sich irgendwo schließen, muß hier die Schleife beim Bezug auf ein bereits vorhandene Element abgebrochen werden.

Die Forschung in Impaktanalyse hat ergeben, je weiter ein Softwarebaustein vom verseuchten Baustein entfernt ist, desto geringer die Wahrscheinlichkeit, das er auch verseucht wird. Direkt verwandte Bausteine, bzw. die mit Beziehungen in der ersten Ordnung, haben eine Wahrscheinlichkeit von 18% verseucht zu werden. Indirekt verwandter Bausteine, also die mit Beziehungen in der zweiten oder höheren Ordnung, haben eine Wahrscheinlichkeit von unter 5% verseucht zu werden und diese Wahrscheinlichkeit nimmt mit jeder Stufe weiter ab. [5]

Aus diesem Grunde wurde im vorliegenden Projekt die Abhängigkeiten nur bis zur zweiten Ordnung verfolgt, d.h. es wurden die Aufrufer der Aufrufer der verseuchten Methoden und die Unterklassen der Unterklassen der verseuchten Klassen mit einbezogen, aber nicht weiter. Auf dieser Weise entsteht eine Tabelle der direkt und indirekt betroffenen Komponente. Dadurch wird auch verhindert, daß die Zahl der Regressionstestfälle ausufert.

4. Reverse Engineering der GEOS Testfälle

Die Zuordnung der Testfälle zu den Komponenten wurde bereits in einem früheren TAV Beitrag erläutert [6]. Diese Aufgabe klingt trivial, ist aber keineswegs so einfach in einem Anwendungssystem mit mehr als 20 Teilsystemen, 3000 Komponente und 80.000 Testfälle. Die Testfälle sind explizit nur Konzepten und Teilsystemen zugeordnet weil dies für den Black-Box Test in der Entwicklung ausreichend war. Erst in der Wartung stellte sich die Notwendigkeit Testfällen mit Komponenten zu verknüpfen. Eine Testfall/Komponente Tabelle ist jedoch die Voraussetzung für einen selektiven Regressiontest.

Im GEOS System wird gegen das Fachkonzept getestet. Ergo sind die Testfälle auf die Fachfunktionen bezogen. Dies entspricht dem herkömmlichen Black-Box Testverfahren wonach Testfälle aus der Spezifikation abgeleitet werden. Der Tester hat einen Anwendungsfall vor sich und spezifiziert einen Testfall für jeden Ausgang des UseCases. Im GEOS Projekt wie in den meisten Projekten dieser Art haben Tester mit dem Code nichts zu tun. So können die Testfälle nur einen Bezug zu den Fachfunktionen haben. In der GEOS Testfall Datenbank sind die Testfälle demzufolge nach Teilsystem und Fachfunktion, bzw. Anwendungsfall, geordnet. Wenn ein bestimmter Anwendungsfall zu testen ist wird aus der dazugehörigen Testfällen eine Testprozedur, bzw. ein Testskript, für den Test mit WinRunner automatisch generiert.

Für den Entwicklungstest hat sich dieses Verfahren bewährt, nicht aber für den Regressionstest. Dies hat zwei Gründe. Zum Einen wird das Konzept mit der Zeit nicht mehr fort geschrieben und kann deshalb nicht länger als Testbasis dienen. Zum Anderen wurde vom Anfang an die Relation zwischen Fachfunktionen und Codeeinheiten

nicht dokumentiert. Wäre dies geschehen, hätte man auf die hier beschriebene „Reverse-Engineering“ der Testfälle verzichten können. Denn auch wenn das Konzept aktuell wäre, wäre daraus nicht erkennbar welche Testfälle für eine bestimmte Komponente gelten. Im Wartungsbetrieb sind die meisten Änderungen und Korrekturen lokaler Natur, d.h., sie beziehen sich auf einzelne Codefunktionen und Attribute unterhalb der semantischen Ebene des übergeordneten Konzeptes. Da das Konzept auf diese Detaillierungsebene nicht heruntergeht, muß es gar nicht angepaßt werden. Die Anpassung beschränkt sich lediglich auf den Code. Also bleibt die Beziehung zu den Testfällen unbekannt, in sofern als sie nicht explizit dokumentiert ist.

