Einsatz von Überdeckungskriterien auf Modellebene – Erfahrungsbericht und experimentelle Ergebnisse –

Mirko Conrad
Daimler Chrysler AG
Mirko.Conrad@DaimlerChrysler.com

Sadegh Sadeghipour
IT Power Consultants
sadegh@itpower.de

Abstract

Die in die elektronischen Steuerungen heutiger Kraftfahrzeuge eingebettete Software wird zunehmend modell-basiert entwickelt. D.h. typischerweise wird frühzeitig im Entwicklungsprozess mit Modellierungs- und Simulationswerkzeugen wie Matlab/ Simulink/ Stateflow ein ausführbares Modell des Steuerungs- und Regelungssystems erstellt, das die Grundlage für die spätere Steuergeräte-Software bildet.

Der den modell-basierten Entwicklungsprozess begleitende Testprozess (modell-basierter Test) kann auf vielfältige Weise vom Vorhandensein eines solchen ausführbaren Modells profitieren und weist daher zahlreiche Spezifika auf. So können z.B. neben den bekannten strukturellen Überdeckungskriterien auf Software-Ebene solche Kriterien auch auf Modell-Ebene erhoben und zur Steuerung der Testtiefe sowie als Testabbruchkriterium verwendet werden.

Da die werkzeuggestützte Erhebung solcher Modell-Überdeckungskriterien erst seit kurzer Zeit einfach möglich ist, existieren bisher nur wenige Erfahrungen mit ihrem Umgang. Der vorliegende Beitrag beschreibt solche im Rahmen einer experimentellen Untersuchung gesammelten Ergebnisse und Erfahrungen beim Einsatz von Modellüberdeckungskriterien für Simulink/Stateflow-Modelle.

Schlüsselwörter: Codeüberdeckung, modell-basierte Entwicklung, modell-basierter Test, Modellüberdeckung, strukturorientierte Überdeckungsmaße, Simulink/ Stateflow, Überdeckungskriterien

1 Einleitung

In der Entwicklung eingebetteter Systeme im Kraftfahrzeug ist seit Ende der 90er Jahre ein Paradigmenwechsel von der klassischen Software-Entwicklung auf Basis textueller Spezifikationen hin zu einem modell-basierten Entwicklungsprozess auf Basis ausführbarer Spezifikationen zu verzeichnen (siehe z.B. [Bec00, RCK+00]). Dabei wird eine ausführbare Funktionsspezifikation (Funktionsmodell) erstellt, die zunächst zusammen mit einem Streckenmodell (Umgebungsmodell) simuliert und später direkt auf dem Steuergerät implementiert werden kann. Ein solches, bereits frühzeitig vorliegendes Funktionsmodell der zu realisierenden Software dient als Basis aller weiteren Entwicklungsschritte. Die Modellierung erfolgt dabei mit Werkzeugen wie Matlab/ Simulink/ Stateflow [MW01] unter Verwendung der aus der Regelungstechnik bekannten Blockschaltbilder und Zustandsübergangsdiagramme.

Im Hinblick auf die Software-Prüfung eröffnet die modell-basierte Entwicklung die Möglichkeit, schon das ausführbare Modell des Systems im Rahmen sogenannter Modelltests einer analytischen Qualitätssicherung zu unterziehen. Dadurch können bestimmte Prüfaktivitäten in frühere Entwicklungsphasen vorverlegt und Fehler so frühzeitig gefunden und kostengünstig korrigiert werden. Die den Modelltests zugrundeliegenden Testszenarien können in späteren Entwicklungsphasen bspw. für den Test der Steuergerätesoftware oder den Hardware-in-the-Loop-Test des Steuergerätes wiederverwendet werden. Ein solcher den modell-basierten Entwicklungsprozess begleitender Test wird als modell-basierter Test bezeichnet; er umfasst eine Kombination unterschiedlicher, sich gut ergänzender Testmethoden.

Zur Steuerung der Testtiefe bzw. des Testendes im Rahmen des modell-basierten Testprozesses können neben den strukturellen Überdeckungsmaßen auf Software-Ebene (Code-Überdeckungsmaße) auch derartige Überdeckungsmaße auf Modellebene (Modell-Überdeckungsmaße) zum Einsatz kommen.

