Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Ausarbeitung Anwendungen 2 - SoSe 2009

Daniel Lorenz

Model based Testing of Embedded Systems

Fakultät Technik und Informatik
Department Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Inhaltsverzeichnis

Kurzzusammenfassung

Diese Ausarbeitung dient der Übersicht der im akademischen sowie komerziellen Bereich tätigen Projekte zum modellbasierten Testen. Mitunter werden die Vorraussetzungen für das modellbasierte Testen erörtert und der Schwerpunkt auf eingebettete Systeme gelegt. Dabei werden Vergleiche und Untersuchungen im Umfeld der Automobilindustrie und zur eigenen Machbarkeitsstudie vorgenommen.

1 Einführung

Heutige Systeme werden an sich und in der Herstellung immer komplexer und somit kostspieliger für die Unternehmen sie herzustellen. Es werden Versuche unternommen, diese Kosten in der Entwicklung so gering wie möglich zu halten, um weiterhin marktfähig zu bleiben. Besonders in der Automobilindustrie, wo die elektronischen Bauteile in ihrer Komplexität weiter zunehmen, wird die Entwicklung immer teurer. Vom ersten Design bis zur Implementierung werden ausführbare Modelle in allen Stufen der Entwicklung in der Automobilindustrie genutzt (siehe Seite 622 Stürmer, 2007). Aber auch das Testen, zur Qualitätsicherung solcher Systeme, wird immer komplexer und bedarf eines systematischen Vergehens. Um dem entgegen zu wirken, werden Verfahren entwickelt, die verschiedenste Aspekte im Entwicklungsprozess unterstützen.

1.1 Model-driven Engineering

Model-driven Engineering (MDE), auch unter den Synonymen Model-Driven-Development (MDD) bekannt, ist das systematische Vorgehen zur Entwicklung von Systemen mittels Modellen. Die Modelle können abstrakt oder spezifisch sein. Sie dienen zur besseren Übersicht und beschreiben Teile und bestimmte Aspekte des Systems. Model-driven Engineering hat sich in den verschiedensten Domänen der Wirtschaft bereits etabliert. In der Softwareentwicklung und im Entwurf von elektronischen Systemen wird das Verfahren schon seit Jahren angewandt. Der Entwickler hat durch die Modellierung eine abstrakte Sicht auf verschiedene Eigenschaften des Systems:

- Struktur
- Verhalten
- Anforderung
- Daten

Somit bekommt der Entwickler ein besseres Verständnis für das System und kann diese Modelle als Basis für die Kommunikation mit anderen Entwicklern nutzen.

1.2 Model-based Testing

Model-based Testing (MBT) - modellbasiertes Testen - ist eine Teildisziplin des Model-driven Engineering. Dabei wird der Versuch unternommen aus Modellen Testdaten und Testszenarien zu generieren um die Qualität des Systems zu untersuchen. MBT hat folgende Vorteile für den Tester:

- strukturiertes Vorgehen beim Testen über Modelle auf einem abtrakteren Level
- bessere Übersicht des Testes anhand des Modells
- automatische Testgenerieren und Testdurchführung - somit Einsparen von Zeit
- Wiederverwendbarkeit von Modellen für bestimmte Testszenarien
- Möglichkeiten zum Testen in verschiedenen Entwicklungsstadien
- Testdokumentation
- Testen auf Modellen

2 Grundlagen

Dieses Kapitel dient dem Verständnis für die Wichtigkeit des Testens von Systemen und der Erläuterung von etablierten Begriffen für das Testen. Außerdem werden zum einen einige Modelle vorgestellt, die für das MBT genuzt werden können, und zum anderen die Vorraussetzungen solcher Modelle für die Testgenerierung und -ausführung.

2.1 Testen

Testen eines Systems zeigt die Anwesenheit von Fehlern und trägt zur Qualitätssicherung des Systems bei. Mittels testen lässt sich aber nicht ausschließen, dass trotzdem ein Fehler im System sein kann, denn vollständiges Testen ist nicht möglich. Das strukturierte Testen und eine gute Risikoabschätzung sind daher von großen Wert und helfen, zumindest die meisten Fehler zu finden.

