Automatische Generierung optimaler modellbasierter Regressionstests

F. Pinte[1], G. Baier[2], F. Saglietti[1], N. Oster[1]

[1]Lehrstuhl für Software Engineering, Universität Erlangen-Nürnberg,
Martensstr. 3, 91058 Erlangen
pinte, saglietti, oster@informatik.uni-erlangen.de

[2]AFRA GmbH,
Henkestr. 77, 91052 Erlangen
gerhard.baier@afra.de

Abstract: Mit der immer größeren Verbreitung modellgetriebener Softwareentwicklung kommt dem modellbasierten Test eine immer wichtigere Rolle zu. Dieser Artikel präsentiert einen neuen Ansatz zur Optimierung des Regressionstests mittels automatischer Erkennung wieder verwendbarer Testfälle und automatischer Generierung einer minimalen Anzahl zusätzlich zu überprüfender Abläufe.

1 Einführung

Die modellgetriebene Softwareentwicklung gewinnt heutzutage immer mehr an Bedeutung. Gerade ihr Einsatz in der industriellen Praxis hat gezeigt, dass sich dabei eine inkrementelle Vorgehensweise besonders anbietet, indem anfänglich erstellte Modelle schrittweise erweitert und anschließend verfeinert werden. Im Allgemeinen ist davon auszugehen, dass im Anschluss an Modell- bzw. Code-Reviews sowie im Laufe des Produkteinsatzes durch Softwareentwickler, Tester oder Benutzer weiterer Änderungsbedarf erkannt und eingebracht wird.

Neben den durch Änderungen bedingten Entwicklungskosten ist vor allem der durch erneute Verifikation des modifizierten Produkts entstehende Aufwand von entscheidender Bedeutung. Es ist daher anzustreben, einen möglichst hohen Anteil der am ursprünglichen Produkt durchgeführten Verifikation zum Zweck der Überprüfung der Änderungskorrektheit wieder zu verwenden. Dies wird insbesondere dadurch erzielt, dass diejenigen Testfälle, die unveränderte Teile des Modells durchlaufen, übernommen und die noch fehlenden Testfälle möglichst systematisch ermittelt werden.

Mit dieser Zielsetzung befasst sich eine Reihe von Forschungsarbeiten, z.B. [Br02, NR07, Fa07], die eine Klassifikation bestehender Testfälle als wieder verwendbare, neu zu validierende und zu verwerfende unterstützen. Weitere Ansätze, u.a. [CPU07, PUA06] befassen sich darüber hinaus mit der Generierung zusätzlich erforderlicher Testfälle, indem sie mittels statischer Analyse der Modelländerungen neue Testfallsequenzen ermitteln.

Unser Ansatz unterscheidet sich in mehrfacher Hinsicht von den bestehenden: Einerseits dadurch, dass wir über die Testfallsequenz hinaus auch die zugehörigen Daten erzeugen, die zur Testausführung erforderlich sind und im Allgemeinen nur mit großem Aufwand ermittelt werden können. Andererseits minimiert unser Ansatz gleichzeitig die Anzahl der zusätzlich erforderlichen und neu zu überprüfenden Testfälle. Dieses Verfahren wurde in ein Werkzeug umgesetzt, das den gesamten oben beschriebenen Regressionstestprozess voll automatisch unterstützt und derzeit beim industriellen Pilotpartner Siemens Medical Solutions erfolgreich eingesetzt wird.

Der vorliegende Beitrag ist folgendermaßen strukturiert: In Kapitel 2 wird die grundsätzliche Automatisierung der Testfallerzeugung zusammenfassend eingeführt. Darauf aufbauend wird in Kapitel 3 unser Werkzeug zur Regressionstestautomatisierung, das bereits industriell eingesetzt wird, beispielhaft illustriert. Auf eine Erweiterung des entwickelten Verfahrens wird abschließend in Kapitel 4 eingegangen.

