Zur Kosteneffektivität des modellbasierten Testens

Alexander Pretschner

Informationssicherheit, ETH Zürich

Abstract: Annahmen zum kosteneffektiven Einsatz des modellbasierten Testens werden aufgezählt. Existierende empirische Evidenz für das Zutreffen dieser Annahmen wird genannt und die Notwendigkeit weiterer empirischer Studien betont.

1 Einleitung

Testen umfasst eine Menge von Aktivitäten, die Differenzen zwischen Ist- und Sollverhalten eines Systems aufdecken oder das Vertrauen in die Abwesenheit solcher Differenzen erhöhen sollen. Das Verständnis von Spezifikationsdokumenten, gewissermaßen ein mentales Modell des zu testenden Systems (system under test, SUT) und seiner Umwelt, erlaubt einem Testingenieur die Definition von Testfällen (im folgenden kurz „Tests“): Ein- und erwartete Ausgaben, die mit den tatsächlichen Ausgaben des SUT verglichen werden. Der Aufwand des Testens wird übereinstimmend mit 50±20% der gesamten Entwicklungskosten angegeben, wobei diese Zahlen bisweilen die Identifikation des Problems (in Code oder Spezifikation) und auch das Debugging beinhalten [14].

Kernidee des modellbasierten Testens (MBT) ist, das mentale Modell durch explizite Verhaltensmodelle von SUT bzw. dessen Umgebung zu ersetzen und so insbesondere den Prozess der Definition von Tests explizit, nachvollziehbar, reproduzierbar und letztlich unabhängig von den speziellen Fähigkeiten eines einzelnen Testingenieurs zu gestalten. Das Verhalten des Modells des SUT wird als Sollverhalten für das SUT interpretiert, geeignet auszuwählende Abläufe des Modells des SUT dementsprechend als Tests. Wenn das Modell des SUT nicht genügend Informationen über die erwarteten Ausgaben codiert und dementsprechend hauptsächlich ein Umgebungsmodell vorliegt, kann daraus nur der Eingabeteil der Testfälle abgeleitet werden, und die erwarteten Ausgaben müssen vom Tester zur Verfügung gestellt werden.

Kern dieses Aufsatzes ist die explizite Benennung von Annahmen über den kosteneffektiven Einsatz des MBT, die häufig implizit getroffen werden. Angesichts ganz erstaunlicher Fortschritte in technologischer Hinsicht wird – hier bewusst nicht abschließend – analysiert, unter welchen Bedingungen MBT kosteneffektiv eingesetzt werden kann und unter welchen Bedingungen der Einsatz fragwürdig erscheint. Das übergeordnete langfristige Ziel ist dabei die Identifikation entsprechender organisatorischer und fachlicher Erfolgsfaktoren. Zweck dieses Aufsatzes ist das Schaffen einer Diskussionsgrundlage für den Workshop.

Beitrag. Dem Verfasser sind keine Arbeiten anderer Autoren zu prinzipiellen Überlegungen zum erfolgreichen Einsatz des modellbasierten Testens bekannt. Aufbauend auf früheren Papieren [22,27], reflektiert dieser Aufsatz die aktuellen Gedanken.

Abgrenzung. In diesem Papier wird auf den Test von Funktionalität fokussiert; Usability- und Stresstests und die Eigenheiten des Testens objektorientierter Software werden nicht betrachtet. Struktur- oder Nutzungsfallmodelle zur automatisierten Generierung von Testinfrastruktur werden ebenfalls ignoriert. „Modelle“ bezeichnen Artefakte, die eine Vereinfachung des Verhaltens eines SUT oder seiner Umwelt darstellen und die (insbesondere) zum Zweck des Testens erstellt wurden [26]. Es wird angenommen, dass Modelle so präzise bzw. formal sind, dass daraus zumindest im Prinzip Tests automatisch erzeugt werden können. Von konkreten Modellierungstechniken und –sprachen wird abstrahiert, und die offenkundigen Zusammenhänge mit dem modellbasierten Test von Hardware bzw. Chipdesigns werden nicht diskutiert.

Überblick. Abschnitt 2 präsentiert zentrale Ideen des MBT. Abschnitt 3 listet fundamentale Annahmen an den erfolgreichen Einsatz des MBT auf, die als Erfolgsfaktoren gelesen werden können. Sofern verfügbar, wird über publizierte Evidenz berichtet. Abschnitt 4 zieht Schlussfolgerungen und benennt attraktive Forschungsrichtungen.