2.1.1 Certified Tester

Der certified Tester ist ein Prüfverfahren des International Software Testing Qualification Board (ISTQB) für die Sicherstellung einer gemeinsamen und vergleichbaren Standard von Begriffen und Verständnis für das Testen.

2.1.2 Softwarequalitäts-Characteristiken

Sowie die ISTQB versucht ein gemeinsames Verständnis für das Testen als Basis zu Kommunikation zu schaffen, definiert die ISO-Norm 9126 einige Faktoren als Softwarequalität, die beim Testen zu berücksichtigen sind. Folgende Merkmale sind zu berücksichtigen:

1. Funktionalität
2. Zuverlässigkeit
3. Benutzbarkeit
4. Effizienz
5. Änderbarkeit
6. Übertragbarkeit

Bei der **Funktionalität** handelt es sich um das Ein- und Ausgabeverhalten des Systems das geprüft werden muss. Die **Zuverlässigkeit** beschreibt die Fehlertolleranz eines Systems als Reaktion eines definierten Defektes. Anhand diesem Kriteriums wird gemessen, wie stabil das System bleibt, nachdem ein Defekt auftrat.
Die **Benutzbarkeit** ist in viele weitere Kriterien untergliedert. Dieses Kriterium gibt an, wie hoch der Grad der Verwendung und Anwendung ist, also wie groß der Aufwand für den Benutzer ist, sein Ziel effektiv, effizient und zufriedenstellend zu erreichen. **Effizienz** beschreibt wieviel Zeit und Verbrauch an Betriebsmitteln für die Erfüllung der Aufgabe verwendet wurde. Das Qualitätsmarkmal **Änderbarkeit** beschreibt die Analysierbarkeit, Modifizierbarkeit und Prüfbarkeit. Das letzte Merkmal **Übertragbarkeit** umfasst die Merkmale der Anpassbarkeit, Austauschbarkeit und Konformität eines Systems.

2.1.3 Teststufen

Gemäß den Stufen der Entwicklung eines Produktes in der modellbasierten Entwicklung gibt es jeweilige Tests, die sicherstellen sollen, ob das Produkt fehlerfrei modelliert und entwickelt wird. Folgende **Teststufen** wurden identifiziert:

1. Komponententest - einzelne Komponenten und Funktionenen eines Systems werden getestet
2. Integrationstest - Komponenten und Funktionen im Verbund werden getestet
3. Systemtest - das System wird auf seine Anforderungen hin validiert und letzte verfikation der Funktionen
4. Abnahmetest - der Kunde oder Endbenutzer testet das System auf Akzeptanz

2.2 Testansätze

Es gibt verschiedene Ansätze Tests in den Entwicklungszyklus einzubauen. Die am häufigsten vorkommenden Vorgehensmodelle sind das Wasserfallmodell, das Spiralmodell und das V-Modell.

Das Wasserfallmodell ist in einzelne Phasen aufgeteilt, die nacheinander ausgeführt werden, wobei jede Phase obligatorische Vorgaben für die nächste Phase angibt. Bei einer falschen Vorgabe muss man in die jeweilige Stufe zurück, in der sie gemacht wurde. Das kann bei der Entwicklung dazu führen, dass man 2 bis 3 Phasen zurück gehen muss und somit möglicherweise viel Arbeit noch mal machen darf.
Das Spiralmodell ist eine Weiterentwicklung des Wasserfallmodells. Es ist ein streng iterativer

Prozess bei dem die Phasen mehrfach in einer Art Spirale durchgeführt werden, und somit Teile der Software fertig werden.

2.2.1 V-Modell

Eines der am häufigsten verwendeten Vorgehensmodelle ist das V-Modell. Es stellt vereinfacht die Phasen der Entwicklung auf der einen Seite und die Phasen des Testens auf der anderen Seite. Der Vorteil an diesem Vorgehen ist, dass der Entwickler die Testphasen den Entwicklungsphasen zuweisen und dadurch einen beim Testen gefundenen Fehler einer bestimmten Entwicklungsphase zuordnen und besser lokalieren kann. Dadurch ist es dem Entwickler möglich in die jeweilige Phase zurück zu gehen und zu untersuchen, welche Auswirkungen eine Änderung auf die weiteren Phasen hätte.
Es gibt viele Weiterentwicklungen des V-Modells. Ein neues V-Modell trägt den Namen V-Modell XT (Version 1.3 ist die neueste) und wird von der Bundesregierung für Informationstechnik als der Ablöser des V-Modells 97 proklamiert (siehe für Informationstechnik, 21.06.2009).