Eine werkzeuggestützte Messung der Modellüberdeckung für Simulink/Stateflow-Modelle ist für einen breiteren Nutzerkreis erst seit dem Erscheinen von Matlab R12, d.h. seit Ende 2000 verfügbar. Erfahrungen im Umgang mit dieser im Vergleich zur Messung der Code-Überdeckung recht jungen Technologie sind nur begrenzt vorhanden. Daher wurden von den Autoren der Einsatz von Modellüberdeckungsmessungen für Simulink/Stateflow-Modelle im Rahmen des modell-basierten Testprozesses experimentell untersucht. Folgende Fragestellungen standen dabei im Mittelpunkt:

- Welche Einsatzmöglichkeiten gibt es für Modellüberdeckungsmessungen im Rahmen der modell-basierten Entwicklung?
- Welche Zusammenhänge bestehen zwischen strukturorientierter Modell- und Codeüberdeckung?
- Können Modellüberdeckungsmessungen die Messung der Codeüberdeckung ergänzen oder teilweise ersetzen?

Der vorliegende Artikel dokumentiert die experimentellen Ergebnisse und die dabei gesammelten Erfahrungen.

2 Strukturorientierte Überdeckungsmaße

Die Bestimmung von strukturorientierten Überdeckungsmaßen auf Code-Ebene (Code Coverage) als Bestandteil des Testprozesses gilt als Best Practice in der industriellen Software-Entwicklung [BY98] und wird von zahlreichen Entwicklungsstandards gefordert (vgl. z.B. [MISRA95]). Überdeckungsmaße auf Code-Ebene können kontroll- (Cx) oder datenflussbezogen (Dx) sein, die Überdeckung wird dabei im Hinblick auf bestimmte Programmelemente gemessen, die während des Test ausgeführt werden. Im Falle des Zweigtests (branch coverage, C1) sind dies beispielsweise Programmzweige.

Strukturorientierte Überdeckungsmessungen auf Modellebene (Model Coverage) werden dagegen allenfalls pilotiert. Hierbei wird die Zahl der durch den Test überdeckten Modellelemente zur Gesamtzahl dieser Elemente ins Verhältnis gesetzt. Im Falle der Zustandsüberdeckung bei Stateflow-Diagrammen sind solche Elemente dann Automatenzustände.

Sowohl Überdeckungsmaße auf Code- als auch auf Modellebene können als Instanzen eines allgemeineren Konzepts von strukturellen Überdeckungsmaßen aufgefasst werden. Die Erhebung struktureller Überdeckungsmaße zur Steuerung der Testtiefe bzw. des Testendes basiert auf der Annahme, dass die Testüberdeckung mit zunehmender Testanzahl steigt, sofern die Testabdeckung noch nicht vollständig ist und die neu hinzugefügten Tests bereits vorhandene Tests nicht wiederholen. Die Definition neuer Tests erfolgt dabei mit dem Ziel, strukturelle Einheiten zu überdecken, die bisher noch nicht getestet wurden. Dieses Auswahlkriterium führt zu einem im Vergleich zum Zufallstest zielgerichteterem und daher effizienterem Testprozess. Die Erreichung einer bestimmten strukturellen Überdeckung wird in der Praxis häufig als notwendiges, aber nicht hinreichendes Testendekriterium verwendet. Eine 100%ige strukturelle Überdeckung kann in der Praxis zumeist nicht erreicht werden, da ein gewisser Anteil der strukturellen Einheiten nicht erreichbar ist. Dieser Anteil kann jedoch vernachlässigt werden (vgl. [MLB+94, BY98]).

Die beiden folgenden Abschnitte beschreiben die bei den experimentellen Untersuchungen benutzten strukturorientierten Überdeckungsmaße auf Modell- und Code-Ebene.

