Modellbasiertes Testen

Ina Schieferdecker
TU Berlin/Fraunhofer FOKUS

1 Motivation

Erste Grundlagen für das modellbasierte Testen unter Nutzung von endlichen Automaten wurden bereits in den 60er Jahren erarbeitet, doch erst jetzt erfahren diese Methoden eine zunehmende Verbreitung, da mittlerweile Modellierungstechniken – eine Grundvoraussetzung für modellbasiertes Testen – in der industriellen Softwareentwicklung verstärkt genutzt werden. Dazu hat nicht zuletzt die Verabschiedung der UML 2.0 (Unified Modelling Language Version 2.0 der Object Management Group) in 2005 und die Propagierung eines modellbasierten Systementwicklungsprozesses genannt MDA (Model Driven Architecture) seit 2000 beigetragen. Interessanterweise geben dabei UML und MDA die Richtung für eine modellbasierte Systementwicklung vor, werden dabei aber nicht explizit bzgl. der Testmethoden und –verfahren. Insofern waren zwei Schritte [1] relativ nahe liegend – wenn gleich langwierig und arbeitsintensiv: Die Entwicklung einer Erweiterung von UML für den Entwurf und die wohl definierte Beschreibung Test Suiten mit Mitteln der UML – und die Erweiterung der MDA um explizite Testartefakte und Testprozessschritte.

2 Überblick

Modellbasiertes Testen kann sowohl für statische als auch dynamische Tests genutzt werden. Es empfiehlt sich für manuelle und werkzeuggestützte, automatisierte Verfahren, wobei bei letzterem eine höhere Effizienz erzielt wird.

Beim statischen Testen werden die Informationen aus dem Systemmodell wie die Systemstruktur und Systemschnittstellen, die Vielfachheit von Systemkomponenten, ihre Relationen untereinander, etc. – im wesentlichen Strukturelemente geprüft. Das Systemmodell (oder Teile davon) wird dabei als Menge von Regeln interpretiert, denen das System entsprechen muss, siehe beispielsweise [3].

Häufiger werden jedoch modellbasierte Verfahren für dynamische Tests genutzt [4]. Hier unterteilen sich die Verfahren in aktive und passive Tests. Bei ersteren werden Stimuli an das System angelegt und die Reaktionen beobachtet und analysiert. Beim passiven Testen werden die Traces der Systemausführung aufgezeichnet und gegen die Spezifikation verglichen. Für das aktive Testen werden Testfälle aus den Daten-, Struktur- und Verhaltensinformationen der Systemmodelle gewonnen und auf das System angewendet. Für das passive Testen werden System-Invarianten und/oder Zustandsbedingungen, die im Systemmodell vorgegeben sind, entlang der Traces (vor- bzw. rückwärtsgerichtet) analysiert.

Abb. 1 stellt die Relationen zwischen System und Testsystem und ihren Modellen dar: die Anforderungen repräsentieren – aus verschiedenen Perspektiven – sowohl das System als auch das Testsystem und – auf verschiedenen Abstraktionsstufen – deren Modelle. Andererseits realisieren System, Testsystem und deren Modelle die Anforderungen. Dabei stehen System und Testsystem dual zueinander: während das Testsystem für die Validierung der Anforderungen im System entwickelt wird, dient auch das System zur Validierung des Testsystems. Gleiches gilt auf der Modellebene – und zeigt eine zusätzliche Validierungsmöglichkeit unter Nutzung des Testmodells auf: das Testmodell kann zur frühzeitigen Validierung des Systemmodells genutzt werden.