Im Falle von GEOS existiert eine umfangreiche System Repository in der alle relevante Beziehungen festgehalten sind. Nachträglich wurden auch die Beziehungen zwischen Fachfunktionen und Komponenten eingebaut. Diese Beziehungen wurden über eine statische Analyse der Konzeptdokumente gewonnen. Aus der statischen Analyse der Testfälle werden die Beziehungen der Testfälle zu den Fachfunktionen abgeleitet. Es existieren also zwei relationale Tabellen

- Fachfunktion.zu Komponente und
- Testfall zu Fachfunktion

Mit einer Vereinigung dieser beiden Tabellen ist es möglich die Testfälle den Komponenten zuzuordnen. Eine weitere Tabelle, die aus der dynamische Analyse hervorgeht, ordnet die Testfälle sogar direkt den einzelnen Methoden in den Komponenten zu.

Bei der dynamischen Analyse wird der Pfad eines jeden Testfalles auf Methodenebene verfolgt. Dazu dient eine Tracedatei mit den Namen der ausgeführten Methoden und die Zeit der Ausführung. Mittels der Zeitzuordnung zwischen dem Zeitpunkt der Methodenausführung und den Start- und Endezeiten der Testfallausführung, werden die Testfälle den Methoden und über die Methoden und Klassen den Komponenten zugeordnet. [7]

Beide Ansätze – die statische und die dynamische – führen zu einer Tabelle der Testfall/Komponente Beziehungen. Zum Schluß bleibt nur noch übrig, die beiden Tabellen zu mergen und daraus die Obermenge zu nehmen. Zum Zeitpunkt des Regressionstests, braucht der Tester nur den Namen der zu testenden Komponente einzugeben und es wird eine Regressionstestprozedur für diese Komponente automatisch generiert. Statt das komplette Teilsystem wird nur noch die betreffenden Komponente getestet.

5. Zusammenfassung der Erfahrung

Durch diese kombinierten Repository gestützten Ansätze wurde es ermöglicht einen selektiven Regressionstest auf Komponentenebene auszuführen. Das bewirkt ein großes Sparpotential, das linear steigt proportional zur Anzahl einzelner Fehlerkorrekturen und lokaler Änderungen. Der Nutzen ist also ein erheblich reduzierter Testaufwand um mindestens 50%. Wie dies in der. Softwaretechnologie immer der Fall ist, fordert diese Errungenschaft ihren Preis. Der Preis hierfür sind die Kosten der Repository und deren periodischen Aktualisierung durch die statischen Analyse der Konzepte, Komponente, Datenbanken und Testfälle. Hinzu kommt, wahlweise, der Preis für die dynamische Analyse der Testausführung. Ob der Nutzen diese Kosten rechtfertigt, ist eine Frage die nur das Produktmanagement beantworten kann.

Literaturhinweise

1] IEEE 610: IEEE Std. 610.12-1990, IEEE Standard Glossary of Software Engineering Terminology, IEEE Press, New York, 1990

2] Rothermel, G./ Harrold, M.-J.: „Analyzing Regression Test Selection“, IEEE trans. on S.E., Vol. 22, Nr. 8, August, 1996, s. 529

3] Hartmann, J./Robson, D.: „Techniques for Selective Revalidation“, IEEE Software, January, 1990, s. 31

4] Sneed, H. / Winter, M.: „Testen objektorientierter Software“, Hanser Verlag, München, Wien, 2002, s. 293

5] Law, J./ Rothermel, G.: „Path Profile-based Dynamic Impact Analysis“, in Proc. of IEEE Int. Conf. on Software Maintenance, IEEE Computer Society Press, Montreal, Oct. 2002, s. 262

6] Haschitschka, M. / Sneed, H. : „Reestablishing Links between Test Cases and Concepts via Static and Dynamic Analysis“, Proc. of GI-TAV Meeting, Köln, März, 2003

7] Sneed,H. „Dynamische Analyse verteilter Client/Server Systeme“, Proc. of ICSTEST-2003, SQS GmbH, Köln, April, 2003