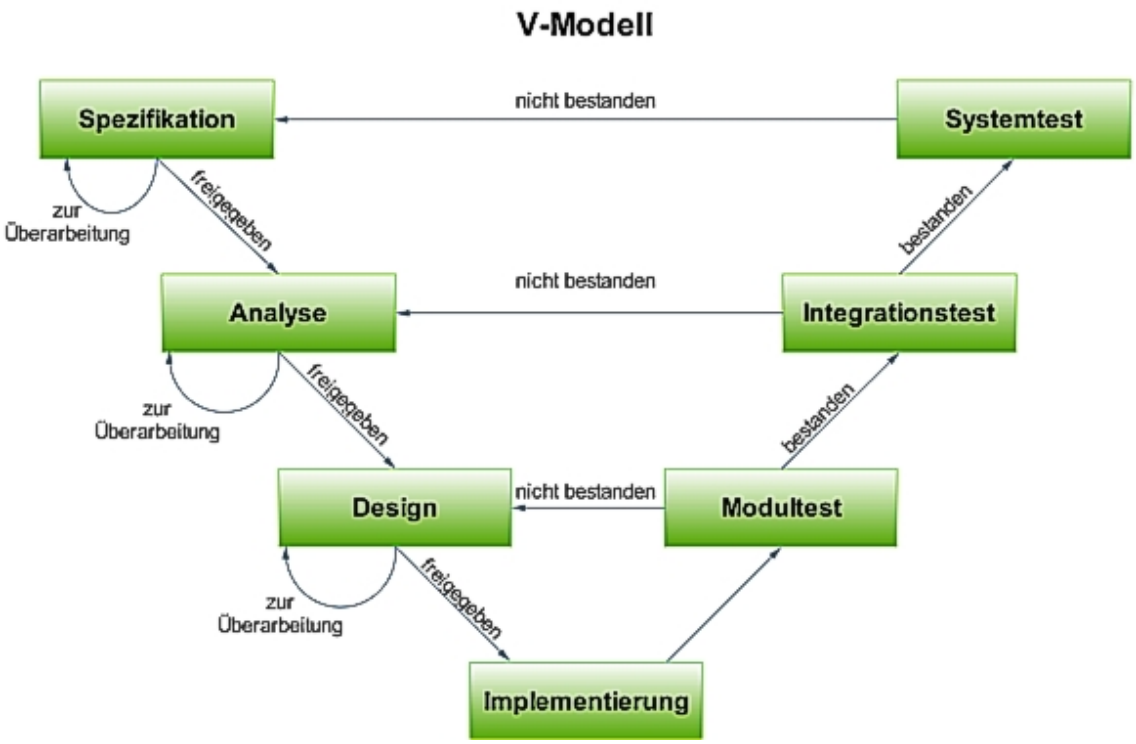

Abbildung 1: Das V-Modell

2.2.2 W-Modell

Das W-Modell ist eine Weiterentwicklung des V-Modells. Es beinhaltet die parallele Entwicklung und Durchführung von Testaktivitäten zu den Entwicklungsphasen. Dabei kann von Personen mit Fachwissen aus der Qualitätssicherung durch paralelles Testen frühzeitig Fehler oder falsche Designentscheidungen detektiert und behoben werden.

Dieses Vorgehensmodell wird in [Paul Baker (2007)] für das MBT vorgeschlagen. Parallel sollen die Modelle zum Testen entwickelt werden und die eigentlichen Modelle sowie dessen Implementierungen testen. Dieser Ansatz zum frühzeitigen erkennen von Fehlern soll Zeit sparen und Lücken in den Anforderungen für das System finden.