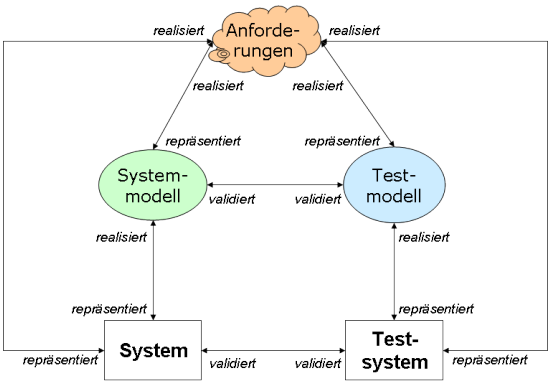

Abb. 1. Relationen zwischen Test und System

3 Varianten modellbasierten Testens

Wie in Abb. 2 dargestellt, gibt es nicht nur eine Möglichkeit, modell-basierte Entwicklungsprozesse unter Nutzung von System- und/oder Testmodellen aufzubauen, sondern verschiedene Varianten.

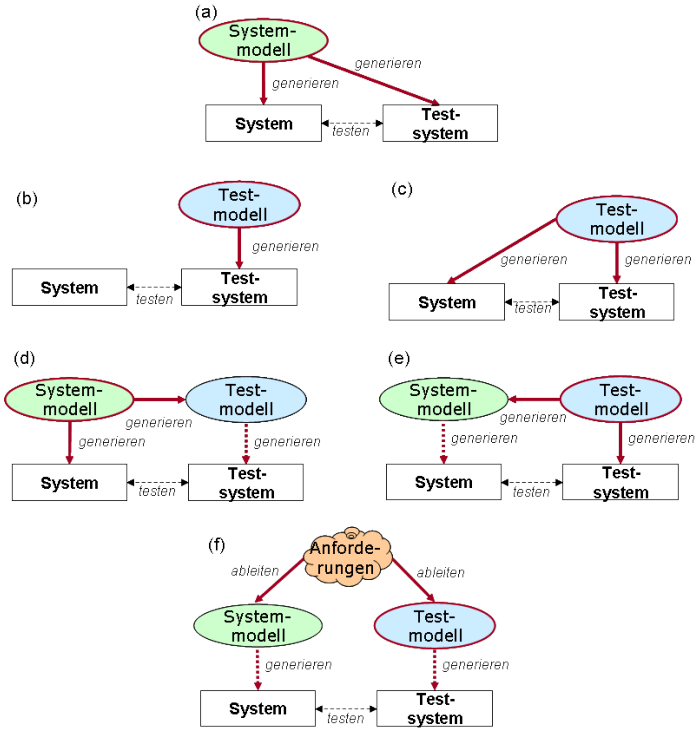

Abb. 2. Varianten modell-basierten Testens

- **systemmodell-getrieben: (a)**

Ein weit verbreiteter Ansatz ist die alleinige Nutzung eines Systemmodells (oftmals in UML, aber auch in MSC (Message Sequence Charts), temporalen Logiken, Petri Netzen, etc.), aus dem nicht nur Teile des bzw. das System, sondern auch Teile des bzw. das Testsystem generiert werden. Eine Methode ist hierbei das sogenannte On-the-fly-Testen, bei dem in Analogie zum On-the-Fly-Model Checking nicht erst der Zustandsraum des Systems erzeugt wird, um dann die Tests abzuleiten, sondern der Zustandsraum dynamisch exploriert wird und das Systemmodell selber als Testtreiber genutzt wird [2]. Andere Methoden erzeugen Testcode direkt aus dem Systemmodell – so z.B. für Integrationstest aus Kollaborationen [3] oder für Unittests aus Zustandsautomaten [5].

Vorteil dieses Verfahrens ist es, dass nur ein Modell genutzt und somit der Modellierungsaufwand reduziert wird und Inkonsistenzen zwischen System und Test verringert werden. Jedoch liegt darin auch ein großer Nachteil: da System und Test aus einer Quelle erzeugt werden, fehlt die Unabhängigkeit von System und Test – die Tests werden Fehler, die bereits im Modell enthalten sind, nicht erkennen können. Auch wird sich die Testsicht nicht von der Modellsicht unterscheiden – nur was im Systemmodell berücksichtigt wurde, kann mit den Tests analysiert werden.