2.1 Überdeckungsmaße auf Modellebene

Das seit Matlab R12 verfügbare Model Coverage Tool [MW01] erlaubt während des Modelltests die Erhebung verschiedener Überdeckungsmaße bei Simulink/Stateflow-Modellen. Dabei wird der Anteil der beim Test durchlaufenen Pfade durch das Modell (*simulation pathways*) zur Anzahl der möglichen Pfade ins Verhältnis gesetzt. In der aktuellen Matlab Release 12.1 können vier verschiedene Überdeckungsmaße auf Modellebene ermittelt werden: Decision Coverage (D1), Condition Coverage (C1), Modified Condition/Decision Coverage (MCDC) und Look-Up Table Coverage (LUT). Für die experimentellen Untersuchungen wurden Decision und Condition Coverage verwendet.

Decision Coverage (D1)

Für die Bestimmung von Decision Coverage (D1) wird die Ausführung von Blöcken, die als Entscheidungspunkte dienen, analysiert. Für jeden Entscheidungspunkt wird die Anzahl der möglichen und tatsächlich ausgeführten Alternativen berechnet. Bei einem Subsystem ergibt sich die Anzahl der Alternativen aus der Summe der im Subsystem enthaltenen Entscheidungspunkte. Zur Decision-Coverage-Bestimmung unter Matlab R12.1 werden u.a. Switch-, Abs-, Logic- und While-Blöcke, Triggered/Enabled Subsysteme sowie Stateflow-Zustände und -Transitionen herangezogen. Abb. 1 zeigt beispielhaft die verschiedenen Pfade durch einen Switch-Block.

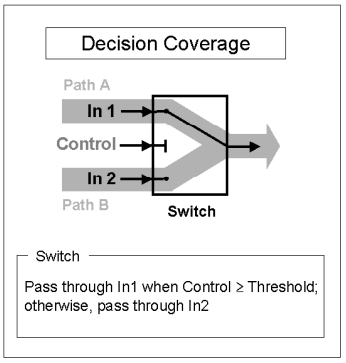

Abb. 1 Pfade durch einen Switch-Block

Condition Coverage (C1)

Für die Bestimmung von Condition Coverage (C1) wird die Überdeckung der logischen Prädikate bei logischen Simulink-Blöcken (z.B. Logic-, Combin. Logic-Blöcke) und Stateflow-Transitionen analysiert. Ein Test erreicht 100%ige C1-Überdeckung auf Modellebene, wenn jeder logische Block und jede Stateflow-Transition mindestens je einmal zu *true* und zu *false* ausgewertet wird.

2.2 Überdeckungsmaße auf Code-Ebene

Während die Verwendung kontrollflussbezogener Überdeckungsmaße (Cx) in der Praxis weit verbreitet ist, kommen datenflussbezogene Überdeckungsmaße (Dx) aufgrund der schwierigeren Automatisierung seltener zum Einsatz. Gängige kontrollflussorientierte Überdeckungsmaße sind Anweisungsüberdeckung

(Statement Coverage, C0), Zweigüberdeckung (C1), Pfadüberdeckung (C4) und verschiedene Varianten der Bedingungsüberdeckung (vgl. z.B. [BC 7925]). Zwischen einzelnen Überdeckungskriterien besteht eine Ordnungsrelation, beispielsweise beinhaltet der Zweigüberdeckungstest den Anweisungsüberdeckungstest (vgl. z.B. [Gri95]).

Kontrollflussbezogene Überdeckungsmessungen auf C-Code können mit verschiedenen Testwerkzeugen ermittelt werden Für die experimentellen Untersuchungen wurden mit dem Testsystem TESSY [Pit99] ermittelte C1-Überdeckungsgrade sowie daraus manuell abgeleitete C0-Überdeckungsgrade verwendet.

Anweisungsüberdeckung / Statement Coverage (C0)

Beim Statement-Test wird das Ziel verfolgt, das Testobjekt so zu testen, dass jede seiner Anweisungen mindestens einmal ausgeführt wird [Gri95]. Das Kriterium Statement Coverage setzt die Zahl der während des Tests ausgeführten Befehle (Statements) zur Zahl der ausführbaren Befehle des Programm-Codes ins Verhältnis.