2 Modellbasiertes Testen

Testen beinhaltet drei einander ggf. überlappende Phasen: die Auswahl, Durchführung und Auswertung von Tests. Für die Auswahl von Tests werden Modelle von SUT, Modelle von dessen Umgebung und Modelle von Fehlern verwendet. Die Anzahl möglicher Tests ist üblicherweise unendlich oder zumindest sehr groß. Der ökonomische Standpunkt führt zum Wunsch nach einer möglichst kleinen Anzahl von zu erzeugenden, durchzuführenden und auszuwertenden Tests. Aus der Perspektive der Qualitätssicherung muss diese Anzahl jedoch ausreichend hoch sein, um die Anzahl verbleibender Fehler auf ein akzeptables Minimum zu reduzieren. Testfallableitung, ob automatisch oder manuell, muss also das Problem lösen, im folgenden Sinn gute Tests zu generieren: Ihre Erzeugung soll billig sein und ihre Auswertung ebenfalls, insbesondere in Bezug auf die Zuordnung von Fehlern (failures) zu Fehlerursachen (faults, errors). Außerdem sollen sie mindestens alle „schweren“ und „häufig“ auftretenden Fehler finden. Schließlich ist der Aufwand der Testdurchführung zu minimieren, was bei gemischten HW/SW-Systemen nicht immer unproblematisch ist.

Selektionskriterien. Zunächst unabhängig von der Verwendung expliziter Modelle wählen Tester als Annäherung an diese Ziele Tests anhand funktionaler, struktureller, fehlerbasierter und stochastischer Kriterien aus. Alle diese Kriterien können auch zur Bemessung einer Testsuite und als Testendekriterium verwendet werden [29]. Sie müssen vom Benutzer gefordert oder definiert werden: MBT basiert immer auf mindestens zwei vom Menschen gelieferten Eingaben, dem Modell und dem Selektionskriterium.

Strukturelle Kriterien fordern, dass während der Ausführung eines Programms oder Modells bestimmte Substrukturen von Datentypen, Kontroll- und Datenflussgraphen abgedeckt werden. Neben dem unklaren Zusammenhang von Abdeckung auf (SUT- und Umgebungs-) Modell und Code sind diese Kriterien in ihrer Fähigkeit, Fehler aufzuspüren, nicht unumstritten, auch im Vergleich mit randomisierten Tests (Referenzen in [24]). Ein großer Vorteil struktureller Kriterien liegt in ihrer Quantifizierbarkeit und darin, dass sie zumindest prinzipiell die vollautomatische Ableitung von Tests erlauben. *Funktionale Kriterien* zielen darauf ab, Funktionalitäten eines SUT zu isolieren und entsprechende Tests zu definieren. Das geschieht ad hoc, in Form von Szenarien oder Constraints über Abläufen. Wenn explizite Modelle der Umwelt des SUT oder von Anforderungen, auch in Form von Eigenschaften, vorliegen, können strukturelle Selektionskriterien auch auf diese Modelle angewendet werden. Unterstützung bei der Definition konkreter funktionaler Kriterien scheint in methodischer, aber nicht in technologischer Hinsicht möglich. *Fehlerbasierte Kriterien* verwenden empirisches Wissen um vermutete Fehler für die Selektion. *Stochastische Kriterien* schließlich basieren auf Wahrscheinlichkeitsverteilungen von Eingabedaten. Eine Gleichverteilung führt zu rein randomisierter Erzeugung des Eingabeteils von Tests, während die Verwendung von Nutzerprofilen [19] bzw. stochastischer Umweltmodelle [28] besonders attraktiv erscheint, weil sie unter bestimmten Bedingungen die Anzahl der verbleibenden, im Feld auftretenden Fehler minimiert

Testfallgenerierung. Aus einem Modell und einem adäquat operationalisierten Selektionskriterium leitet ein menschlicher oder maschineller Testfallgenerator Abläufe des Modells ab. Wenn hierfür ausschließlich ein Umgebungsmodell verwendet wurde, müssen die erwarteten Ausgaben des SUT hinzugefügt werden Projektionen von Abläufen des Modells auf Ein- und Ausgabe sind dann Tests für das SUT. Automatisierte Testfallgenerierung basiert auf dedizierten Suchalgorithmen, symbolischer Ausführung, Model Checking, deduktivem Beweisen oder Erfüllbarkeitsüberprüfern für propositionale Logik.