2 Automatische Testdatengenerierung mit UnITeD

Die automatische Unterstützung der Testphase beschränkt sich im Allgemeinen auf die Ausführung der Testfälle bzw. auf die Generierung der Testsequenzen, d.h. Abfolgen von Operationsaufrufen und Signalen an Komponenten. Das Projekt UnITeD (Unterstützung Inkrementeller TestDaten) befasst sich mit der weiterführenden Aufgabe, aus UML-Modellen vollständige Testfälle (also Testsequenzen mit zugehörigen Eingabedaten) automatisch zu generieren. Das am Erlanger Lehrstuhl für Software Engineering davor entwickelte und bereits erfolgreich eingesetzte Werkzeug .gEAr [Os07] erzeugt mittels genetischer Algorithmen Testfälle auf Quellcodeebene. Im Rahmen des Nachfolgeprojekts UnITeD wurde das .gEAr zugrunde liegende Verfahren zum Zweck der automatischen Unterstützung modellbasierter Komponenten- und Integrationstests entsprechend angepasst. Gemeinsame Zielsetzung von .gEAr und UnITeD ist die Erzielung einer möglichst hohen Überdeckung des Testobjekts nach einem vorgegebenen Kriterium (z.B. Verzweigungsüberdeckung auf Codeebene bzw. Zustandsüberdeckung auf der Ebene einer Zustandsmaschine) bei gleichzeitiger weitgehender Minimierung der dazu erforderlichen Testfälle. Angesichts des mit der Überprüfung der Testfallausführung verbundenen Aufwands ist letztere von besonderer Bedeutung. Einzelheiten zu den Verfahren und ihren Anwendungen sind den entsprechenden Veröffentlichungen zu entnehmen [Os07a, PSO08].

3 Regressionstesterstellung mittels UnITeD

3.1 Klassifikation der Änderungen und Regressionstestfälle

Änderungsbedarf kann zum Hinzufügen, Entfernen oder Ändern von Funktionalität führen. Auf Modellebene äußern sich derartige Modifikationen durch:

- Erweiterung des Modells um zusätzliche graphische Elemente (Knoten oder Kanten).

- Entfernen graphischer Elemente.
- Umbenennung graphischer Elemente ohne Änderung ihrer Semantik.
- Semantische Änderung graphischer Elemente.
- Verfeinerung von Modellteilen.

An obigen Alternativen erkennt man dass Änderungen sowohl innerhalb einer Abstraktionsebene stattfinden als auch von einer Abstraktionsebene auf eine verfeinerte führen können. Anhand der jeweils vorliegenden Änderungen bzw. Erweiterungen wird die bestehende Testfallmenge analysiert und in folgende Klassen unterteilt:

- Klasse „I“: Testfälle, die ausschließlich unveränderte Modellbereiche durchlaufen und deshalb unmittelbar übernommen werden können.
- Klasse „X“: Testfälle, die im veränderten Modell nicht mehr lauffähig sind und in ihrer ursprünglichen Form zu verwerfen sind.
- Klasse „?“: Testfälle, die veränderte Modellbereiche durchlaufen, deren Verhalten deshalb neu überprüft werden muss.
- Klasse „N“: Testfälle, die neu hinzugefügte Modellbereiche durchlaufen und abdecken.

3.2 Beispiel

Beispielhaft werden diese Konzepte an dem in Abbildung 1 illustrierten Modell veranschaulicht, das analog einem UML Aktivitäts- bzw. Zustandsdiagramm das Systemverhalten beschreiben soll. Der Knoten 4 mit zugehörigen Ein- und Ausgangskanten sei in Folge einer Modelländerung entfernt worden. Dafür seien Knoten 6 und 7 mit entsprechenden Kanten hinzugenommen worden. Darüber hinaus sei in Knoten 5 die Annotation bezüglich der Verhaltenskorrektheit revidiert worden („check: x>=0“ statt „check: x>0“).

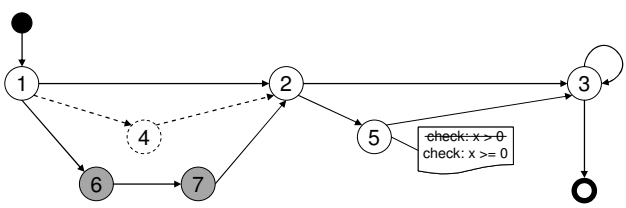

Abbildung 1: Geändertes Modell

Die ursprüngliche kantenüberdeckende Testfallmenge T={T1, T2, T3} lässt sich im Hinblick auf die durchgeführten Änderungen folgendermaßen klassifizieren:

- T1: S-1-2-3-3-E kann unverändert übernommen werden, gehört somit der Klasse „I“.
- T2: S-1-4-2-3-E ist im neuen Modell nicht lauffähig, gehört somit der Klasse „X“.
- T3: S-1-2-5-3-E muss neu überprüft werden, gehört somit zur Klasse „?“.