Zweigüberdeckung / Branch Coverage (C1)

Ziel des Zweig-Tests ist es, jeden Programmzweig des Testobjektes mindestens einmal zu durchlaufen. Unter einem Programmzweig wird dabei, grob definiert, ein möglicher Weg vom Programmanfang bzw. von einer Verzweigung der Kontrollstruktur bis zur nächsten Kontrollstruktur bzw. zum Programmende verstanden. Das Kriterium Zweigüberdeckung, ist als Verhältnis der durchlaufenen Zweige zur Gesamtzahl vorhandener Zweige im Quellcode definiert (vgl. [Gri95], [Pit99]).

3 Untersuchungsgegenstand und -ablauf

Die Tests wurden sowohl auf der Modellebene als auch auf der Code-Ebene durchgeführt. Die Untersuchungsgegenstände (Testobjekte) waren drei verschiedene Subsysteme (Module) eines mit Simulink/Stateflow modellierten Fahrdynamiksystems realer Größe und Komplexität. Zwei der untersuchten Subsysteme enthielten sowohl Simulink- als auch Stateflow-Anteile, das dritte Subsystem ausschließlich Simulink-Anteile. Die Testobjekte wurden unter Verwendung der modellbasierten Testumgebung MTest V1.33 [CDF+99, RCK+00] einem Modelltest unterzogen. Dabei wurden vorhandene, mit Hilfe der Klassifikationsbaum-Methode für eingebettete Systeme [Con01, CTE/ES01] beschriebene Black-Box-Testszenarien zugrunde gelegt, aus denen Testdaten in Form von Werteverläufen über der Zeit gewonnen wurden. In den vorhandenen Testszenarien wurden mehrere inhaltlich zusammenhängender Testsequenzen zu Gruppen (Tests) zusammengefasst.

Tabelle 1 Experimentell ermittelte Modell- und Code-Überdeckungen (Zusammenfassung)

Testobjekt	Test	Modell-Überdeckung [%]		Code-Überdeckung [%]	
		D1	C1	C0	C1
Subsystem 1	Σ Testsequenzen Test 001	49	58	57	54
	Σ Testsequenzen Test 002	37	55	54	43
	Σ Testsequenzen Test 003	25	42	47	33
	Σ Testsequenzen Test 004	49	53	51	48
	Σ Black-Box-Tests	**54**	**66**	**61**	**58**
	Σ Testsequenzen Zufallstests	35	62	55	41
	Σ Gesamt	**62**	**84**	**68**	**58**
Subsystem 2	Σ Testsequenzen Test 001	63	65	68	66
	Σ Testsequenzen Test 002	87	82	87	88
	Σ Black-Box-Tests	**88**	**85**	**88**	**89**
	Σ Testsequenzen Zufallstests	75	85	87	79
	Σ Gesamt	**95**	**90**	**96**	**95**
Subsystem 3	Σ Testsequenzen Test 001	100	69	100	100
	Σ Black-Box-Tests	**100**	**69**	**100**	**100**
	Σ Gesamt	**100**	**69**	**100**	**100**

Ergänzend zu den systematisch ermittelten Black-Box-Testszenarien wurden für die Subsysteme 1 und 2 zufällige Testdaten in Form von Werteverläufen über der Zeit generiert. Die Länge der zufällig erzeugten Testszenarien sind mit denen der systematischen ermittelten vergleichbar. Insgesamt wurden 36 Black-Box- und 8 Zufalls-Testsequenzen in die Untersuchungen einbezogen. Die durchschnittliche Dauer einer Testsequenz betrug dabei 11 Sekunden.

Während des Modelltests wurden die erreichten Decision- und Condition-Coverage-Überdeckungen auf Modellebene mit dem Model Coverage Tool aus Matlab R12.1 bestimmt [MW01]. Im Anschluss an den Modelltest wurde mit Hilfe des Codegenerators TargetLink V1.3 [dS01] Gleitkomma-C-Code für die Testobjekte erzeugt. Die Testdaten aus dem Modelltest dienten als Stimuli für die erneute Durchführung der Tests (SW-Test) mit Hilfe des Testsystems TESSY

V2.018 [Pit99]. Dabei wurden die C0- und die C1-Überdeckung auf Code-Ebene ermittelt.