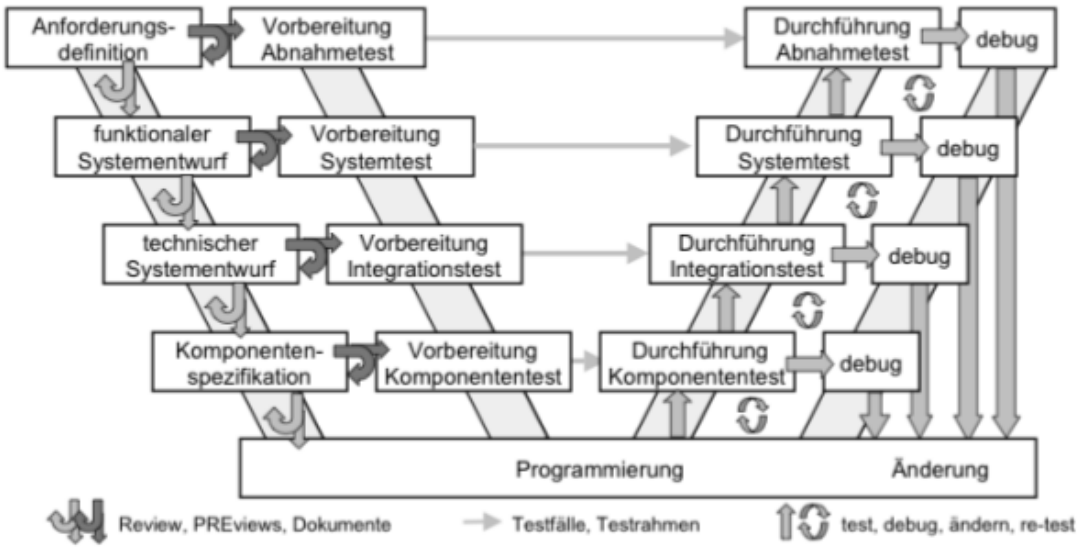

Abbildung 2: Das W-Modell

2.3 Testarten

Beim Testen muss zwischen grundsätzlich verschiedenen Testarten unterschieden werden. Sie beleuchten verschiedene Qualitätsmerkmale und benötigen verschiedene Informationen über die Anforderungen eines Systems.

2.3.1 Funktionale vs. nicht-funktionale Tests

Funktionale Tests beschreiben eine Gruppe von Tests, die das Verhalten des SUT^G untersuchen. Dabei wird das Ein- und Ausgabeverhalten des SUT gegen die funktionalen Anforderungen getestet. Die funkionalen Anforderungen geben an was das System leisten muss, um überhaupt einsetzbar zu sein. Dagegen stehen die nicht-funktionalen Tests. Sie verifizieren das SUT nach der Benutzbarkeit, Effizienz und Zuverlässigkeit. Darunter gehören Lasttests, Performanztests, Stresstests und Test der Stabilität.

2.3.2 Statische vs. dynamische Tests

Statische Tests werden parallel zur Entwicklungsphase durchgeführt. Sie beinhalten Methoden zur Ermittlung von Metriken, Datenfluss- und Kontrollflussanalyse. Durch entsprechende Werkzeuge oder von Personen wird das SUT analysiert, ohne den Code auszuführen und mit Testdaten zu betreiben. Ganz anders bei den dynamischen Tests, bei denen die Reaktion des SUT untersucht und analysiert wird. Hierbei werden mit Daten gemäß eines Aspektes (Ein/Ausgabeverhalten, Pfadüberdeckung) das System stimuliert und dabei untersucht.

2.4 Modellierungssprachen

Modellierungssprachen spielen bei MDE und bei MBT eine große Rolle. Ihre Möglichkeiten beschränken und erweitern die Möglichkeiten der Modellierung von Systemen oder Tests. Modelle sind Abbildungen der Realität und können nur einen gewissen Teil der Wirklichkeit abbilden. Sie sind beschränkt auf gewisse Aspekte und geben sie in einem Rahmen wieder. Dabei spielt der Detaillierungsgrad eine große Rolle darüber, inwieweit das Modell auf verschiedene Systeme angewendet werden kann. Folgende Modellierungssprachen sollen einen kleinen Überblick geben, wie Modellierungssprachen mit ihren Gestaltungsmöglichkeiten eingeschränkt sind oder verwendet werden können. Die hier vorgestellten Modelle bilden nur einen sehr kleinen Teil darüber ab, was es an Modellierungssprachen gibt. Es wird aber bewusst auf diese Modellierungssprachen eingegangen, weil sie im Laufe dieser Studie in einem anderen Kontext wieder verwendet werden.

