Abdeckungskriterien in der modellbasierten Testfallgenerierung: Stand der Technik und Perspektiven

Mario Friske, Bernd-Holger Schlingloff
Fraunhofer FIRST
Kekuléstraße 7
D-12489 Berlin
{mario.friske,holger.schlingloff}@first.fhg.de

Abstract: Zur Beurteilung der Qualität von Testsuiten ist der Abdeckungsgrad ein wichtiges Kriterium. Beim modellbasierten Blackbox-Test wird dabei nicht die Abdeckung im Ziel-Code, sondern in der Spezifikation gemessen. Wir beschreiben anhand eines konkreten Beispiels die von aktuellen Testfallgenerierungswerkzeugen erreichte Abdeckung und argumentieren, dass eine zusätzliche Metastrategie für die Kombination von Abdeckungskriterien erforderlich ist.

1 Einleitung

Die modellbasierte Entwicklung stellt ein neues Paradigma in der Entwicklung eingebetteter Systeme dar. Ausgehend von einer Spezifikation der Anforderungen in natürlicher Sprache wird dabei ein plattformunabhängiges Funktionsmodell erstellt, welches schrittweise zu einem Implementierungsmodell erweitert wird. Aus diesem Implementierungsmodell kann dann der ausführbare Code für die eingebettete Zielplattform automatisch generiert werden. Als Modellierungsformalismen werden dabei häufig Zustandsautomaten der UML oder Signalflussgraphen aus Matlab/Simulink® verwendet.

Zunehmend wird das modellbasierte Paradigma auch verwendet, um Testsuiten automatisch zu generieren: aus dem Funktionsmodell werden in diesem Fall direkt Testfälle erzeugt, mit denen die weiteren Verfeinerungen und Implementierungen getestet werden. Wenn diese Vorgehensweise begleitend zur Systementwicklung eingesetzt wird, können die Tests dazu dienen, die Korrektheit der einzelnen Verfeinerungsschritte zu überprüfen.

Aus verschiedenen Gründen ist die Implementierung jedoch oft nur als Blackbox verfügbar. In diesem Fall dient das Funktionsmodell als Referenz, gegenüber der die Korrektheit von Anforderungen getestet wird. Das Funktionsmodell ist dabei im Allgemeinen eher als deklarative denn als imperative Spezifikation formuliert, in der das „was“ und nicht das „wie“ eines Systems beschrieben wird. Beispielsweise werden unabhängige Aktivitäten wie das Drücken einer Taste durch den Benutzer und das Einschalten eines Motors durch das System in der deklarativen Spezifikation als parallele Aktionen spezifiziert, die kausal voneinander abhängen. Funktionsmodelle für den modellbasierten Test beschreiben also im Allgemeinen eher die Anforderungen an ein System als die Art und Weise, in der die Tests durchgeführt werden sollen.

Es erhebt sich die Frage nach der Vollständigkeit der automatisch generierten Testsuiten. Die im Bereich der Luft- und Raumfahrt häufig verwendete Norm DO-178 B erfordert für Systeme der höchsten Kritikalitätsstufe eine vollständige Testabdeckung nach den Kriterien *„Modified Condition Decision Coverage (MC/DC)“*, *„Branch/Decision Coverage“* und *„Statement Coverage“*. Im Automobilbereich wird häufig auf die Norm IEC 61508 zurückgegriffen, während im Bahnbereich die CENELEC-Norm EN50128 maßgeblich ist. Beide Normen verlangen für Systeme der Sicherheitsstufe SIL4, dass Funktions- bzw. Blackbox-Tests durchgeführt werden, wobei eine Grenzwertanalyse und Äquivalenzklassenbildung dringend empfohlen wird. Aus Sicht eines Testingenieurs ist es wünschenswert, dass die generierten Testsuiten eine definierte Überdeckung der Anforderungen garantieren können.

In [GS05] wird ein Überblick über Abdeckungskriterien für den modellbasierten Test gegeben und die abdeckungsorientierte Vorgehensweise mit statistischen Testmethoden verglichen. Im Folgenden werden Abdeckungskriterien an einem konkreten Beispiel evaluiert und Erweiterungsmöglichkeiten aufgezeigt.