Abstraktion. In methodischer Hinsicht ergibt das MBT nur dann Sinn, wenn das Modell des SUT abstrakter als das SUT ist, also einen echten Informationsverlust beinhaltet [23]. Andernfalls würde der Aufwand, das Modell zu validieren, genau dem Validierungsaufwand für das SUT entsprechen (bisweilen wird argumentiert, dass der Abstraktionsunterschied nicht methodisch notwendig ist, weil allein die Redundanz von Modell und SUT zum Aufdecken von Problemen führt.) Das impliziert einerseits, dass nur das im Modell codierte Verhalten getestet werden kann und andererseits, dass die verschiedenen Abstraktionsniveaus überbrückt werden müssen (verschiedenen Abstraktionen im modellbasierten Testen sind an anderer Stelle beschrieben [21]): Der Eingabeteil des Tests wird konkretisiert, bevor er auf das SUT angewendet wird, und die Ausgabe des SUT wird abstrahiert, bevor sie mit der erwarteten Ausgabe des Tests, also des Modells, verglichen wird. Im Extremfall kann das Abstraktionsniveau so hoch sein, dass nur zwischen „Ausnahme geworfen“ und „keine Ausnahme geworfen“ oder „funktioniert“ und „funktioniert nicht“ (s. etwa eine Studie im Bereich des evolutionären Testens [7]) unterschieden wird. Konkretisierung und Abstraktion werden von dedizierten Adapterkomponenten implementiert.

Redundanz. Schließlich basiert Testen immer auf einer Form von Redundanz. Mit der Ausnahme von Stress- und Leistungstests ist es fragwürdig, ein einzelnes Modell für die Generierung von Produktionscode und Tests zu verwenden: Das SUT würde gewissermaßen gegen sich selbst getestet. Auf diese Art können allerdings Umweltannahmen und Codegeneratoren getestet werden. Verschiedene Szenarien des MBT, die die zeitliche Abfolge von Modell- und Systementwicklung berücksichtigen, werden an anderer Stelle diskutiert [23].

3 Annahmen und Evidenz

Dieser Abschnitt, der Kern des vorliegenden Papiers, fasst die Annahmen für den erfolgreichen Einsatz des MBT zusammen. Die Diskussion ist dann aufgespalten in eine Diskussion über Annahmen, für die dem Verfasser keine explizite Evidenz bekannt ist und in die Annahmen, für die Evidenz publiziert ist.

3.1 Annahmen

Häufig getroffene Annahmen über den kosteneffektiven Einsatz des MBT beziehen sich auf die im Folgenden aufgeführten Aspekte. Die Annahmen beziehen sich auf den Test von Funktionalität des SUT; modellbasierte Tests für Codegeneratoren, Compiler und Umweltannahmen werden dabei explizit aus der Diskussion ausgenommen.

1. *Modelle, Anforderungen und Spezifikationen:* Der Vorgang des Modellierens an sich hilft beim Verständnis und bei der präzisen Formulierung der Anforderungen (1a). Wenn ein Entwicklungsprozess auf sehr präzise Spezifikationen angewiesen ist, wie das etwa im Zusammenspiel von OEMs und Zulieferern im automobilen Bereich der Fall ist, dann sind Modelle als Spezifikationen ohnehin notwendig. Die Möglichkeiten des MBT führen dann zu einem zusätzlichen Wert des Modells (1b).
2. *Existenz adäquater Modelle:* Es gibt ein Modell, bei dem der Tradeoff zwischen methodisch notwendiger Abstraktion und zum Testen erforderlicher Detaillierung gerechtfertigt werden kann (offenbar bezieht sich diese Annahme nur auf die Ausprägungen des MBT, bei denen das Ausgabeverhalten des SUT hinreichend detailliert codiert wird). Weil dieses Modell eine Vereinfachung darstellt, ist es leichter zu validieren als das entsprechende SUT.
3. *Effektivität:* Modellbasiertes Testen findet (potentielle) Fehler im SUT.
4. *Relativer Aufwand und relative Qualität:* Der Aufwand, Modelle und modellbasierte Testselektionskriterien zu erstellen, zu validieren und zu warten ist kleiner als der Aufwand, eine ohne Modelle erstellte Testsuite zu erstellen, zu validieren und zu warten. Das gilt insbesondere für Modifikationen des SUT etwa bei sich ändernden Anforderungen. Wenn der Aufwand nicht geringer ist, rechtfertigt doch zumindest die höhere „Qualität“ der Tests den erhöhten Ressourceneinsatz. Bisweilen wird angenommen, dass große Testsuiten gar nicht gewartet werden können, dass das für Modelle hingegen aber sehr wohl der Fall ist.