Um die Kantenüberdeckung im neuen Modell zu erreichen ist zusätzlich zur Testfallmenge {T1, T3} ein neuer Testfall notwendig, etwa:

- T4: S-1-6-7-2-5-3-3-3-E überdeckt die neuen Knoten 6 und 7, gehört somit zur Klasse „N“.

3.3 Methode und Werkzeugunterstützung

Mittels des zum Werkzeug aus Kapitel 2 gehörenden Modellsimulators und einem statischen Vergleich der im alten und neuen Modell durchlaufenen Pfade wird für jeden bestehenden Testfall untersucht, ob er ausschließlich unveränderte Modellbereiche durchläuft (Klasse „I“), erneut zu validieren ist (Klasse „?“) oder im neuen Modell nicht mehr simulierbar ist (Klasse „X“).

Zur Überdeckung derjenigen Elemente im neuen Modell, welche durch die Testfälle aus den Klassen „I“ und „?“ nicht durchlaufen werden, wird der in Kapitel 2 bereits erwähnte Ansatz auf Basis evolutionärer Verfahren gezielt eingesetzt. Dabei werden die wieder zu verwendenden Testfälle in die initiale Population zwar aufgenommen, jedoch im weiteren Verlauf der Generierung nicht mehr verändert. Die durch Evolution angestrebte Optimierung erfolgt multi-objektiv: einerseits mit dem Ziel einer möglichst vollständigen Testüberdeckung bezüglich eines vorgegebenen modellbasierten Überdeckungskriteriums (z.B. Überdeckung aller Zustände eines Automaten), andererseits mit dem Ziel der Minimierung der dazu erforderlichen Testfallanzahl. Die der Evolution zugrunde liegende Fitnessfunktion wird als gewichtete Summe der Optimierungsziele (u.a. Überdeckung und Testfallanzahl) ermittelt.

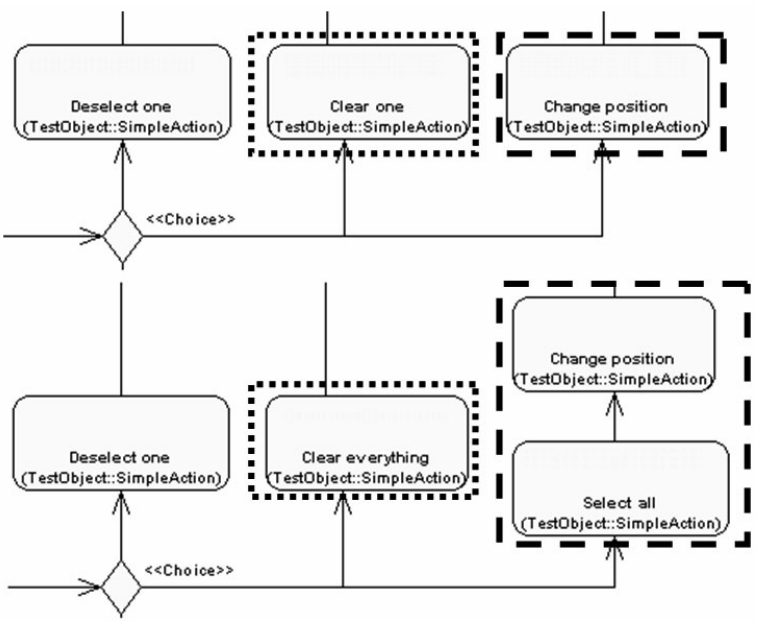

Abbildung 2: Änderung eines Aktivitätsdiagramms
oben: Aktivitätsdiagramm vor der Änderung, unten: Aktivitätsdiagramm nach der Änderung

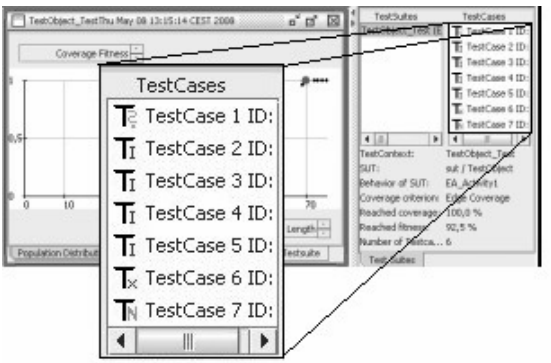

Abbildung 3: Regressionstestanzeige des Werkzeugs

Die Anwendung der Regressionstestfunktionalität des Werkzeugs UnITeD auf die in Abbildung 2 ausschnittsweise dargestellte Änderung eines Aktivitätsdiagramms ist in Abbildung 3 wiedergegeben. Das Werkzeug kennzeichnet jeden Testfall durch entsprechende Symbole (siehe Klassenbezeichnungen aus 3.1), wie im vergrößerten Ausschnitt zu sehen.