2 Stand der Technik

Wir wollen den aktuellen Stand der Technik in der modellbasierten Testfallgenerierung an dem in Abbildung 1 dargestellten Beispiel erläutern. Es handelt sich um eine auf eine Bewegungsachse beschränkte Realisierung der in [HP02] spezifizierten Sitzsteuerung. Die Bewegung des Sitzes in einer Achse soll über einen Sitztaster mit drei Tasterstellungen gesteuert werden. Wird der Taster betätigt, soll der Sitz in die entsprechende Richtung bewegt werden. Bei Erreichen der jeweiligen Endposition soll die Bewegung unterbrochen werden. Eine weitere Bewegung in diese Richtung soll erst wieder möglich sein, nachdem der Sitz per Taster in die entgegengesetzte Richtung zurückgefahren wurde.

Die Modellierung des Statecharts erfolgte in Anlehnung an die in [DKvK+02] beschriebenen Richtlinien. Das Statechart hat einen orthogonalen Zustand, welcher vier nebenläufige Regionen hat. In diesen sind jeweils die Sensoren *Taster* und *Endposition*, die eigentliche *Steuerung* und der Aktuator *Motor* modelliert. Wird eine durch die Sensoren überwachte Größe geändert, so erfolgt der entsprechende Zustandswechsel. Die Änderung wird der *Steuerung* durch ein zugehöriges Ereignis signalisiert. Die *Steuerung* kontrolliert den Zustand des Aktuators *Motor* ebenfalls durch Ereignisse.

In dem Modell ist auch eine Abhängigkeit modelliert, welche normalerweise im zugehörigen Umgebungsmodell realisiert werden würde, hier jedoch in direkt integriert wurde: Eine Bewegung des Sitzes in der Endposition in die Gegenrichtung hat einen Übergang von der Endposition in die Normalposition zur Folge.

Aus diesem Statechart sollen nun mit einem Testfallgenerator Testfälle erzeugt werden. Ein Testfall ist dabei eine Sequenz von Ereignissen, welche sowohl Eingaben an das zu testende System als auch dessen Reaktionen enthält. Zur automatischen Erstellung solcher Testfälle verwenden wir im Folgenden das Werkzeug ATG [IL], welches als Zusatz zur modellbasierten Entwicklungsumgebung Rhapsody® von I-Logix™ erhältlich ist. Andere