Tabelle 1 fasst die experimentell ermittelten Überdeckungsgrade der drei Subsysteme auf Modell- und Code-Ebene zusammen. Sie enthält für jedes Subsystem neben den in den einzelnen Tests insgesamt erreichten Überdeckungen auch die Summe der durch die Black-Box-Tests erreichten Überdeckungsgrade und die Gesamtüberdeckung aus Zufalls- und Black-Box-Tests.

4 Analyse der exprimentellen Ergebnisse

In diesem Abschnitt werden die im vorherigen Abschnitt dargestellten Überdeckungsdaten analysiert.

4.1 Erreichte Modell- und Code-Überdeckungen

Während in der Literatur Aussagen zu mit Black-Box-Tests erreichbaren Code-Überdeckungen zu finden sind (vgl. [BY98] und [Ben01]), sind den Autoren entsprechende Angaben für erreichbare Modellüberdeckungen nicht bekannt.

Die in den Experimenten ermittelten (über alle zum jeweiligen Subsystem gehörenden Black-Box-Testsequenzen akkumulierten) C1-Code-Überdeckungsgrade von 58, 89 bzw. 100% (Tabelle 1) liegen in- oder oberhalb der in [MLB+94] angegebenen Bandbreite (40-60%), jedoch bis auf Subsystem 3 unterhalb der von [Ben01] geforderten Überdeckung (90%). Die erreichten Modellüberdeckungen schwankten zwischen 54 und 100% bei der D1- bzw. zwischen 66% und 85% bei der C1-Modell-Überdeckung und liegen damit in vergleichbaren Größenordnungen.

Durch Zufallstests wurden nur geringe Überdeckungsgrade erreicht, jedoch trugen sie zur Erhöhung der Gesamtüberdeckung bei. Das zeigt, dass die Zufallstests die systematisch ermittelten Testszenarien nicht ersetzen sondern sie ergänzen können. Allerdings stellt sich meist die Entscheidung über den Erfolg oder Misserfolg von Tests (Testorakelproblem) bei zufällig ausgewählten Daten als problematisch dar.

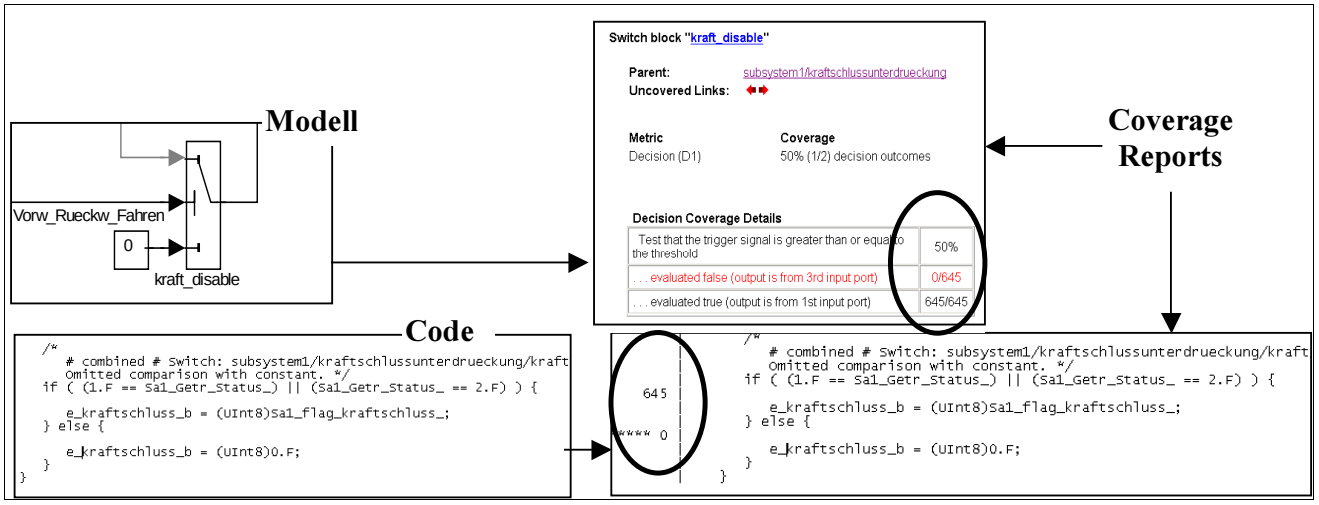

Abb. 2 Vergleich von D1- und C1-Überdeckung auf Modell- und Code-Ebene

4.2 Einschätzung der Überdeckungsmaße auf Modellebene

Die unter Matlab R12.1 bereitgestellten Überdeckungsmaße sind eher kontrollflussorientiert.

