mehr zum thema:
www.ets.tu-berlin.de
www.fokus.fraunhofer.de/motion

MODELLBASIERTES TESTEN

Modellbasiertes Testen ist ein Schlagwort, unter dem verschiedenste Techniken zur Nutzung von Modellartefakten im Testprozess verstanden werden. Vor allem aber stellt der Begriff heraus, dass im Zuge der modellbasierten Systementwicklung entlang MDA und anderen Methoden ebenso modellbasierte Techniken zur Generierung, Gütebewertung, Ausführung und Wartung von Testartefakten genutzt werden sollten, um eine Qualitätssteigerung und Kostenreduktion beim Testen zu erreichen. Der Artikel stellt eine Klassifikation und Bewertung bestehender Methoden vor.

die autorin

Ina Schieferdecker (E-Mail: Ina.Schieferdecker@fokus.fraunhofer.de) ist Professorin für Entwurf und Testen von kommunikationsbasierten Systemen an der Technischen Universität Berlin und leitet bei Fraunhofer FOKUS das Kompetenzzentrum Modellieren und Testen.

Es fällt nicht schwer, Referenzen zu finden, die die Problematik fehlender Systematik für das Testen von Software hervorheben. Beispielhaft sei auf die VSEK-Studie zu erfolgreichen und gescheiterten Soft- und Hardwareprojekten verwiesen (vgl. [VSEK]). Ebenso fällt es nicht schwer, eine Systematik für das Softwaretesten zu finden. Auch wenn erste Grundlagen für das modellbasierte Testen unter Nutzung von endlichen Automaten bereits in den 60er Jahren erarbeitet wurden, haben diese Methoden bisher noch keine weite Verbreitung gefunden. Hier ist sicher zu berücksichtigen, dass Modellierungstechniken – eine Grundvoraussetzung für modellbasiertes Testen – erst seit relativ kurzer Zeit in der industriellen Softwareentwicklung verstärkt genutzt werden. Dazu haben nicht zuletzt die Verabschiedung der *UML 2.0 (Unified Modeling Language Version 2.0)* der Object Management Group in 2005 (vgl. [OMG-b]) und die Propagierung eines modellbasierten Systementwicklungsprozesses, der *MDA (Model Driven Architecture)*, seit 2000 (vgl. [OMG-a]) beigetragen. Interessanterweise geben dabei UML und MDA die Richtung für eine modellbasierte Systementwicklung vor, werden dabei aber nicht explizit bezüglich der Testmethoden und -verfahren. Insofern waren zwei Schritte relativ nahe liegend, wenngleich langwierig und arbeitsintensiv: die Entwicklung einer Erweiterung der UML für den Entwurf und die wohl definierte Beschreibung von Test-Suiten mit Mitteln der UML sowie die Erweiterung der MDA um explizite Testartefakte und Testprozessschritte.

Einordnung modellbasierten Testens in die Testverfahren

Eine prinzipielle Einordnung des modellbasierten Testens in Testverfahren zeigt Abbildung 1. Modellbasiertes Testen kann sowohl für statische als auch für 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 usw. (im Wesentlichen Strukturelemente) geprüft. Das Systemmodell (oder Teile davon) wird dabei als Menge von Regeln interpretiert, denen das System entsprechen muss (vgl. z. B. [Abd00]).

Häufiger werden jedoch modellbasierte Verfahren für dynamische Tests genutzt (vgl. [Bri02]). 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 mit der 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 Systeminvarianten und/oder Zustandsbedingungen, die im Systemmodell vorgegeben sind, entlang den *Traces* (vorwärts bzw. rückwärts gerichtet) analysiert.