- **testmodell-getrieben: (b) und (c)**

Bei der Nutzung eines eigenständigen Testmodells kommen Testmodellierungstechniken – proprietäre oder standardisierte – zum Einsatz. Standardisierte Techniken sind hierbei das UML 2.0 Testing Profile U2TP [6] (ein OMG Profil der UML) und die Testing und Test Control Notation TTCN-3 [7](ein ETSI und ITU-T Standard). U2TP hat den Vorteil, dass es Elemente und Beschreibungsmittel des Systemmodells wieder verwenden kann, dass also System- und Testmodell in einer gemeinsamen Sprachfamilie definiert sind. Zudem erbt U2TP die Möglichkeit von UML, Modelle – hier Testmodelle – schrittweise zu entwerfen und zu entwickeln. TTCN-3 andererseits hat den Vorteil der durchgängigen Automatisierung der Testausführung auf lokalen oder verteilten Testplattform. Es bietet ebenso wie U2TP eine graphische Darstellung der Tests, erfordert aber eine separate Einarbeitung (natürlich erfordert auch U2TP eine Einarbeitung, wobei aber die über UML hinausgehenden Konzepte von U2TP im Umfang überschaubar sind).

Die Variante (c) ist in Analogie zu (a) eine mögliche Variante, die aber wie (a) an der Singularität des Modells krankt. So wird sie in [8] betrachtet, aber nicht in den Vordergrund gestellt. Verschreibt man sich aber den zwei Prinzipien „test-first“ und „model-basiert“, dann ist dies eine mögliche Variante, die jedoch eine bessere Ausprägung in (e) erfährt.

- **system- und testmodell-getrieben: (d), (e) und (f)**

Die Varianten (d), (e) und (f) setzen die Modellbasierung am konsequentesten um: sowohl für die System- als auch für die Testseite werden Modelle genutzt. Nach der jeweiligen Ableitung/Generierung der Modelle können diese ausgebaut und verfeinert werden. Der Prozess kann systemmodellgetrieben (d), testmodellgetrieben (e) oder weitestgehend separiert (f) sein, um die unabhängigen Sichten der Designer/Entwickler und der Testdesigner/Testentwickler zu wahren.

Die meisten Arbeiten betrachten Variante (d), wobei oftmals automatenbasierte Methoden zum Einsatz kommen [10]. Variante (e) wurde zwar von Firmen wie Telelogic oder IBM propagiert, aber meines Wissens nach nicht weiter analysiert. Variante (f) ist die auch nach [11] optimale Variante, die in [9] im Rahmen der MDA exemplarisch ausgeführt wird.

Vorteile dieser Varianten liegen in der Nutzung zweier Modelle, die bereits vor der Implementierung des Systems/Testsystems auf Korrektheit an sich, auf Konsistenz untereinander als auch auf Überdeckungsmaße hin untersucht werden können. Der anscheinende Nachteil, dass soviel in ein System- als auch in ein Testmodell investiert werden muss, wird durch die erhöhte Effizienz (Unterstützung bei der Codegenerierung, bei der Wartung, beim Umgang mit Systemvarianten und -konfigurationen) wettgemacht. Nicht zu letzt steigt die schlussendliche Systemqualität.

4 Erzeugung von Testmodellen aus Systemmodellen

Betrachtet man die UML als prominenten Vertreter für die Systemmodellierung und die verschiedenen Diagramarten der UML, so kann man mittlerweile für jede Diagrammart eine Generierungsmethode finden, mit der Tests für die Unit-, Komponenten, Integrations- oder Systemebene erzeugt werden. Tabelle 1 gibt einen Überblick zur vorrangigen Nutzung von UML Diagrammen zum Testen. Diese Tabelle zeigt die Schwerpunkte der Nutzung, hat aber keinen Anspruch auf Vollständigkeit– man kann wohl bei geänderter Perspektive fast jedes Kreuz in dieser Tabelle setzen.