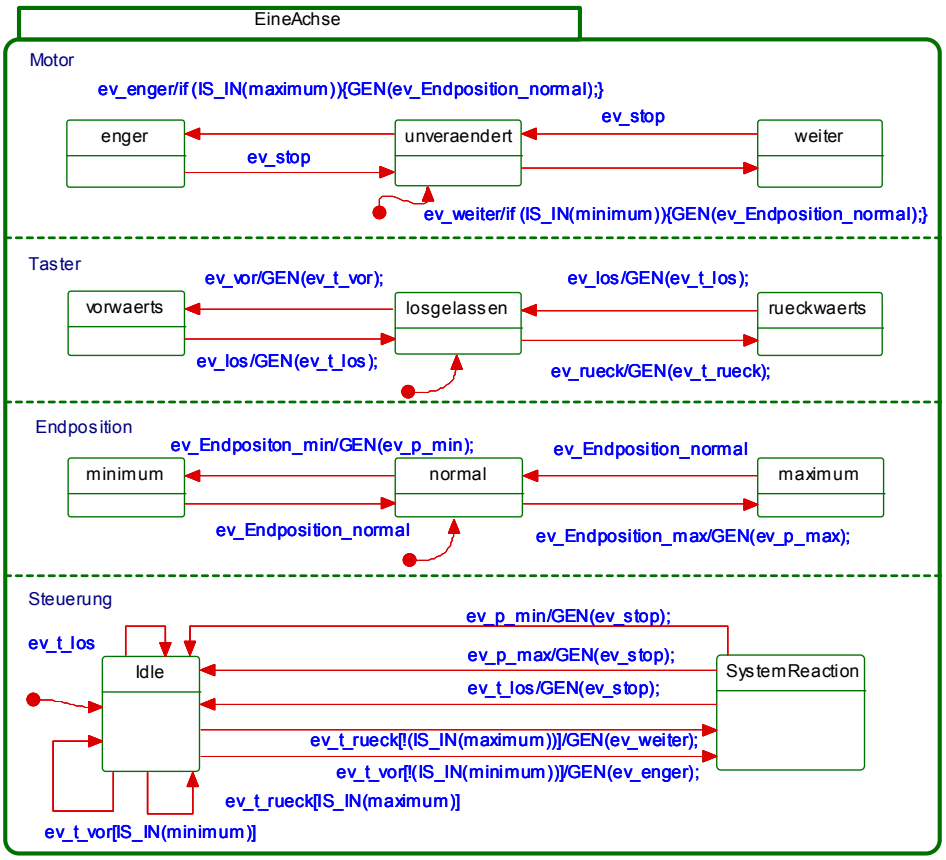

Abbildung 1: Statechart einer vereinfachten Sitzsteuerung

Werkzeuge mit ähnlicher Funktionalität sind z.B. Conformiq® [Con] und Reactis® [Rea]. ATG analysiert das UML-Modell und den daraus generierten C++ Code und generiert einen Satz von Testfällen. Die erreichbare Abdeckung ist MC/DC auf dem generierten Code. Dieses Abdeckungskriterium auf Quelltextebene umfasst die Modellebenenkriterien Transitions-, Ereignis- und Zustandsüberdeckung.

Im ATG werden zunächst die an den Schnittstellen zu betrachtenden Ereignisse festgelegt. Im Fall der Sitzsteuerung sind das nur die Ereignisse, die eine Zustandsänderung der Sensoren und Aktuatoren bewirken. Daraus erzeugt ATG Testziele auf Modell- und Code-Ebene. Auf Modellebenen wird für jedes der Modellelemente *Zustand*, *Transition*, *Operation* und *Ereignis* ein Testziel definiert. Auf Code-Ebene wird für jede Wertekombination in Bedingungen, die zum Erreichen von MC/DC erforderlich ist, ein Testziel erstellt.

Im Beispiel ergeben sich so 96 Testziele. Davon werden beim automatischen Generieren der Testfälle 81 erreicht. Im Detail sind das 17/17 Zustände, 24/24 Transitionen, 0/0 Operationen, 14/14 Ereignisse und 26/41 Code-Abdeckung. Daraus resultiert eine An-

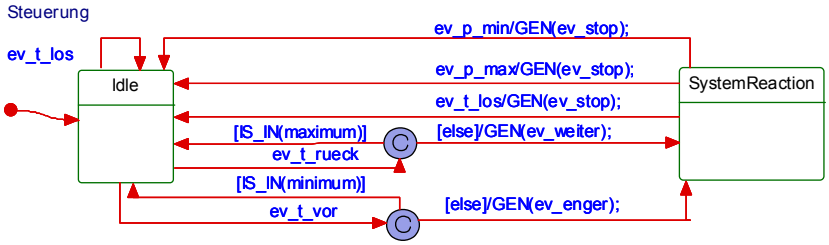

Abbildung 2: Modifikation des Statecharts zur Erhöhung der Testabdeckung

weisungsüberdeckung von 180/213. Bei der Analyse der Testfälle zeigt sich, dass einige Testziele nicht erreicht werden können, da an bestimmten Stellen im Quelltextes innerhalb verschachtelter Fallunterscheidungen sowohl Bedingungen als auch deren Negation generiert werden:

```
//## transition 17
if(!(IS_IN(maximum)))
    { ... }
else
    {
        //## transition 23
        if(IS_IN(maximum))
            { ... }
    }
```

Die Bedingung in der zweiten if-Anweisung kann nicht negiert werden. Daher ist das Kriterium MC/DC für diese Anweisung nicht erfüllbar.