Abbildung 2 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

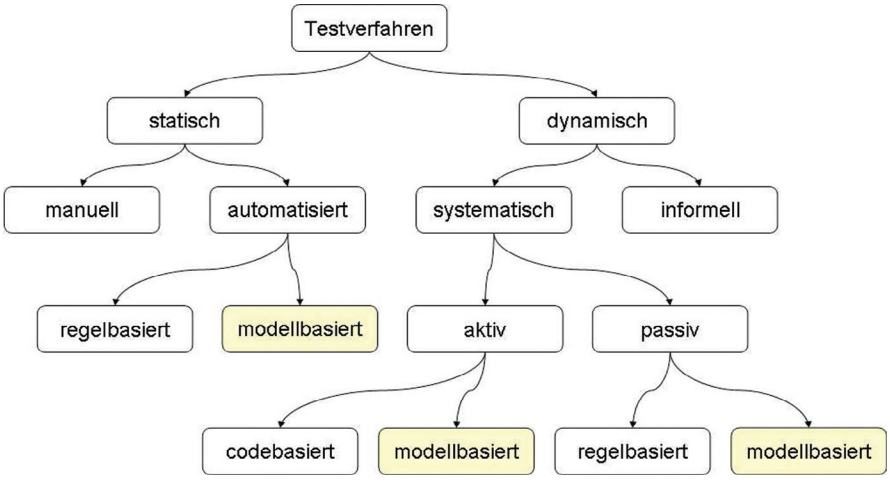

Abb. 1: Einordnung modellbasierter Testverfahren

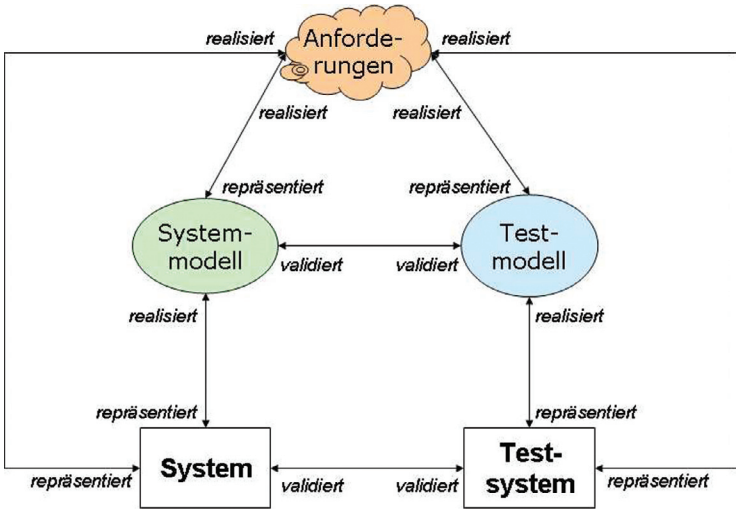

Abb. 2: Relationen zwischen Test und System

zusätzliche Validierungsmöglichkeit unter Nutzung des Testmodells auf: Das Testmodell kann zur frühzeitigen Validierung des Systemmodells genutzt werden.

Varianten modellbasierten Testens

Wie in Abbildung 3 dargestellt, gibt es nicht nur eine Möglichkeit, modellbasierte Entwicklungsprozesse unter Nutzung von System- und/oder Testmodellen aufzubauen, sondern verschiedene Varianten.

Variante (a): systemmodell-getrieben
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 das Systeme bzw. Teile davon, sondern auch das Testsysteme bzw. Teile davon generiert werden. Eine Methode ist hierbei das so genannte *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 (vgl. [Jer99]). Andere Methoden erzeugen Testcode direkt aus dem Systemmodell – so z.B. für Integrationstests aus Kollaborationen (vgl. [Abd00]) oder für Unit-Tests aus Zustandsautomaten (vgl. [Kim99]).

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 können Fehler, die bereits im Modell enthalten sind, nicht erkennen. Auch wird sich die Testsicht nicht von der Modellsicht unterscheiden – nur, was im Systemmodell berücksichtigt wurde, kann mit den Tests analysiert werden.

Varianten (b) und (c): testmodell-getrieben
Bei der Nutzung eines eigenständigen Testmodells kommen proprietäre oder standardisierte Testmodellierungstechniken zum Einsatz. Standardisierte Techniken sind hierbei:

- das *UML 2.0 Testing Profile (U2TP)*, ein OMG Profil der UML (vgl. [Sch05])
- die *Testing und Test Control Notation (TTCN-3)*, ein ETSI und ITU-T-Standard (vgl. [Gra03])

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 von UML die Möglichkeit, Modelle – hier Testmodelle – schrittweise zu entwerfen und zu entwickeln. TTCN-3 hat andererseits den Vorteil der durchgängigen Automatisierung der Testausführung auf lokalen oder verteilten Testplattformen. 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 [Bec02] betrachtet, aber nicht in den Vordergrund gestellt. Verschreibt man sich aber den zwei Prinzipien „test-first" und „modellbasiert", dann ist dies eine