5. *Wiederverwendung:* Im Kontext von Produktlinien oder allgemein variantenreicher Systeme können Modelle, Testselektionskriterien und Adapterkomponenten leichter wiederverwendet werden als entsprechend manuell erstellte Testsuiten.
6. *Testanzahl:* Wenn ein Modell einmal erstellt ist, können bei minimalem Aufwand beliebig viele (und im Fall reaktiver Systeme auch beliebig lange) Tests erzeugt werden. Eine größere Anzahl an Testfällen ist i.a. einer kleineren vorzuziehen.

3.2 Diskussion der Annahmen ohne dem Verfasser bekannte publizierte Evidenz

Im nächsten Abschnitt 3.3 wird empirische Evidenz für die Annahmen (1), (3) und (4) angeführt. Im verbleibenden Teil dieses Abschnitts erfolgt zunächst eine kurze Diskussion der anderen Annahmen, für die dem Verfasser keine publizierte explizite Evidenz bekannt ist.

In Bezug auf die Existenz adäquater Modelle (Annahme 2) ist festzuhalten, dass bei der Verwendung von Modellen zur automatischen Ableitung von sowohl Code als auch Testfällen stets ein fundamentaler Widerspruch zwischen den Vorteilen einer abstrakten Beschreibung und der notwendigen Detailliertheit der generierten Artefakte besteht. Abstraktion tritt in mindestens zwei Ausprägungen auf, nämlich in der Form der Kapselung und in der Form expliziten Informationsverlusts, von denen im MBT insbesondere die zweite starke Relevanz besitzt.

Abstraktion durch Kapselung wird etwa durch dedizierte Programmiersprachenkonstrukte (etwa Konditionale, Schleifen, Unterprogramme, Garbage Collectors), Middleware, Betriebssystemschichten und Bibliotheken erzielt. Die nicht explizit aufgeführte Information wird durch das Laufzeitsystem oder den Compiler eingefügt. Auch wenn die Flexibilität des Programmierers/Modellierers dadurch bewusst eingeschränkt wird, beinhaltet diese Form der Abstraktion keinen wirklichen Informationsverlust. Die Modellierung kontinuierlicher Systeme durch Matlab Simulink-Blockdiagramme etwa ist ein typisches Beispiel für Modellierung, die auf dieser Form der Abstraktion beruht.

Bei der *Abstraktion durch expliziten Informationsverlust* hingegen kann fehlende Information nicht ohne weiteres durch einen Makroexpansionsmechanismus wiederhergestellt werden. Diese Form findet etwa dann statt, wenn Teile der Funktionalität, etwa bestimmte Spezialfälle, im Modell nicht berücksichtigt werden. Das Ignorieren von Zeitverhalten oder anderen QoS-Attributen ist eine andere weit verbreitete Abstraktion dieses Typs. Bei der Erzeugung von Produktionscode oder Testfällen muss die fehlende Information offenbar explizit hinzugefügt werden.

In der Praxis des modellbasierten Testens findet man beide Formen, meist in Kombination. Details (nicht Präzision!) gehen nun überwiegend durch die zweite Form der Abstraktion verloren. Entscheidende Bedeutung kommt deshalb den die Abstraktionsunterschiede überbrückenden Adapterkomponenten zu; Schätzungen gehen von etwa 40% des Gesamtaufwands für deren Implementierung aus [27] (Teile der Adapterkomponenten – Testharnesses – sind allerdings auch bei nicht modellbasierten Testansätzen nötig). Es ist stets zu überprüfen, wie der Aufwand für diese Komponenten im Verhältnis zu den verminderten Aufwendungen bei der Validierung des Modells anstelle des Codes steht.

Zahlreiche proof-of-concept-Studien zum MBT (Annahme 3, s.u.) geben Anlass zu der Vermutung, dass adäquate Modelle stets gefunden werden können. Der zweite Teil der Annahme, der sich auf den verminderten Aufwand einfacherer Artefakte bezieht, erscheint intuitiv erfüllt.