2.4.1 DSL

Eine DSL^G ist ein Synonym für eine Modellierungssprache, dessen Syntax und Symantik auf eine spezielle Domäne angepasst ist. Im Automobilbereich hat sich die Modellierungssprache Simulink^G weit verbreitet und wird für die Modellierung der Elektronik verwendet. Meist handelt

es sich um verteilte eingebettete Systeme die untereinander Kommunizieren. Simulink ist geeignet für den Entwurf, der Simulation, der Implementierung und das Testen von Regelungs-, Signalverarbeitungs- und Kommunikationssystemen. Im eigentlichen Sinne ist Simulink ein kommerzielles Werkzeug basierend auf Mathlab^G mit einer eigenen DSL.

2.4.2 UML

UML (Unified Modeling Language) ist eine weit verbreitete sehr abstrakte Modellierungssprache in der Softwareentwicklung und immer mehr in der Entwicklung von Systemen aller Art. Im November 1997 standardisierte die OMG^G (siehe OMG, 21.06.2009) die UML. Die UML vereinigt eine Menge von Diagrammen (graphische Notationen) die man semantisch miteinander verknüpfen kann. Damit hat die UML (neueste Version UML 2.2) einen großen Vorteil gegebüber anderen Modellierungssprachen, weil sie dadurch viel mehr Möglichkeiten der Modellierung besitzt. Die UML ist in der Norm ISO/IEC 1950 notiert.

Profile sind eine Möglichkeit in der UML die Diagramme Syntaktisch und Semantisch zu erweitern oder zu beschränken. Mittels Stereotypen und Constraints werden solche Profile definiert. Damit ist es möglich UML Diagramme auch für domänenspezifische Modellierung zu nutzen. UTP^G ist ein Beispiel für eine Erweiterung die für das Modellieren von Testszenarien entwickelt wurde (siehe Paul Baker, 2007).

2.4.3 SysML

Die UML ist leider nicht darauf ausgelegt, Systeme im Sinne des System Engineering zu modellieren. Daher hat das INCOSE^G 2001 das Projekt SysML (System Modeling Language) ins leben gerufen. Im Jahre 2007 veröffentlichte auch die OMG die Sprache. SysML vereinigt eine Teilmenge mit der UML und erweitert sie mit eigenen Diagrammen. Die Ziele der SysML als eigene Modellierungssprache sind:

- das Definieren von Systemanforderungen
- Evaluieren von Anforderungen
- Systeminformationen kommunizieren

Welche Diagramme neu hinzugekommen und welche abgeändert wurden kann man in [Korff (2008)] und [Weilkiens (2008)] nachlesen.

2.4.4 TTCN-3

TTCN-3[G] ist eine abstrakte Modellierungssprache (Programmiersprache), die für das Definieren, das Durchführen und die Dokumentation von Testfällen entwickelt wurde. Standardisiert von der ITU[G] und der ETSI[G] wurde TTCN-3 für Kommunikationssysteme entwickelt. TTCN-3 besitzt drei Arten der Darstellung, die Core Language, das Graphical Format und das Tabular Format. Jede dieser Darstellungen lassen sich bijektiv in die andere abbilden.

3 Projekte im Bereich MBT

In diesem Kapitel werden Projekte vorgestellt, die mit MBT, der Modellierung von eingebetteten Systemen und im entferntesten Sinne mit der Automobilindustrie zu tun haben. Dies ist keine ausführliche Auflistung der Projekte in diesen Bereichen sondern ein Auswahl von verschiedensten Untersuchungen die in die Richtung von MBT von eingebetteten Systemen geht. Für eine ausführliche Auflistung siehe [Hartman (21.06.2009)] und [IX (März 2009)] für allgemeine MBT Projekte und Tools, und [Zander-Nowicka (21.06.2009)] für Projekte und Tools für MBT im Automobilbereich.