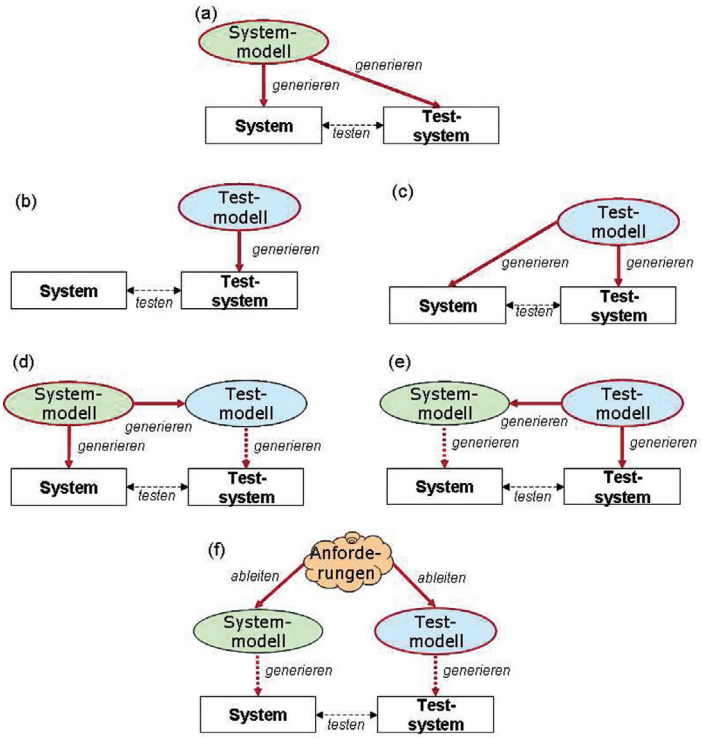

Abb. 3: Varianten modellbasierten Testens

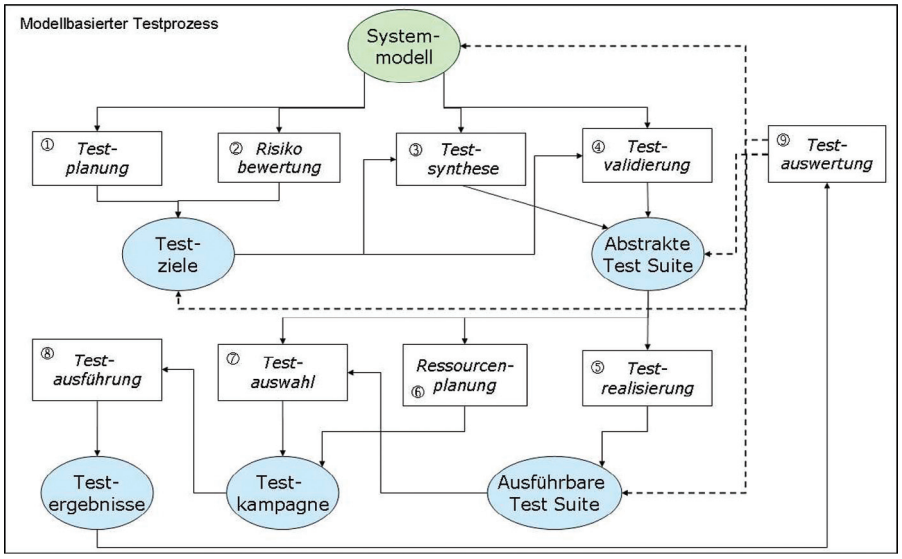

Abb. 4: Elemente eines Testmodells

mögliche Variante, die jedoch eine bessere Ausprägung in (e) erfährt.

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

Die Varianten (d), (e) und (f) setzen die Modellbasierung am konsequentesten um, d. h. 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 systemmodell-getrieben (d), testmodell-getrieben (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 (vgl. [Fuj91]). Variante (e) wurde zwar von Firmen wie Telelogic oder IBM propagiert, aber meines Wissens noch nicht weiter analysiert. Variante (f) ist die auch nach [Pre05] optimale Variante, die in [Bus06] im Rahmen der MDA exemplarisch ausgeführt wird.