Während die allgemein verwendeten Code-Coverage-Kriterien allgemeiner Natur (d.h. werkzeugunabhängig definiert) und auf verschiedene imperative Programmiersprachen der 3. Generation anwendbar sind, sind die durch das Model Coverage Tool bestimmbaren Überdeckungsmaße nach Meinung der Autoren eher toolspezifischer Natur. Argumente hierfür sind zum einen die sehr beispielbezogenen Beschreibungen der Kriterien durch den Werkzeughersteller [MWCov01] und zum anderen die beim Versionswechsel von Matlab R12 auf R12.1 beobachtete Ausweitung der zur Bestimmung der Coverage-Werte herangezogenen Blöcke bei der D1-Modell-Überdeckung. Dies kann zu versionsabhängig unterschiedlichen Überdeckungszahlen beim Decision Coverage auf Modellebene führen. Dieser Zustand wird von den Autoren als nicht befriedigend eingeschätzt.

Das Vorhandensein verschiedener Überdeckungskriterien in Matlab R12.1 erlaubt eine differenziertere Betrachtung der Modellüberdeckungen: Das Kriterium Decision Coverage (D1) ermöglicht einen ersten Eindruck über die Güte der Tests hinsichtlich der Modellüberdeckung. Die weiteren Metriken können dann möglicherweise zur Ermittlung der Ursachen fehlender Überdeckungen beitragen.

4.3 Zusammenhänge zwischen Modell- und Code-Überdeckung

Aufgrund von strukturellen Ähnlichkeiten zwischen Modell und daraus generiertem Code sind gewisse Zusammenhänge zwischen erreichten Modell- und Code-Überdeckungen zu vermuten. Aussagen in der Literatur stützen diese Annahme, bspw. wird in [Har01] Model Coverage als "Analyse von Testabdeckungen während des Debuggens …, was einer Code-Coverage im herkömmlichen Sinne ähnlich ist", charakterisiert.

Bei den Experimenten wurde u.a. eine starke Korrelation zwischen Decision Coverage (D1) auf Modellebene und Branch Coverage (C1) auf Code-Ebene, welches in der Literatur auch Decision Coverage genannt wird (vgl. [MLB+94]), festgestellt. Dieser Zusammenhang lässt sich strukturell erklären, da Code-Zweige an den Entscheidungspunkten entstehen. Deshalb liegen die Zweigüberdeckungswerte des Codes nah an den Entscheidungsüberdeckungswerten des Modells.

Abb. 2 veranschaulicht den Zusammenhang zwischen D1- und C1-Überdeckung auf der Modell- und Code-Ebene am Beispiel eines Switch-Blocks. Der Switch-Block wurde im Zuge der Codegenerierung mit TargetLink durch eine If-Abfrage, d.h. einen Programmzweig, im Code realisiert. Somit berechneten das Model Coverage Tool und TESSY beide die gleiche Anzahl von Ausführungen (645 Mal) für einen der beiden Zweige an diesem Entscheidungspunkt.

Abb. 3 stellt den Zusammenhang zwischen D1- und C1-Überdeckung graphisch dar. Dort sind die D1-Modellüberdeckungsgrade aus Tabelle 1 auf der x-Achse und die entsprechenden C1-Codeüberdeckungsgrade auf der y-Achse eingetragen. Bei Berücksichtigung der einzelnen Sequenzen beträgt die berechnete Korrelation zwischen D1 und C1 0,983 und die Korrelation zwischen D1 und C0 0,973 (nicht in Abb. 3 dargestellt). Die Nähe der Korrelationszahlen zu 1 weist auf den engen, statistischen Zusammenhang zwischen D1-Überdeckung auf der Modellebene und der C1- bzw. C0-Überdeckung auf der Code-Ebene hin.