3.1 MARTE

MARTE[G] ist ein Projekt der OMG bei dem das UML Profil SPT abgelöst wird. Mit MARTE soll eine modellbasierte Beschreibung von real-time und embedded Systemen möglich sein. Dabei stehen Schedulability, Performance und Zeit im Vordergrund der Modellierung. Das MARTE Projekt ist aktuell noch am laufen (weitere Informationen Korff (2008) und MARTE (21.06.2009)).

3.2 AGEDIS

AGEDIS[G] war ein drei Jahre langes Projekt das Software Testing Tools untersuchte. Ins leben gerufen 2002 von der europäischen Union und in Zusammenarbeit mit France Telecom, Intrasoft und IBM wurden einige Erkenntnisse erschlossen.

- MBT gibt ein besseres Verständnis über das SUT, wodurch ein effiktiver Weg zur Analyse komplexer Anforderungen möglich werden
- der Aufwand bei MBT nach einer Änderung der Anforderungen war kleiner als beim herkömmlichen Testen

- die Tools für die Generierung von Tests waren zum Zeitpunkt der Untersuchung unzureichend

3.3 Early-Fault-Detection

Die Erlang Consulting Limited brachte eine Studie über das frühe detektieren von Fehlern mittels MBT (siehe Boberg, 2008). Bei der Untersuchung ging es Vordergründig um die Fragen:

- Welche Herausforderungen stellt MBT in den verschiedenen Phasen der Entwicklung?
- Welche Vorteile bringt MBT in den verschiedenen Entwicklungsphasen?

Mit Hilfe einer Methode namens Fault-Slip-Through wurde untersucht, in welcher Entwicklungsphase ein Fehler detektiert wurde, und wieviele dieser Fehler längst in einer vorherigen Phase hätten detektiert werden müssen. Diese Methode wurde auf MBT und handgeschriebene Tests angewendet. Dabei kam heraus, dass bei MBT mehr Fehler detektiert wurden als bei den handgeschriebenen Tests und, dass die Quote der Fault-Slip-Through Fehler kleiner war. Somit sprach das Ergebnis für MBT. Die Studie fand zudem aber heraus, dass bei den handgeschriebenen Tests Fehler entdeckt wurden, die mit MBT nicht entdeckt wurden, sodass handgeschriebene Tests ihre Daseinsberechtigung haben und als Ergänzung zu MBT empfohlen werden.

3.4 AUTOSAR

AUTOSAR^G ist ein offener Standard für Elektronik-Architekturen im Auto. Die Automobilbranche benutzt in der Regel zwar Simulink und Stateflow für die Modellierung, bemüht sich aber zunehmend an die UML anzupassen. Mit EAST-ADL (Electronics Architecture and Software Technologies - Architecture Despription Language) einem von der ITEA^G entwickeltes UML Profil wurde eines der ersten Ansätze in Richtung UML gemacht. Mittlerweile gibt es ein AUTOSAR Profil für die UML und es gibt zahlreiche Bemühungen in Zusammenarbeit mit IBM diese zu etablieren (siehe Hafner (21.06.2009) und Kyandoghere (21.06.2009)).

3.5 MOTION

Motion, ein Projekt des Frauenhofer Instituts in Berlin, beschäftigt sich mit Methoden und Werkzeugen für effizienten Testentwurf, Testableitung, Testspezifikation, Testimplementierung, lokale und verteilte Testausführung sowie die Analyse der Testergebnisse. Die Testmethoden und -werkzeuge sind die Grundlage für die Entwicklung von umfangreichen Testsuites für ausgewählte Zielsysteme in verschiedenen Domainen, z.B. der Telekommunikation und der

Automobilindustrie (siehe MOTION, 21.06.2009). Motion geschäftigt aber besonders mit dem ganzheitlichen Entwicklungszyklus, von der Analyse bis hin zur Implementierung. Das Projekt Motion hat die weltweit erste Entwicklungsumgebung für die Sprache TTCN-3 in der TTSuite entwickelt. Mehr dazu siehe 4.4

4 Tools im Bereich MBT

4.1 Papyrus

Papyrus ist ein open source Projekt eines UML 2.0 Design Tools. Papyrus ist ein Plugin für den Code-Editor Eclipse von IBM und beherscht folgende UML Profile:

- UML 2.0 OCL
- MARTE
- SysML
- EAST-ADL

Papyrus besitzt einen eigenen Code-Generator und eröffnet durch offene Schnittstellen die Möglichkeit des Hinzufügens von weiteren Diagrammen und Code Generatoren.