Durch die in Abbildung 2 dargestellte alternative Modellierung unter Verwendung von Konditionskonnektoren lässt sich dieses Problem vermeiden. Die bei der Testfallgenerierung erreichte Abdeckung erhöht sich auf 83/96 Testziele, d.h. die Code-Abdeckung erhöht sich auf 26/39 oder 66 Prozent. Zustände, Transitionen, Operationen und Ereignisse werden nach wie vor vollständig abgedeckt. Im Falle der Transitionen sind es nun zwei mehr, d.h. 26/26. Da aus einem geänderten Modell auch anderer Code generiert wird, schlägt sich die Verbesserung nicht notwendigerweise in der gemessenen Anweisungsüberdeckung nieder. Im Beispiel ergibt sich sogar ein leicht geringerer Wert von 178/211.

Die 13 nicht erreichten Testziele bezüglich Code-Abdeckung erklären sich folgendermaßen: viermal wird das Code-Abdeckungs-Testziel „!IS_IN(EineAchse)“ nur teilweise erreicht, da der Zustand *Eine_Achse* nicht verlassen wird. Fünfmal wird für die den einzelnen Zuständen zugeordnete Funktion *dispatchEvent* keine vollständige *Switch Coverage* erreicht. Viermal wird kann der Funktionsaufruf „EineAchse_Exit“ nicht erreicht werden, da der Zustand *EineAchse* nicht verlassen wird. Diese 13 nicht erreichten Testziele resultieren aus der Art, wie Rhapsody den Code aus dem Statechart generiert und sind auch nicht ohne weiteres zu erreichen. Auf dem erreichbaren Code kann folglich mit den generierten Testfällen 100% Zustandsüberdeckung, 100% Transitionsüberdeckung, 100% Er-

eignisüberdeckung und 100% MC/DC-Abdeckung erzielt werden.

Beim Export wird pro Testziel der resultierende Testfall in einer entsprechend benannten Datei abgelegt. Dadurch entstehen 92 Testfälle. Da verschiedene Testziele zu gleichen Ereignissequenzen führen können, enthalten die Testfälle Duplikate und Inklusionen. Das Format der generierten Testfälle unterscheidet Ein- und Ausgaben, beinhaltet Zeitstempel und ist in der Lage, mit mehr als einem zur Verarbeitung anstehenden Ereignis umzugehen. Da im Beispiel nur die Abfolge der Ereignisse von Interesse ist, lassen sich die Testfälle vereinfacht als Ereignissequenzen darstellen. Nachdem Duplikate und Inklusionen durch einen Postprozessor entfernt wurden, bleiben die in Abbildung 3 gezeigten 13 Testfälle übrig.

```
TC1:
ev_Endpos_max
ev_vor
ev_enger
ev_Endpos_norm

TC2:
ev_Endpos_min
ev_rueck
ev_weiter
ev_Endpos_norm

TC3:
ev_Endpos_norm

TC4:
ev_los

TC5:
ev_rueck
ev_weiter
ev_Endpos_max
ev_stop

TC6:
ev_rueck
ev_weiter
ev_los
ev_stop

TC7:
ev_vor
ev_enger
ev_Endpos_max
ev_stop

TC8:
ev_Endpos_min
ev_Endpos_norm

TC9:
ev_Endpos_max
ev_Endpos_norm

TC10:
ev_vor
ev_enger
ev_los
ev_stop

TC11:
ev_rueck
ev_weiter
ev_Endpos_min
ev_stop

TC12:
ev_Endpos_max
ev_rueck
ev_los

TC13:
ev_Endpos_min
ev_vor
```

Abbildung 3: Aus dem Statechart gemäß Abdeckungskriterium MC/DC generierte Testfälle