Die mögliche Wiederverwendung (Annahme 5) von Modellen und Testselektionskriterien, insb. Umweltmodellen, basiert auf der folgenden Idee. MBT erhöht den Abstraktionsgrad der Artefakte, mit denen der Tester hantiert: Anstelle von einzelnen Testfällen und Testsuiten wird mit Verhaltensmodellen und Testselektionskriterien operiert. Aus einem Testselektionskriterium werden möglicherweise automatisch ggf. viele Testfälle generiert. Die Idee ist nun, dass Änderungen auf einem höheren Abstraktionsniveau (Modell, Selektionskriterien) leichter durchzuführen und zu validieren sind als auf einem niedrigeren Niveau (Testfälle). Argumentiert wird üblicherweise über die „Lokalität" der Änderung: Eine lokale, leicht durchzuführende Änderung auf Modellebene mit möglicherweise globalen Implikationen ist leichter zu validieren als die entsprechende globale Änderung auf Testsuiteebene. Das wird i.a. von der Art des Validierungsmechanismus abhängen.

Adapterkomponenten, die sowohl den Abgleich der Abstraktionsniveaus vornehmen als auch die Tests durchführen, sind ebenfalls Kandidaten für Wiederverwendung.

Zur Anzahl von Testfällen (Annahme 6) schließlich ist festzuhalten, dass automatische Testfallgeneratoren i.a. natürlich beliebig viele Tests generieren können. Es besteht zunächst aber nicht unbedingt ein Zusammenhang zwischen der Güte einer Testsuite, im Sinn von Anzahl detektierter Fehler, und ihrer Größe. Im Bereich eingebetteter Systeme (etwa Chipkarten, Netzwerkcontroller) kann die Durchführung eines Tests aufgrund von Rüstzeiten etwa 15 Sekunden dauern, was die Maximalanzahl durchzuführender Tests beschränkt. Für Businessinformationssysteme gilt das i.a. natürlich nicht, was die Technologie u.a. für die Erzeugung von Regressionstests z.B. nach Refactorings interessant macht. Durchgeführte Tests müssen außerdem analysiert werden, und wenn aus einer riesigen Testsuite etwa die Hälfte aller Tests auf demselben Fehler basiert, wird die Analyse schnell sehr teuer; es kann dann sinnvoll sein, alle Tests erneut durchzuführen und somit die Anzahl detektierter Differenzen zwischen Soll- und Istverhalten massiv zu reduzieren.

3.3 Diskussion der Annahmen mit publizierter Evidenz

Dass eine explizite Modellierung bei der präzisen Definition von Anforderungen und Systemen (Annahme 1a) hilfreich ist, ist weitgehend unumstritten. Existierende Evidenz wird hier bewusst nicht umfassend zitiert. Im Bereich des modellbasierten Testens sind als Evidenz insbesondere zwei Studien zu nennen [5,24]. Die erste beschreibt verschiedene Stadien des Modells eines Flugsteuerungssystems und zeigt, wie verschiedene QS-Techniken (Reviews, Model Checking, Ableitung von Modelltraces zur manuellen Überprüfung) verschiedene Fehler im Modell aufdecken.

Die zweite zeigt am Beispiel eines Netzwerkcontrollers, wie viele Fehler durch die Modellierung aufgedeckt wurden und insbesondere auch, dass bestimmte Modellierungsfehler erst durch das Ausführen modellbasiert generierter Tests offenbar wurden. In der Praxis der Ausführung modellbasiert erstellter Tests zeigt sich, dass Fehler stets sowohl in der SUT als auch im Modell gefunden werden, was die Notwendigkeit der Redundanz unterstreicht (Abschnitt 2).

Bezüglich der intuitiv einleuchtenden Annahme, dass Modelle neben ihrer Funktion als Spezifikation gewissermaßen als Nebeneffekt auch für die Generierung von Tests (Annahme 1b) verwendet können, sind dem Verfasser keine Dokumentationen bekannt. Ein Grund dafür könnte sein, dass die meisten Studien zum MBT auf Modellen basieren, die für bereits existierende SUTs erstellt wurden.

Die Effektivität des MBT (Annahme 3) in Bezug auf detektierte Fehler, zumeist ohne Berücksichtigung der Kosten und ohne Vergleich mit anderen QS-Techniken, ist eindrücklich dokumentiert. Das liegt daran, dass diverse Forschergruppen zur Bewertung der von ihnen entwickelten Technologie Fallstudien publizieren und in diesen stets Fehler finden konnten [1,3,5,9,10,11,12,15,16,18,20,24,25,17]. Diese Erkenntnisse müssen natürlich mit den Resultaten alternativer QS-Techniken in Bezug gesetzt werden – dass bei sorgfältiger Modellierung eines Systems Probleme gefunden werden, überrascht zunächst nicht (s. den folgenden Abschnitt zur Annahme 4). Dennoch zeigen die Studien klar das Potential des MBT auf.