Vorteile dieser Varianten liegen in der Nutzung zweier Modelle, die bereits vor der Implementierung des Systems/Testsystems sowohl auf Korrektheit an sich und auf Konsistenz untereinander als auch auf Überdeckungsmaße hin untersucht werden können. Der anscheinende Nachteil, dass soviel in ein System- und 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 zuletzt steigt schlussendlich die Systemqualität.

Elemente eines Testmodells

Zur Vereinfachung nehme ich im Folgenden die Existenz eines System- und Testmodells an. Wie Abbildung 4 verdeutlicht, gliedert sich ein Testmodell selber in verschiedene Artefakte, auch wenn unter modellbasiertem Testen häufig nur die reine Ableitung von Tests aus Systemmodellen und das daraus entstehende Testmodell (bzw. der Testcode) verstanden wird. Dies ist zu einschränkend, da nicht nur der Testsynthese-Prozess sondern der gesamte Testprozess modelltechnisch unterstützt werden kann.

Ein modellbasierter Testprozesses umfasst wie jeder andere Testprozess die folgenden Artefakte:

- *Testziele:* Was ist mit welchen Mitteln zu überprüfen?
- *Test-Suite:* Eine Menge von Testfällen, die die konkreten Testabläufe und Testdaten entsprechend der Testziele beschreiben.
- *Testkampagne:* Die Auswahl, Parametrisierung und der Ablauf der Testfälle.
- *Testergebnisse:* Der konkrete Ablauf der Tests zusammen mit den Testresultaten.

Es wird jedoch explizit zwischen Test-Suite (oft auch „abstrakte Test-Suite" genannt) und Testsystem (oft auch „ausführbare Test-Suite" genannt) unterschieden. Die separate Betrachtung von abstrakten Testfällen, die den Testablauf (also das Testverhalten) umfassen, und den konkreten Testfällen (die um die Testdaten ergänzten Testfälle) sind Ausprägungen der Test-Suite. Zudem stellt Abbildung 4 heraus, dass eine abstrakte Test-Suite im allgemeinen einer Umsetzung in Form einer Testrealisierung bedarf. Sollte jedoch die Test-Suite für eine manuelle Testkampagne genutzt werden, entfällt dieser Schritt. Es wäre jedoch zu überlegen, ob das Testsystem den Tester durch die Testschritte der einzelnen Testfälle führt und interaktiv die Ergebnisse abfragt.

Ob nun all diese Artefakte im selben oder in verschiedenen, miteinander verknüpften Testmodellen abgelegt sind, sei dahingestellt. Wesentlich ist hierbei nur, dass die Artefakte formalisiert erfasst sind und dass die Relationen untereinander und die zu den Artefakten aus der Systementwicklung erfasst sind.

In einem modellbasierten Testprozess zeigen die Schritte bei der Entwicklung und Nutzung der Testartefakte Besonderheiten auf, die im Folgenden erläutert werden.

Die gewählten Testziele und/oder Teststrategien werden in Relation zum Systemmodell gestellt. Beim Einsatz von U2TP kann dies unter Nutzung des Konzepts *Test Objective* erfolgen, das eine Abhängigkeit von Systemmodellelementen zu Elementen eines Testmodells in Form von Kommentaren, Teststrukturen, etc. darstellt.

Die abstrakte Test-Suite selber, also die technologieunabhängige Beschreibung der Testabläufe und -daten, kann – wie bereits erwähnt – mit proprietären oder den standardisierten Techniken U2TP oder TTCN-3 erfolgen. Eine Gegenüberstellung von U2TP und TTCN-3 zeigt Tabelle 1.

Diese Testspezifikationen erlauben eine automatisierte Generierung ausführbarer Tests. Hier hat TTCN-3 eine besondere Stärke. Durch die Definition einer offenen Testarchitektur wird die Anbindung sowohl an beliebige Schnittstellen zu testender Systeme als auch an beliebige Testplattformen ermöglicht. So umfassen die für TTCN-3 definierten Ausführungsschnittstellen *TRI (TTCN-3 Runtime Interfaces)* und *TCI (TTCN-3 Control Interfaces)* einen Systemadapter für die Interaktion mit dem zu testenden System

	U2TP	TTCN-3
Testentwurf	✓	(–)
Testspezifikation	✓	✓
Testausführung	(–)	✓
Format	Graphisch	Textuell und graphisch
Transformation	U2TP nach TTCN-3 (✓)	TTCN-3 nach U2TP ✓

Tabelle 1: U2TP vs. TTCN-3

oder einen Testmanagement-Adapter für die Einbindung in Testausführungs-Oberflächen.