3 Perspektiven

Obwohl die so gewonnene Testsuite eine 100%ige Testabdeckung nach MC/DC-Kriterium auf dem Modell erreicht, ist sie intuitiv nicht ausreichend. Etliche Eigenschaften, die bei einer manuell erstellten Testsuite geprüft würden, sind in dieser automatisch erstellten Testsuite nicht enthalten. Ein typisches darüber hinaus gehendes Kriterium ist beispielsweise die in den Anforderungen spezifizierte Eigenschaft, dass sich der Sitz nach Erreichen der Endposition auch bei erneuter Betätigung des Tasters nicht bewegt. Auf Grund der automatischen Generierung ist es nicht möglich, bestimmte als gefährlich vermutete Teilpfade im Statechart als Testfälle einzustellen. Ein anderer zu testender Aspekt ist beispielsweise: „Die Abschaltung in eine Richtung funktioniert auch noch beim zweiten Mal.“ Auch solche wiederholten Abläufe werden durch die in ATG implementierte Testerzeugungsstrategie nicht erfasst.

MC/DC auf dem Code wird in der Industrie oft als Abdeckungskriterium verwendet. Im Beispiel werden die Testfälle unter Verwendung des Kriteriums MC/DC aus dem Modell generiert. Wird das Modell als Spezifikation zum Black-Box-Test einer unbekannten Implementierung verwendet, so ist dies natürlich keine Aussage über die Codeabdeckung der Implementierung. Nach [Pos96] lassen sich Defekte in der Implementierung in die drei

Klassen *wrong code*, *missing code* und *extra code* einteilen. Die Bestimmung von *extra code* erfordert beim spezifikationsbasiertes Testen zusätzliche Codeabdeckungsmessungen. Das spezifikationsbasierte Testen ist besonders geeignet, *missing code* aufzudecken.

Für die Aufdeckung von Defekten der dritten Kategorie *wrong code* ist es ratsam, stärkere Kriterien als das von den aktuellen Testfallgeneratoren umgesetzte MC/DC zu verwenden. In [Bin99] werden solche Kriterien zum Black-Box-Test von Zustandsmaschinen diskutiert, u.a. *All n-Transition Sequences*, *All Round-trip Path* und *M-Length Signature*. Mit zunehmender Mächtigkeit der Strategien steigt die Zahl der abgedeckten Fehlerklassen. Zu jeder dieser Strategien lassen sich jedoch jeweils Fehlerklassen konstruieren, welche durch diese nicht abgedeckt werden können. Außerdem steigt die Anzahl der notwendigen Testfälle explosionsartig an. In [Bin99] wird der Versuch unternommen, die einzelnen Testverfahren nach den von ihnen abgedeckten Fehlerklassen zu klassifizieren. Eine solche abdeckungsorientierte Klassifikation entspricht wesentlich mehr der intuitiven Herangehensweise bei der manuellen Testfallerstellung als die üblichen verfahrensorientierten Klassifikationen von Testverfahren [Lig02].

In der Praxis werden häufig die angewandten Abdeckungskriterien durch die zur Verfügung stehenden Werkzeuge bestimmt. Aktuell verfügbare Werkzeuge unterstützen Kombinationen von Abdeckungskriterien und zugehörigen Generierungsstrategien nur unzureichend. In unserem Beispiel könnte eine kombinierte Strategie beispielsweise diese Kriterien umfassen: (a) MC/DC auf dem generiertem Code, (b) Erzeugen bestimmter Ereigniskombinationen und (c) Erzeugen von Sequenzen bestimmter Länge. Sofern die Ereignisse Parameter enthalten, ist die Kombinationen von Parametern ein mögliches zusätzliches Kriterium.

In einem Industrieprojekt haben wir eine solche kombinierte Strategie realisiert. Dabei werden zusätzliche Modellelemente in das Modell eingefügt, welche den Generator dazu bringen, weitere Testfälle zu erzeugen. Die so erhaltene Testsuite wird mit einem von uns entwickelten Postprozessor weiterverarbeitet, um zusätzliche Abdeckungskriterien zu erreichen.