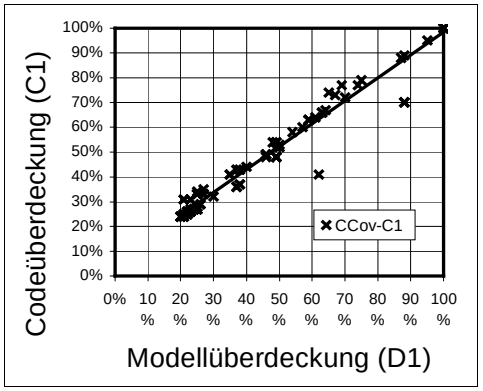

Abb. 3 Zusammenhang zwischen Modell und Code Coverage

Unter den gegebenen Randbedingungen kann also die C1-Code-Überdeckung durch die D1-Modell-Überdeckung angenähert werden. Die D1-Modell-Überdeckung erlaubt hier eine Abschätzung der zu erwartenden Code-Überdeckung.

Der exakte mathematische Zusammenhang variiert in Abhängigkeit von den bei der Codegenerierung verwendeten Übersetzungsalgorithmen. So können unterschiedliche Codegeneratoren bzw. unterschiedliche Codegeneratoreinstellungen zu unterschiedlichen Umsetzungen ein und des selben Modellteils (z.B. einer verschachtelten Abfrage) und damit potentiell zu unterschiedlichen Überdeckungen führen.

Zusammenfassung und Ausblick

Im Rahmen eines Experimentes wurden drei funktionale Module eines mit Simulink/Stateflow modellierten Fahrdynamiksystems einem Modelltest und der daraus generierte C-Code einem Softwaretest unterzogen. Eine Analyse der beim Test gemessenen Modell- und Codeüberdeckungen zeigte, dass vergleichbare Modell- und Codeüberdeckungsmaße existieren. Die Korrelation zwischen diesen Maßen hängt von der Art der (manuellen oder automatischen) Umsetzung des Modells in Code ab.

Die modell-basierte Entwicklung ermöglicht, dass neben den verbreiteten strukturorientierten Überdeckungsmaßen auf Code-Ebene auch Strukturtestkriterien auf Modell-Ebene ermittelt und in die Steuerung und Bewertung des Testprozesses einfließen können. Diese Überdeckungsmaße können bereits frühzeitig im Entwicklungs- und Testprozess, d.h. bevor C-Code vorhanden ist, erhoben werden. Der Vorteil dieser Vorgehensweise besteht in der Vorverlagerung der entsprechenden Testaktivitäten und folglich der frühzeitigen Fehlerentdeckung und kostengünstigen Fehlerkorrektur.

Die Modell-Überdeckungen können (im Hintergrund) bei der Durchführung der funktionalen Modelltests erhoben werden. Der Tester kann durch eine grobe Analyse der erzeugten Coverage-Reports feststellen, welche Funktionalitäten nicht geprüft wurden. Zur Abdeckung dieser Funktionalitäten müssen neue Testsequenzen erstellt werden. Eine genaue Analyse der Coverage-Reports bezüglich einzelner, zu überdeckenden Blöcke und unterschiedlicher Metriken könnte weitere Hinweise zur Definition neuer Testsequenzen und Auswahl von Testdaten geben.

Zu erreichende Modell-Überdeckungen können (neben anderen Kriterien) zur Steuerung des Testumfangs und als (notwendiges) Testendekriterium für den funktionalen Modelltest verwendet werden. Unter bestimmten Umständen erlauben sie eine Abschätzung der zu erwartenden Code-Überdeckung und können zukünftig für die mittels Codegeneratoren erzeugten Programmteile möglicherweise bestimmte Strukturtestkriterien auf Code-Ebene ersetzen bzw. ergänzen.