Die für die industrielle Verbreitung von Techniken des MBT relevanteste Form der Evidenz bezieht sich auf den relativen Aufwand/Nutzen des MBT (Annahme 4): wie aufwendig ist die Erstellung modellbasierter Tests, d.h. von Modell, Selektionskriterien und Adapter, im Vergleich zu herkömmlichen Formen des Tests (zum Vergleich mit anderen QS-Maßnahmen s. andere Arbeiten [13]), und wie unterscheidet sich die „Qualität“ der Tests, z.B. in Bezug auf die Anzahl detektierter Fehler?

Untersuchungen über die Qualität von Tests finden sich in den folgenden Aufsätzen (1,2 berücksichtigen den Nutzen und ignorieren Kosten; 3,4 berücksichtigen auch Kosten).

1. Dalal et al. [11] fassen Erfahrungen aus vier industriellen Projekten zusammen. Ihr Ansatz zum modellbasierten Testen basiert ausschließlich auf Umwelt- bzw. Eingabedatenmodellen (pairwise testing), d.h. es gibt kein automatisiertes Orakel. MBT findet signifikant mehr Fehler als nicht modellbasiertes Testen.
2. Pretschner et al. [24] untersuchen MBT auf der Basis von erweiterten Zustandsmaschinen mit I/O. Die Modellierung findet eine große Anzahl von Fehlern in den vorher vorhandenen Spezifikationen, modellbasierte Tests finden signifikant mehr Fehler als nicht modellbasierte Tests, und automatisch generierte modellbasierte Tests finden nicht mehr Fehler als manuell erstellte modellbasierte Tests.
3. Bernard et al. [3] beschreiben ihren Ansatz mit in B geschriebenen Modellen. Testfälle werden anhand struktureller Kriterien erzeugt. Eine auf Anforderungen (nicht gefundenen Fehlern) basierende Subsumptionsbeziehung zwischen Testfällen wird definiert, und es wird gezeigt, dass automatisch generierte Tests i.a. eine ad-hoc

handerstellte Referenzsuite umfassen. Modellbasiertes Testen ist halb so aufwendig wie herkömmliches Testen.

4. Farchi et al. [15] beschreiben zwei Studien, in denen MBT mit in einer Variante der Murphi-Eingabesprache notierten Modellen Fehler findet, die ohne explizite Modelle nicht gefunden wurden. MBT erfordert einen um 15 Prozentpunkte niedrigeren Aufwand als nicht explizit modellbasiertes Testen.

Schließlich dokumentieren die Arbeiten von Clarke [8] und Blackburn et al. [5] bezüglich sowohl Effektivität als auch Effizienz signifikante Vorteile des MBT gegenüber nicht-modellbasiertem Testen. Da die Diskussion vermutlich aus Vertraulichkeitsgründen stark abstrahiert erfolgt, wird hier auf eine nähere Beschreibung verzichtet.

4 Schlussfolgerungen

Testen ist notwendigerweise modellbasiert [4], ob das Sollverhalten nun implizit gegeben oder in ein explizites Modell codiert ist. Viele bis heute offene Fragen, etwa die nach geeigneten Testselektionskriterien, sind demnach unabhängig davon, ob konventionelles Testen oder die explizit modellbasierte Spielart betrachtet wird. Dieses Papier hat die Erfolgsfaktoren in Form von zumeist impliziten Annahmen und existierende Evidenz benannt, die sich insbesondere auf die kostengünstige und hochqualitative Entwicklung von nicht nur zum Testen verwendeten, hinreichend abstrakten und detaillierten Modellen sowie Testselektionskriterien und Adaptern beziehen.

Die erwarteten Vorteile des MBT lassen sich dann wie folgt zusammenfassen.

1. Die Tatsache, dass das o.g. implizite mentale Modell explizit gemacht wird, erlaubt es, auf strukturierte Art und Weise ausreichend viele und gute Tests bei akzeptablen Kosten zu erzeugen.
2. MBT liefert „bessere“ und „billigere“ Tests als Strategien, die nicht auf expliziten Modellen basieren
3. Der Vorgang der Modellierung an sich sorgt dafür, dass Probleme in den Spezifikationsdokumenten frühzeitig erkannt und behoben werden können. Das MBT erzeugt somit einen zusätzlichen Wert des Modells.