Tabelle 1. Nutzung von UML Diagrammen zum Testen

…-diagramm der UML	**Teststufe**				**Testart**		**Testverfahren**	
	Unit	Komponente	Integration	System	funktional	nicht-funktional	Statisch	dynamisch
Struktur								
Klassen- (class)	×	×	×	×	×		×	×
Kompositionsstruktur- (composite structure)		×	×		×		×	×
Komponenten- (component)		×	×	×	×		×	×
Verteilungs- (deployment)			×	×	×		×	
Objekt- (object)		×	×		×		×	×
Paket- (package)			×		×		×	
Verhalten								
Anwendungsfall- (use case)			×	×	×	×[1]	×	×
Aktivitäts- (activity)			×	×	×	×[1]	×	×
Sequenz- (sequence)			×	×	×	×[1]	×	×
Kommunikations- (communication)		×	×		×	×[1]	×	×
Interaktionsübersichts- (interaction overview)			×	×	×	×[1]	×	×
Zeitverlaufs- (timing)		×	×			×[1]	×	×
Zustands- (state machine)	×	×	×	×	×	×[1]	×	×

[1]Oftmals unter Nutzung von UML Erweiterungen (Profilen) für nichtfunktionale Konzepte – bspw. dem Schedulability, Performance and Time Profil (SPT) oder dem Quality-of-Service and Fault Tolerance (QoS/FT) Profil

5 Methoden

Die wesentlichen Methoden beim modellbasierten Testen sind in Abb. 3 dargestellt – die Methoden auf dem Testmodell sind in (a) gezeigt – und die zwischen System- und Testmodell in (b).

Das Testmodell kann schrittweise verfeinert werden z.B. durch Hinzunahme von Testschritten, Parametrierung von Testfällen, Detaillierung von Testdaten, Parametrierung von Testdaten, oder Umstrukturierung der Test Suite. Dabei kann nach jedem Verfeinerungsschritt die Korrektheit der Test Suite geprüft werden. Hier kann die „klassische“ Korrektheit von Modellen/Programmen geprüft werden, wie z.B. dass alles was benutzt wird auch definiert ist, dass Dinge nicht mehrfach definiert sind, etc. Zudem können testspezifische Aspekte wie das Setzen von Testergebnissen (Verdicts) je Testfall oder das Abfangen ausbleibender Systemreaktionen durch Zeitgeber (Timer) geprüft werden.

Steht ein Systemmodell zur Verfügung, so kann zudem die Konsistenz zwischen Testmodell und Systemmodell geprüft werden. Die Konsistenz bezieht sich auf strukturelle Aspekte wie die Übereinstimmung von System- und Testschnittstellen, das Enthaltensein der positiven Testabläufe im Systemmodell genauso wie das Fehlen der negativen Testabläufe, die Eindeutigkeit der Tests bei der Bewertung der Systemreaktionen, etc.

Aus einem Testmodell wird der Code für die ausführbaren Tests i.allg. unter Nutzung von Testplattformen (ggfs. unter Nutzung von Testgeräten) erzeugt, d.h. dass sich das Testsystem aus dem aus dem Testmodell erzeugten Code, aus dem Code der Testplattform (auch Laufzeitumgebung der Tests) und den Testgeräten (spezialisierte Testgeräte oder aber allgemeine Computer) zusammensetzt.

Beim Testen des Systems kann eine gekoppelte oder entkoppelte Verknüpfung von System und Testsystem erfolgen. Entweder wird der Systemcode mit dem Testsystemcode direkt verbunden, oder aber die beiden interagieren über kommunizierende Schnittstellen, was eine Verteilung der Testkonfiguration oder entferntes Testen ermöglicht. Bei entkoppelter Verknüpfung wird zudem der Einfluss des Testsystems auf das System reduziert. Auch wenn diese Betrachtungen auf den ersten Blick unabhängig vom modellbasierten Testen sind, so kann aber für eine gewählte Testmodellierungstechnik eine generische, adaptierbare, wieder verwendbare Testplattform erstellt werden. Solch eine Testplattform kann allgemeine Konzepte des Testens umsetzen (in Analogie zur Umsetzung von Verteilungs- und Kommunikationsprinzipien durch eine Middleware oder zur Umsetzung von Datenstrukturierungs- und –zugriffskonzepten durch Datenbanken) und so den Aufwand bei der Testrealisierung minimieren. Das wird erfolgreich mit TTCN-3 praktiziert – auch bei U2TP sind ähnliche Ergebnisse zu erwarten – z.B. mittels einer Ausführung von U2TP Tests über TTCN-3 Plattformen.