Darüber hinaus sollten Details der Testkampagnen formalisiert erfasst werden und die Konfiguration des Systems, und des Testsystems, die Selektion der Tests, deren Parametrisierung usw. umfassen.

Schlussendlich sollten die *Traces* der Testausführung formalisiert erfasst werden. Das ermöglicht eine wohl definierte Ablage der Testläufe und auch eine spätere Analyse/Revision der Testergebnisse. Hier bietet sich sicher XML als Format an, wie es beispielsweise im *TTCN-3 Logging Interface* genutzt wird.

Erzeugung von Testmodellen aus Systemmodellen

Auch wenn das Testmodell nicht nur die abstrakten Test-Suiten umfasst, so nehmen diese doch eine besondere Stellung ein. Um der folgenden Betrachtung zur Ableitung von Test-Suiten aus Anforderungen bzw. aus Systemmodell-Elementen einen technologischen Rahmen zu geben, nutzen wir eine Erweiterung der MDA (vgl. [Bus06]), die zusätzlich zu den von der OMG empfohlenen Systemartefakten auch die entsprechenden Testartefakte umfasst. Dieser Ansatz erlaubt ein system- und testmodellgetriebenes Vorgehen entlang der MDA.

Bei diesem Ansatz stellen wir jeder Systemmodellstufe eine entsprechende Testmodellstufe gegenüber:

- dem Fachmodell *CIM (Computation-Independent Model)* die Fachtests *CIT (Computation Independent Tests)*,
- dem plattformunabhängigen IT-Modell mit der Systemlogik *PIM (Platform Independent Model)* die plattformunabhängigen Tests *PIT (Plattform-Independent Tests)* und
- dem plattformspezifischen IT-Modell *PSM (Platform-Specific Model)* die plattformspezifischen Tests *PST (Platform-Specific Tests)*.

Dieses Vorgehen erlaubt die integrierte System- und Testentwicklung und ermöglicht damit eine frühzeitige Fehlererkennung bzw. -vermeidung. Schon aufgrund der Tatsache, dass beim Systementwurf (CIM und PIM) und bei der Systemrealisierung (PIM und PSM) die Tester involviert sind, wird auf die Testbarkeit der Systemanforderungen und des Systems geachtet. Das impliziert, dass die Anforderungen umfassend, eindeutig und nachvollziehbar umgesetzt werden – die Testmodelle decken mögliche Schwachstellen auf. Mehrdeutigkeiten, fehlende Konzepte und Inkonsistenzen können so schon auf Modellebene korrigiert werden. Durch das Testen auf Modellebene können sowohl statische als auch dynamische Fehler im Systemmodell erkannt werden. Das Verfahren wurde erfolgreich für Fallstudien japanischer Softwarehersteller angewendet und wird für den industriellen Einsatz vorbereitet.

Betrachtet man die UML als prominenten Vertreter für die Systemmodellierung und die verschiedenen Diagrammarten 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 2 gibt einen Überblick zur vorrangigen Nutzung von UML-Diagrammen zum Testen. Die Tabelle zeigt die Schwerpunkte der Nutzung, hat aber keinen Anspruch auf Vollständigkei t– man kann wohl bei geänderter Perspektive fast jedes Kreuz in dieser Tabelle setzen.

Methoden

Die wesentlichen Methoden beim modellbasierten Testen sind in Abbildung 6 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, Parametrisierung von Testfällen, Detaillierung von Testdaten, Parametrisierung von Testdaten oder eine Umstrukturierung der Test-Suite. Dabei kann nach jedem Verfeinerungsschritt die Korrektheit der Test-Suite geprüft werden. Hier kann die „klassische" Korrektheit von