Wir sind gerade dabei, die in diesem Projekt gewonnenen Erfahrungen zu verallgemeinern. Bei der manuellen Erstellung von Testsuiten geht ein Testingenieur oftmals von gewissen Risiken und Szenarien aus, die getestet werden sollen. Auch bei der automatischen Erzeugung von Testfällen aus Spezifikationen wird daher eine Metastrategie benötigt, um die Generierung zu steuern. Kriterien für die Testabdeckung sollten idealerweise auf der gleichen Abstraktionsebene wie in der modellbasierten Entwicklung spezifiziert werden können, d.h. das „was“ und nicht das „wie“ sollte im Vordergrund stehen. Eine Realisierungsmöglichkeit für die Angabe solcher Metastrategien besteht darin, die zu erzielenden Abdeckungskriterien, abzudeckenden Risiken und Kombinationen davon als *deklarative Testgenerierungsdirektiven* zu formulieren. Ein Beispiel für solche Direktiven sind die Kombinationsregeln im CTE/XL [LW00].

Aus dieser Idee ergeben sich eine Vielzahl von Fragestellungen, insbesondere im Hinblick auf Automatisierungsmöglichkeiten und Werkzeugunterstützung. Wie spezifiziert man zu testende Aspekte und zugehörige Abdeckungskriterien? In wie weit kann dies von Werkzeugen unterstützt werden? Wie ist eine Integration in bestehende Werkzeuge realisierbar?

Denkbar sind beispielsweise Werkzeuge, die Modell und Anforderungen analysieren und dem Tester modellspezifische Abdeckungskriterien zur Auswahl anbieten.

4 Zusammenfassung

In diesem Papier haben wir an Hand eines konkreten Beispiels die Verwendung des Werkzeuges ATG zur automatischen Generierung von Testfällen aus UML Zustandsautomaten beschrieben. Es zeigt sich, dass die resultierende Testsuite für modellbasierte Blackbox-Tests keinen intuitiv zufriedenstellenden Abdeckungsgrad bietet. Wir haben skizziert, wie automatisch generierte Testfälle durch zusätzliche Nachbearbeitung kombiniert werden können, um die Abdeckung zu verbessern. Allgemein sind nach unserer Auffassung beim spezifikationsbasierten Test risiko- und szenariogesteuerte Direktiven als Metastrategie für die automatische Testfallerzeugung wünschenswert.

Literatur

[Bin99] Robert V. Binder. *Testing Object-Oriented Systems: Models, Patterns, and Tools*. Object Technology Series. Addison Wesley, 1999.

[Con] Conformiq Software Ltd. Conformiq Test Generator. http://www.conformiq.com/.

[DKvK+02] Christian Denger, Daniel Kerkow, Antje von Knethen, Maricel Medina Mora und Barbara Paech. Richtlinien - Von Use Cases zu Statecharts in 7 Schritten. IESE-Report Nr. 086.02/D, Fraunhofer IESE, 2002.

[GS05] Christophe Gaston und Dirk Seifert. Evalutating Coverage Based Testing. In Manfred Broy, Bengt Jonsson, Joost-Pieter Katoen, Martin Leucker und Alexander Pretschner, Hrsg., *Model-Based Testing of Reactive Systems*, Jgg. 3472 of *Lecture Notes in Computer Science*, Seiten 293–322. Springer Verlag Berlin Heidelberg, 2005.

[HP02] Frank Houdek und Barbara Paech. Das Türsteuergerät - eine Beispielspezifikation. IESE-Report Nr. 002.02/D, Fraunhofer IESE, 2002.

[IL] I-Logix. Rhapsody Automatic Test Generator (ATG). http://www.ilogix.com/.

[Lig02] Peter Liggesmeyer. *Software-Qualität: Testen, Analysieren und Verifizieren von Software*. Spektrum Akadamischer Verlag, 2002.

[LW00] Eckard Lehmann und Joachim Wegener. Test Case Design by Means of the CTE XL. Proceedings of the 8th European International Conference on Software Testing, Analysis & Review (EuroSTAR 2000), Kopenhagen, Denmark, December 2000.

[Pos96] R. M. Poston. *Automating Specification-Based Software Testing*. IEEE Computer Society, Los Alamitos, 1st. Auflage, 1996.

[Rea] Reactive Systems Inc. Reactis. http://www.reactive-systems.com/.