Eingebettete Systeme. Die hier aufgeführten Betrachtungen sind bewusst so allgemein gehalten, dass sie sowohl für den Bereich eingebetteter Systeme als auch für den von Businessinformationssystemen zutreffen. Dennoch gibt es natürlich auch im Bereich des (modellbasierten) Testens spezielle relevante Charakteristika beider Domänen. Erstens ist der Test eingebetteter Systeme insofern erschwert, als der interne Zustand des Systems üblicherweise nicht bekannt ist, was dazu führt, dass spezielle Signalfolgen angelegt werden müssen, um diesen Zustand zu bewerten. Zweitens sind die Qualitätsanforderungen an eingebettete Systeme insofern höher, als Rückrufaktionen von Chipkarten oder automobilen Steuergeräten sehr teuer sein können. In der Konsequenz „darf“ der Testprozess ggf. etwas aufwendiger ausfallen. Drittens ist die Anzahl der Testfälle für eingebettete Systeme bisweilen durch die Hardware eingeschränkt—anstelle von Millionen von Testfällen können ggf. nur Tausende ausgeführt werden. Viertens sind eingebettete Systeme (heute noch) durch einen eher komplexen Kontrollfluss als durch komplexe

verwendete Datentypen charakterisiert. Das hat Konsequenzen insofern, als strukturelle Abdeckungskriterien für den ersten Fall besser verstanden zu sein scheinen. Schließlich sind Modelle eingebetteter Systeme und insbesondere Modelle ihrer Umwelt häufig kontinuierlicher oder hybrider Natur, was direkte Konsequenzen für die Technologie der Testfallgenerierung zeitigt.

Vielversprechende Forschungsgebiete. Erste Evidenz für den erfolgreichen und möglicherweise kosteneffektiven Einsatz des MBT ist verfügbar. Die folgenden Fragen harren allerdings noch einer soliden Beantwortung: Wieviele Fehler werden während der Modellierungsphase allein gefunden? Wie effektiv ist die Automatisierung? Wie schneidet MBT im Vergleich mit konkurrierenden Technologien ab? Wie verhalten sich Kosten und Nutzen zueinander, insbesondere unter der Annahme, dass existierende Testmanagementsysteme eingesetzt werden? Welche domänenspezifischen Charakteristika gibt es? Welche organisatorischen und prozessbezogenen Voraussetzungen sind unabdingbar? Die Technologie der Testfallerzeugung (s. die Überblicksarbeiten zu existierenden Werkzeugen [2,27]) ist zumindest für deterministische ereignisdiskrete Systeme ohne harte Echtzeitanforderungen soweit ausgereift, dass empirische Studien zur Beantwortung dieser Fragen bei allen existierenden Schwierigkeiten in Angriff genommen werden können.

Eine weitere relevante offene Frage, die aus dem Feld des MBT herausführt, liegt in der Entscheidung über adäquate Abstraktionsniveaus bei der ingenieurmäßigen Erstellung von Modellen von SUT und Umwelt, unterstützt etwa durch mehr oder weniger domänenspezifische Abstraktionsmuster. Ebenfalls aus dem Gebiet des MBT heraus führen Forschungen, wie Modelle effizient erstellt werden können. Das beinhaltet Versionierungsmechanismen für Modelle, Modell-Debugger, werkzeuggestützte Entwicklungsschritte wir Refactorings und Regressionstests. Die Verknüpfung von Verhaltensmodellen mit Testmiddleware scheint technisch ebenfalls noch nicht befriedigend gelöst. Die Forschung zu Spezifikationssprachen für Selektionskriterien (nicht für die Tests selbst, wie TTCN-3) befindet sich noch in den Anfängen. Schließlich ist bis heute unklar, auf welcher Grundlage Selektionskriterien für konkrete Anwendungen oder Fehlerklassen definiert werden sollen.

Der Verfasser ist überzeugt, dass das MBT eine vielversprechende Technologie darstellt. Die zugrundeliegenden Ideen erscheinen insbesondere dann attraktiv, wenn sie im Zusammenhang übergreifender modellbasierter Entwicklungsprozesse gesehen werden.

Danksagung. B. Köpf, J. Philipps und B. Schätz bin ich für wertvolle Anregungen und Kommentare zu einer frühen Version dieses Aufsatzes dankbar.