Ziel weiterer Untersuchungen sollte es sein, die aus dem klassischen Software-Test bekannte effektive Teststrategie (vgl. [Gri95]) an die Besonderheiten des modell-basierten Tests und der Möglichkeit der Erhebung von Strukturtestkriterien auf Modell-Ebene anzupassen. In diese effektive Teststrategie für den modell-basierten Test sollten auch neuere Entwicklungen im Bereich der Verifikations- und Testwerkzeuge einfließen. Bspw. können mit Model-Checkern und evolutionären Algorithmen automatisiert Testdaten zur Erreichung bisher nicht überdeckter Modell- oder Code-Teile ermittelt bzw. die Nichterreichbarkeit bestimmter Strukturelemente untermauert oder gar bewiesen werden.

Danksagung

Die Experimente fanden im Rahmen der Entwicklung und Erprobung der modell-basierten Testumgebung MTest statt und wurden durch Diskussionen innerhalb des Projektes Modell-basierte Entwicklung (mode.de) befruchtet. Allen Beteiligten, insbesondere jedoch Ines Fey, Matthias Grochtmann, Wolfram Mendt, Torsten Klein und Hans-Werner Wiesbrock, gilt der herzliche Dank der Autoren.

Literatur

[Bec00] P. Bechberger: Modellbasierte Software-Entwicklung für Steuergeräte. Automobiltechnische Zeitschrift/ Motortechnische Zeitschrift (ATZ / MTZ), Sonderausgabe "Automotive Electronics", Jan. 2000.

[Ben01] Richard Bender: How Do You Know When You Are Done Testing? Proc. of 2nd Int. Conf. on Software Testing (ICS TEST '01), Bonn, 4.-6.4.2001.

[BS 7925] Standard for Software Component Testing. Working Draft 3.4. British Computer Society, Specialist Interest Group in Software Testing (BCS SIGIST), 2001.

[BY98] K. Burr, W. Young: Combinatorial Test Techniques: Table-Based Automation, Test Generation, and Code Coverage. International Conference on Software Testing, Analysis, and Review. CA, 1998.

[Con01] M. Conrad: Beschreibung von Testszenarien für Steuergeräte-Software - Vergleichskriterien und deren Anwendung. 10. Internationaler Kongress Elektronik im Kraftfahrzeug, Baden-Baden, Germany, 2001.

[CDF+99] M. Conrad, H. Dörr, I. Fey, A.Yap: Model Based Generation and Structured Representation of Test Scenarios. Proc. of Workshop the Workshop on Software-Embedded Systems Testing (WSEST '99), Maryland, USA, 1999.

[CTE/ES01] Classification Tree Editor for Embedded Systems – CTE/ES, Benutzerhandbuch, Razorcat Development GmbH, ISBN 3-8311-1848-5, Berlin, 2001. CTE/ES at Razorcat Berlin: www.razorcat.com

[dS01] *dSpace, The*: TargetLink Production Code Generation Guidde, dSpace Inc., www.dspaceinc.com , 2001.

[Gri95] K. Grimm: Systematisches Testen von Software – Eine neue Methode und eine effektive Teststrategie. Dissertation, GMD Bericht Nr. 251, München/Wien 1995.

[Har01] B. Hartmann: Turbo für die SW-Entwicklung - Zeit sparen durch graphisches Programmieren. Design & Elektronik, Heft 1, Januar 2001, S. 106-108

[MLB+94] Y. K. Malaiya et. al.: The Relationship between Test Coverage and Reliability, Proceedings of the Fifth International Symposium on Software Reliability Engineering, 1994.

[MISRA95] The Motor Industry Software Reliability Association (MISRA): Development Guidelines For Vehicle Based Software, 2. Aufl., 1995.

[MW01] *MathWorks, The*: Matlab – Using Simulink. The MathWorks, Inc., www.mathworks.com, 2001.

[MWCov01] *MathWorks, The*: Simulink Model Coverage, www.mathworks.com/products/slperftools/, 2001.

[Pit99] R. Pitschinetz: Das Testsystem TESSY Version 2.5. Technischer Bericht FT3/S-1999-001. DaimlerChrysler AG, Berlin 1999.

[RCK+00] A. Rau, M. Conrad, H. Keller, I. Fey, C. Dziobek: Integrated Model-based Software Development and Testing with CSD and MTest. and MTest. Proc. of International Automotive Conference 2000 (IAC'00), Stuttgart, 2000. www.it.fht-esslingen.de/~rau/forschung/iac2000.pdf