Schlussendlich – und im Zentrum der aktuellen Forschung – steht aber die Generierung von Testmodellen aus Systemmodellen. Während die Kontrollstrukturen des Systemmodells wie sie im Automaten, in Sequenz- oder Aktivitätsdiagrammen beschrieben sind, bereits relativ gut berücksichtigt werden, stellt die Behandlung von Systemdaten noch ein Problem dar. Z.B. sind gängige Verfahren zum Testdatenentwurf wie die Klassifikationsbaummethode, Grenzwertanalyse, Dateninvarianten noch nicht gut mit den Verhaltensmodellen verknüpft.

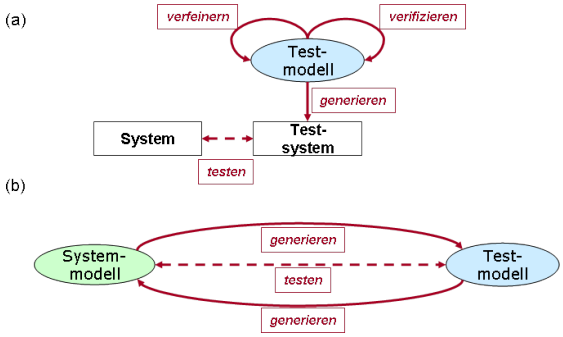

Abb. 3. Methoden modell-basierten Testens

Bei der Testgenerierung aus Modellen können wie bei strukturbasierten Tests Überdeckungsmaße (bei Automaten beispielsweise Zustands-, Transitions-, Pfadüberdeckung, etc.) und andere Metriken (wie Testschrittverzahnung, Testparametrierung) genutzt werden, um die Güte der Tests zu bewerten und Testendekriterien zu definieren – siehe bspw. [13].

6 Ausblick

Dieser Artikel gibt einen klassifizierenden Überblick über existierende Methoden modell-basierten Testens. Auf Grund der Fülle der Arbeiten kann nur eine repräsentative Auswahl wiedergeben werden – eine etwas anders ausgerichtete Taxonomie findet sich unter [12]. Einen weiterführenden Einblick erhält man ebenso über Konferenzreihen wie TestCom (Testing of Communication-Based Systems), FATES (Formal Approaches to Testing of Software), MBT (Model-Based Testing) u.a.

Für einen industriellen Durchbruch von MBT ist neben der Verfügbarkeit von Werkzeugen eine Einbindung von MBT in die Prozesse, die Ausbildung der Tester und der weitere Ausbau der Werkzeuge nötig. Dazu das europäische Projekt D-MINT (Deployment of Model-Based Technologies to Industrial Testing, 2007-2010, www.d-mint.org) aus dem ITEA (Information Technology for European Advancement) Programm mit Partnern aus Deutschland, Finnland, Frankreich, Holland, Irland, Schweden und Spanien Beiträge geliefert.

Interessanterweise setzen Industriekonsortien bei der Entwicklung von Technologien und Systemen in zunehmendem Maße auf modellbasierte Verfahren. Dazu zählen ETSI (European Telecommunication Standards Institute), dass für die Telekommunikation seit langem sowohl Systemmodelle als auch Testspezifikationen für Protokolle und Dienste erarbeitet; die AUTOSAR (Automotive Open System Architecture) Initiative, die konsequent einen UML-basierten Ansatz für die Plattformentwicklung verfolgt; als auch ESA (European Space Agency), die zunehmend Testspezifikationen erarbeitet.