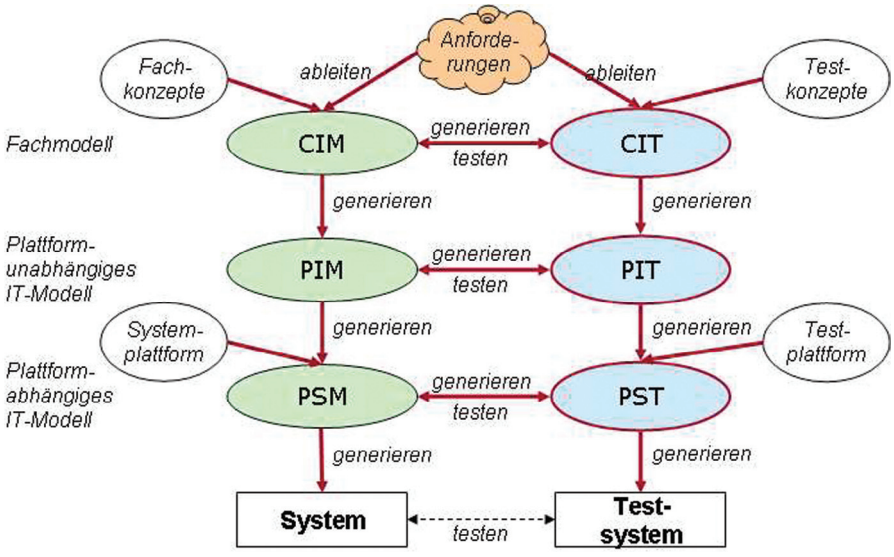

Abb. 5: MDA-Erweiterung für modellbasierte Tests

...-diagramm der UML	Teststufe				Testart		Testverfahren	
	Unit	Komponente	Inetgration	System	funktional	nicht-funktional	statisch	dynamisch
Struktur								
Klassen- *(class)*	x	x	x	x	x		x	x
Kompositions-struktur- *(composite structure)*		x	x		x		x	x
Komponenten- *(component)*		x	x	x	x		x	x
Verteilungs- *(deployment)*			x	x	x		x	
Objekt *(object)*		x	x		x		x	x
Paket- *(package)*			x		x		x	
Verhalten								
Anwendungsfall- *(use case)*			x	x	x	x[1])	x	x
Aktivitäts- *(actitiy)*			x	x	x	x[1])	x	x
Sequenz- *(sequence)*			x	x	x	x[1])	x	x
Kommunikations- *(communication)*		x	x		x	x[1])	x	x
Interaktions-übersichts- *(interaction overview)*			x	x	x	x[1])	x	x
Zeitverlaufs- *(timing)*		x	x			x[1])	x	x
Zustands- *(state machine)*	x	x			x	x[1])	x	x

Tabelle 2: Vorrangige Nutzung von UML-Diagrammen zum Testen

[1]) Oftmals unter Nutzung von UML-Erweiterungen (Profilen) für nicht-funktionale Konzepte, z. B. dem SPT-Profil (Schedulabiity, Performance and Time) oder dem QoS/FT-Profil (Quality-of Service and Fault Tolerance)

Modellen/Programmen geprüft werden, wie z. B., dass alles, was benutzt wird, auch definiert ist, dass Dinge nicht mehrfach definiert sind usw. 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 Test- und Systemmodell geprüft werden. Die Konsistenz bezieht sich auf strukturelle Aspekte, wie etwa die Übereinstimmung von System- und Testschnittstellen, das Enthaltensein der positiven Testabläufe im Systemmodell genauso wie das Fehlen der negativen Testabläufe oder die Eindeutigkeit der Tests bei der Bewertung der Systemreaktionen.

Aus einem Testmodell wird der Code für die ausführbaren Tests im allgemeinen unter Nutzung von Testplattformen (gegebenenfalls 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 Geräte 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