5 Literaturverzeichnis

1. A. Belinfante, J. Feenstra, R. d. Vries, J. Tretmans, N. Goga, L. Feijs, S. Mauw, L. Heerink : *Formal test automation: A simple experiment*. Proc. 12th Intl. workshop on Testing of Communicating Systems, S. 179-196, 1999
2. A. Belinfante, L. Frantzen, C. Schallhart: *Tools for Test Case Generation*. [6], S. 391-438

3. E. Bernard, B. Legeard, X. Luck, F. Peureux: *Generation of test sequences from formal specifications: GSM 11.11 standard case-study*, SW Practice&Experience 34 (10):915-948, 2004
4. R. Binder: *Testing Object-Oriented Systems: Models, Patterns&Tools*. Addison Wesley, 1999
5. M. Blackburn, R. Busser, A. Nauman: *Why model-based test automation is different and what you should know to get started*. Proc. Intl. Conf. on Practical SW Quality and Testing, 2004
6. M. Broy, B. Jonsson, J.-P. Katoen, M. Leucker, A. Pretschner (Hrsg.): *Model-Based Testing—a tutorial volume*. Springer LNCS 3472, 2005
7. O. Buehler, J. Wegener: *Automatic testing of an autonomous parking system using evolutionary computation*. SAE world congress, 2004.
8. J. Clarke: *Automated Test Gen. from Behavioral Models*. Proc.11th SW Quality Week, 1998
9. D. Clarke, T. Jéron, V. Rusu, E. Zinovieva: *Automated Test and Oracle Generation for Smart-Card Applications*. Proc. E-smart, S. 58-70, 2001
10. I. Craggs, M. Sardis, T. Heuillard: *AGEDIS Case Studies: Model-Based Testing in Industry*. Proc. 1st Eur. Conference on Model Driven Software Engineering,S. 129-132, 2003
11. S. Dalal, A. Jain, N. Karunanithi, J. Leaton, C. Lott, G. Patton, B. Horowitz: *Model-based testing in practice*. Proc. ICSE'99, S. 285-294, 1999
12. J. Dushina, M. Benjamin, D. Geist: *Semi-formal test generation with Genevieve*. Proc. 38th conf. on Design automation, S. 617-622, 2001
13. A. Endres, D. Rombach: *A Handbook of Software and Systems Engineering*. Add Wes, 2003
14. M. Fagan: *Reviews and Inspections*. In Software Pioneers—Contributions to Software Engineering, Springer, S. 562-273, 2002
15. E. Farchi, A. Hartman, S. Pinter: *Using a model-based test generator to test for standard conformance*. IBM Systems Journal 41 (1):89-110, 2002
16. L. Fournier, A. Koyfman, M. Levinger: *Developing an Architecture Validation Suite: Application to the PowerPC Architecture*. Proc. 36th ACM Design Automation Conference, S. 189-194, 1999
17. M. Horstmann, W. Prenninger, M. El-Ramly: *Case Studies*. [6], S.439-461
18. H. Kahlouche, C. Viho, M. Zendri: *An industrial experiment in automatic generation of executable test suites for a cache coherency protocol*. Proc. IFIP TC6 11th Intl. Workshop on Testing Communication Systems, S. 211-226, 1998
19. J. Musa: *Software Reliability Engineering*. Author House, 2. Auflage, 2004
20. J. Philipps, A. Pretschner, O. Slotosch, E. Aiglstorfer, S. Kriebel, K. Scholl: *Model-based test case generation for smart cards.* Proc. 8th Intl. Workshop on Formal Methods for Industrial Critical Systems, S. 168-192, 2003
21. W. Prenninger, A. Pretschner: *Abstractions in Model-Based Testing*. ENTCS 116:59-71, 2005
22. A. Pretschner: *Model-Based Testing in Practice*. Proc. FM'2005, S. 537-541, 2005
23. A. Pretschner, J. Philipps: *Methodological Issues in Model-Based Testing*. [6], S. 281-291
24. A. Pretschner, W. Prenninger, S. Wagner, C. Kühnel, M. Baumgartner, B. Sostawa, R. Zölch, T. Stauner: *One Evaluation of MBT and its Automation*. Proc. ICSE 2005, S.392-401
25. J. Shen, J. Abraham: *An RTL Abstraction Technique for Processor Micorarchitecture Validation and Test Generation*. J. Electronic Testing: Theory&Application 16 (1-2):67-81, 1999
26. H. Stachowiak: *Allgemeine Modelltheorie*. Springer, 1973
27. M. Utting, A. Pretschner, B. Legeard: *A Taxonomy of Model-Based Testing*. Eingereicht bei J. Information and Software Technology, 2005
28. G. Walton, J. Poore: *Generating transition probabilities to support model-based software testing*. Software: Practice and Experience 30(10):1095-1106, 2000
29. H. Zhu, P. Hall, J. May: *Software Unit Test Coverage and Adequacy*. ACM Computing Surveys 29:366-427, 1997