Eine zunehmende Rolle wird hier sicher auch die Test-Terminologie spielen, die durch das ISTQB (International Software Testing Qualification Boards) – in Deutschland durch das GTB (German Testing Board) vertreten – erarbeitet und im Rahmen von Ausbildungsschemata zum Certified Tester vermittelt wird. Noch spielen modellbasierte, modellgetriebene und/oder spezifikationsgetriebene Testverfahren nur eine untergeordnete Rolle, doch finden sich auszugsweise Grundprinzipien modellbasierter Testverfahren in den Lehrplänen. Ein weiterer Ausbau – z.B. mittels eines separaten Moduls für modellbasierte Testverfahren – ist für die Expertenebene des Tester-Syllabus angedacht.

Literatur

[1] P. Baker, Z. R. Dai, J. Grabowski, O. Haugen, I. Schieferdecker, C. Williams: Model-Driven Testing: Using the UML Testing Profile, Springer, 2007, ISBN 3540725628, 9783540725626.

[2] T Jeron, P Morel: Test generation derived from model-checking, 11th Intern. Conf. on Computer Aided Verification, 1999.

[3] A. Abdurazik, J. Offutt: Using UML Collaboration Diagrams for Static Checking and Test Generation. « UML» Conference, Springer LNCS 1939, 2000.

[4] L. Briand, Y. Labiche: A UML-Based Approach to System Testing. Software and Systems Modeling (2002) 1: 10–42.

[5] Y.G. Kim, et al: Test cases generation from UML state diagrams. IEE Proceedings Software. Vol. 146. No. 4, 1999.

[6] I Schieferdecker: The UML 2.0 Test Profile as a Basis for Integrated System and Test Development: GI Jahrestagung, Informatik 2005, Bonn, Sept. 2005.

[7] J. Grabowski, et al: An Introduction into the Testing and Test Control Notation (TTCN-3). - Computer Networks Journal, Vol.42, Issue 3, 2003

[8] K Beck: Test-Driven Development: By Example, Addison-Wesley, 2002.

[9] M. Busch, et al: Model Transformers for Test Generation from System Models, Conquest 2006, Hanser Verlag, September 2006, Berlin, Germany.

[10] S Fujiwara, G Bochmann et al: Test selection based on finite state models.IEEE Transactions on Software Engineering, 1991.

[11] A. Pretschner, J. Philipps: 10 Methodological Issues in Model-Based Testing, Springer LNCS 3472, 2005.

[12] M. Utting, A. Pretschner, B. Legeard: A taxonomy of model-based testing. Working Paper: 04/2006, The University of Waikato, 2006.

[13] B. Zeiss et al: Applying the ISO 9126 Quality Model to Test Specifications - Exemplified for TTCN-3 Test Specifications, Akzeptiert für Software Engineering Konferenz, Hamburg, 2007

Links

MDA	www.omg.org/mda
UML	www.omg.org/uml
U2TP	www.fokus.fraunhofer.de/go/u2tp
ISTQB	www.istqb.org
TTCN-3	www.ttcn-3.org
VSEK Studie	http://www.software-kompetenz.de/?28616

Werkzeuge

Conformiq Test Generator	http://www.conformiq.com/ctg.php
TPTP	http://www.eclipse.org/tptp/
Testing Technologies TTmodeler	http://www.testingtech.com/products/ttplugins_modeler.php
Smartesting Test Designer	http://smartesting.com/index.php/cms/en/explore/products
Conformiq Qtronic:	http://www.conformiq.com/qtronic.php

SpecExplore	http://research.microsoft.com/specexplorer/
TGV	http://www.irisa.fr/pampa/VALIDATION/TGV/TGV.html
TORX	http://fmt.cs.utwente.nl/tools/torx/introduction.html
AutoFocus	http://autofocus.in.tum.de/index-e.html