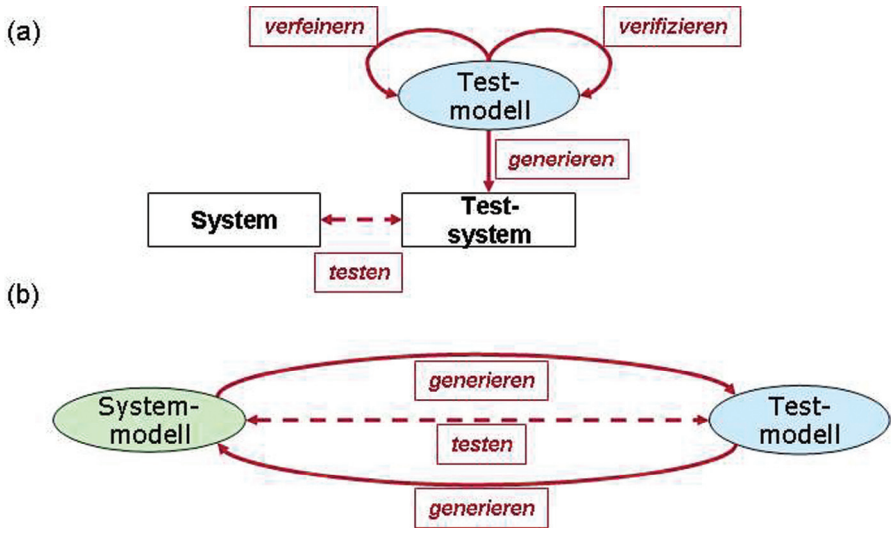

Abb. 6: Methoden modellbasierten Testens

Umsetzung von Verteilungs- und Kommunikationsprinzipien durch eine Middleware oder zur Umsetzung von Datenstrukturierungs- und Datenzugriffskonzepten 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.

Im Zentrum der aktuellen Forschung steht aber die Generierung von Test- 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. Zum Beispiel sind gängige Verfahren zum Testdatenentwurf – wie die Klassifikationsbaum-Methode, Grenzwertanalyse und Dateninvarianten – noch nicht gut mit den Verhaltensmodellen verknüpft.

Bei der Testgenerierung aus Modellen können wie bei strukturbasierten Tests Überdeckungsmaße (bei Automaten beispielsweise Zustands-, Transitions-, Pfadüberdeckung) und andere Metriken (z. B. Testschrittverzahnung, Testparametrisierung) genutzt werden, um die Güte der Tests zu bewerten und Testende-Kriterien zu definieren (vgl. z. B. [Zei07]).

Werkzeuge

Es gibt eine repräsentative, aber überschaubare Menge an Werkzeugen zum modellbasierten Testen, die erfolgreich eingesetzt werden. Zu den testmodell-getriebenen Werkzeugen zählen hierbei:

- *Conformiq Test Generator*, der eine proprietäre Erweiterung der UML um Testkonzepte nutzt (www.conformiq.com/ctg.php),
- *Eclipse TPTP*, ein Open-Source-Werkzeug, das neben *Tracing* und *Unit-Level*-Tests auch eine Realisierung von U2TP (jedoch nicht auf Grundlage der UML) bietet (www.eclipse.org/tptp/),
- *Telelogic Tau Tester*, ein TTCN-3 Werkzeug für die Entwicklung und Implementierung der Tests (über Tau G2 steht auch eine U2TP-Realisierung zur Verfügung) (www.telelogic.com/products/tau/tester) und
- *TestingTech TTworkbench*, ein in Eclipse eingebettetes TTCN-3 Werkzeug für die Entwicklung und automatisierte Ausführung der Tests (www.testingtech.de/products/ttwb_basic.php).

Zu den systemmodell-getrieben Werkzeugen gehören:

- *Conformiq Qtronic*, das über eine Anbindung an existierende UML-Werkzeuge die Testgenerierung aus den Systemmodellen unterstützt (www.conformiq.com/qtronic.php),
- *Leirios Test Generator*, das über eigene Automaten die Testgenerierung anbietet, verbunden mit Strategien zur Auswahl der Tests und Bewertung der Testgüte (www.leirios.com/contenu.php?cat=26&PHPSESSID=97c6ece42a70885af41e1033353fa030), und
- *Microsoft SpecExplore*, das aus .Net-Applikationen die Automaten erzeugt und unter Nutzung verschiedener Suchstrategien Tests generiert (research.microsoft.com/specexplorer/).

Nicht zuletzt seien hier akademische Werkzeuge genannt, die über Prototypen hinausgehen:

- *TGV*, das Testsequenzen für Ein-/Ausgabe-Transitionssysteme generiert (www.irisa.fr/pampa/VALIDATION/TGV/TGV.html),
- *TORX*, das auch Testsequenzen für Ein-/Ausgabe-Transitionssysteme generiert, aber ebenso *On-the-Fly*-Tests unterstützt (fmt.cs.utwente.nl/tools/torx/introduction.html) und
- *AutoFocus*, das basierend auf proprietären, aber zur UML ähnlichen Diagrammen für die Systemmodellierung sowohl die Verifikation als auch die Testgenerierung ermöglicht (autofocus.in.tum.de/index-e.html).