4 Effizienzsteigerung der Regressionstestgenerierung

Die in Kapitel 3 beschriebene Vorgehensweise lässt sich dadurch verbessern, dass selbst die im neuen Modell nicht lauffähigen Testfälle (Klasse „X“) bei der Generierung der zusätzlich Erforderlichen (Klasse „N“) weitgehend berücksichtigt und mittels Anpassung ebenfalls wieder verwendet werden.

Dies verspricht eine deutliche Reduktion des Validierungsaufwandes: falls längere, bereits überprüfte Teilsequenzen eines ursprünglich verworfenen Testfalls übernommen werden können, lässt sich der daraus hervorgehende Testfall mit vergleichsweise geringerem Aufwand validieren. Ähnlich wie im Zusammenhang mit der bereits erwähnten automatischen Testfallerzeugung kann auch diese Fragestellung mit Hilfe evolutionärer Verfahren optimiert werden, wobei hier der anzupassende Testfall als Basis für die Anfangspopulation dient. Zusätzlich geht im Rahmen der lokalen Optimierung der Abstand jedes Individuums zum ursprünglichen Testfall in die Fitnessbewertung ein.

Im obigen Beispiel würde sich folgender Testfall der Klasse „N“ durch geringfügige Adaption des nach Kapitel 3 noch zu verwerfenden Testfalls T2 (Klasse „X“) ergeben:

- T4’: S-1-6-7-2-3-E (Knoten 6 und 7 ersetzen Knoten 4).

Diese Fragestellung ist Gegenstand eines aktuellen Teilprojektes.

5 Zusammenfassung

Dieses Papier befasst sich mit der automatischen Generierung und Optimierung von Regressionstestfällen auf Basis von UML Modellen. Die Testtätigkeit kann dadurch in zweifacher Hinsicht reduziert werden: einerseits durch Wiederverwendung automatisch erkannter, weiterhin gültiger Testfälle, andererseits durch automatische Erzeugung einer minimalen Anzahl zusätzlicher, neu zu validierender Testfälle.

Literaturverzeichnis

[Br02] Briand, L. C. et. al.: Automating Impact Analysis and Regression Test Selection Based on UML Designs. In Proceedings of the IEEE Int. Conference on Software Maintenance, Montreal (ICSM), 2002.

[CPU07] Chen, Y.; Probert, R. L.; Ural, H.: Model-based Regression Test Suite Generation Using Dependence Analysis. In Proceedings of the 3rd International Workshop on Advances in Model-Based Testing, 2007.

[Fa07] Farooq, Q. et. al.: An Approach for Selective State Machine based Regression Testing. In Proceedings of the 3rd International Workshop on Advances in Model-Based Testing, 2007.

[NR07] Naslavsky, L.; Richardson, D. J.: Using Traceability to Support Model-Based Regression Testing. In Proceedings of the Twenty-second IEEE/ACM International Conference on Automated Software Engineering, 2007.

[Os07] Oster, N.: Automatische Generierung optimaler struktureller Testdaten für objektorientierte Software mittels multi-objektiver Metaheuristiken. Dissertation, in Arbeitsberichte des Instituts für Informatik, Vol. 40, Nr. 2, Universität Erlangen-Nürnberg, 2007.

[Os07a] Oster, N. et. al.: Automatische, modellbasierte Testdatengenerierung durch Einsatz evolutionärer Verfahren. In Informatik 2007 - Informatik trifft Logistik, Vol. 110 of Lecture Notes in Informatics, Gesellschaft für Informatik, 2007.

[PSO08] Pinte, F.; Saglietti, F.; Oster, N.: Automatic Generation of Optimized Integration Test Data by Genetic Algorithms. In Software Engineering 2008 - Workshopband, Vol. 122 of Lecture Notes in Informatics, Gesellschaft für Informatik, 2008.

[PUA06] Pilskalns, O.; Uyan, G.; Andrews, A.: Regression Testing UML Designs. In 22nd IEEE International Conference on Software Maintenance (ICSM'06), 2006.