Ausblick

Dieser Artikel gibt einen klassifizierenden Überblick über existierende Ansätze und Methoden modellbasierten Testens. Auf Grund der Fülle der Arbeiten kann aber nur eine repräsentative Auswahl wiedergeben werden – eine etwas anders ausgerichtete Taxonomie findet sich in [Utt06]. Einen weiterführenden Einblick erhält man auch über Konferenzreihen wie TestCom (Testing of Communication-Based Systems), FATES (Formal Approaches to Testing of Software), MBT (Model-Based Testing) und andere.

Nicht nur die Tatsache, dass diese Konferenzreihen vorrangig wissenschaftlich ausgerichtet sind, sondern auch die überschaubare Anzahl an Werkzeugen für das modellbasierte Testen zeigen, dass modellbasierte Testverfahren noch nicht in der Breite genutzt werden. Der industrielle Durchbruch kann jedoch erwartet werden. Dazu sind über die rein technischen Verfahren hinaus die Einbindung in die Prozesse, die Ausbildung der Tester und der weitere Ausbau der Werkzeuge nötig. Dabei wird auch das in 2007 startende europäische Projekt *D-MINT (Deployment of Model-Based Technologies to Industrial Testing)* aus dem *ITEA*-Programm (*Infor-*

mation Technology for European Advancement) mit Partnern aus Deutschland, Finnland, Frankreich, Holland, Irland, Schweden und Spanien helfen.

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

Eine verstärkte 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, modell-getriebene und/oder spezifikations-getriebene Testverfahren nur eine untergeordnete Rolle, doch finden sich auszugsweise Grundprinzipien modellbasierter Testverfahren in den Lehrplänen. Ein Ausbau, z. B. mittels eines separaten Moduls zu modellbasierten Testverfahren für Testexperten, wäre wünschenswert. ■

Literatur & Links

[Abd00] A. Abdurazik, J. Offutt, Using UML Collaboration Diagrams for Static Checking and Test Generation, in: Proc. of UML Conference, Springer LNCS 1939, 2000

[Bec02] K. Beck, Test-Driven Development: By Example, Addison-Wesley, 2002

[Bri02] L. Briand, Y. Labiche, A UML-Based Approach to System Testing, in: Software and Systems Modeling (2002) 1, S. 10-42

[Bus06] M. Busch et al, Model Transformers for Test Generation from System Models, Conquest 2006, Hanser Verlag, 2006

[FOK] Fraunhofer Institut für Offene Kommunikationssysteme (FOKUS), U2TP, siehe: www.fokus.fraunhofer.de/u2tp

[Fuj91] S. Fujiwara, G. Bochmann et al, Test selection based on finite state models, in: IEEE Transactions on Software Engineering, 1991

[Gra03] J. Grabowski et al: An Introduction into the Testing and Test Control Notation (TTCN-3), in: Computer Networks Journal, Vol.42, Issue 3, 2003

[GTB] German Testing Board (GTB), Homepage, siehe: www.german-testing-board.info

[ISTQB] International Software Testing Qualifications Board (ISTQB), Homepage, siehe: www.istqb.org

[Jer99] T. Jeron, P. Morel, Test generation derived from model-checking, 11th Int. Conf. on Computer Aided Verification, 1999

[Kim99] Y.G. Kim et al, Test cases generation from UML state diagrams, in: IEE Proc. Software, Vol. 146, No. 4, 1999

[OMG-a] Object Management Group (OMG), Model Driven Architecture, siehe: www.omg.org/mda

[OMG-b] Object Management Group (OMG), Unified Modeling Language, siehe: www.omg.org/uml

[Pre05] A. Pretschner, J. Philipps, 10 Methodological Issues in Model-Based Testing, Springer LNCS 3472, 2005

[Sch05] I. Schieferdecker, The UML 2.0 Test Profile as a Basis for Integrated System and Test Development, in: Proc. of: GI Jahrestagung, Informatik 2005, Bonn, Sept. 2005

[TTCN] Testing & Test Control Notation (TTCN-3), Homepage, siehe: www.ttcn-3.org

[Utt06] M. Utting, A. Pretschner, B. Legeard, A taxonomy of model-based testing, Working Paper 4/06, The University of Waikato, 2006

[VSEK] Virtuelles Software Engineering Kompetenzzentrum (VSEK), Umfrage SUCCESS, siehe: www.software-kompetenz.de/?28616

[Zei07] 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)