4.2 TDE/UML

TDE/UML ist ein reduziertes UML 2.0 Design Tool mit integrierten Testgenerator. Entwickelt wurde TDE/UML von Siemens Corporate Research (SCR). TDE/UML generiert Tests auf der Programmiersprache Java und testet auf den Testleveln

- Komponenten-
- Integrations-
- System-
- und Abnahmetests

4.3 MATT

MATT^G ist ein Tool für das automatische Testen von Simulink Modellen. Es beherscht Black Box Tests und gibt dem Benutzer die Möglichkeit benutzerdefinierte Tests zu entwickeln. Spezialisiert auf Datenüberdeckung erlaubt es zudem das offline Testen, also das Untersuchen der Testergebnisse nach dem Testlauf.
Für eine detaillierte Liste von MBT Tools für Matlab/Simulink Modellen siehe Zander-Nowicka (21.06.2009).

4.4 TTworkbench

TTworkbench ist ein Tool des Unternehmens Testing Technologies das eng mit dem Frauenhofer Institut zusammen arbeitet. TTworkbench wird vom Frauenhofer Institut und Testing Technologies weiter entwickelt. Das Tool definiert und generiert Tests mittels TTCN-3. Das Tool deckt sowohl funktionale als auch nicht-funktionale Tests auf allen Teststufen (Komponenten-, Integrations-, System- und Abnahmetests) ab. Siehe GmbH (21.06.2009)

4.5 IBM Rational Rhapsody

Das letzte Tool entstammt aus dem Know How von zwei Unternehmen. Rhapsody kommt ehemals von Telelogic. Telelogic wurde aufgekauft und Rhapsody wurde mit dem Wissen aus IBM Rational zusammen zu IBM Rational Rhapsody verbunden. Das Eclipse Plugin beinhaltet einen UML 2.0 Design, einen Code-Generierungs und einen Testfallgenerator Tool. Es beherscht das Modellieren mit SysML, UML 2.0 OCL und das neue AUTOSAR UML Profil.

5 Der eigene MBT Ansatz

Ich verfolge den Ansatz MBT anhand der Automobilbranche mit open source und frei verfügbaren Werkzeugen zu betreiben, ähnlich wie bei Kyandoghere (21.06.2009). Es sollen nichtfunktionale Tests mit MBT durchgeführt werden. Für die Performace- und Robustnesstests braucht man geeignete Modelle zum Spezifizieren von Testszenarien. Ich habe mir gedacht SysML als Basis vereinigt mit MARTE und AUTOSAR für die Modellierung zu verwenden. Dabei steht eine Untersuchung der Verträglichkeit einer Mischung zwischen den Profilen der UML und SysML an. Das Ziel ist es zumindestens auf der Basis der UML zu bleiben und mit open source Werkzeugen (bsp. Papyrus 4.1) MBT zu betreiben.

6 Zusammenfassung

MBT ist eine relativ neue Disziplin, die in einigen Domänen wie der Automobilindustrie stark vertreten ist und erfolgreich eingesetzt wird. Zwar erzeugt MBT weiteren Arbeitsaufwand, aber gemessen an den Aufwand für das händische Testen und der Genauigkeit von MBT ist der Aufwand um vieles kleiner. Gerade im Bereich der eingebetteten Systeme bedarf es noch mehr an systematischen Testen. “Heutzutage erfüllen 44% der eingebetteten Systemdesigns 20% der Erwartungen an Funktionalität und Leistung. Das liegt zum Teil daran, dass ein passender Testansatz für funktionale Validierung und Verifikation im Bereich der eingebetteten Systeme fehlt “(siehe Zander-Nowicka, 21.06.2009). Es ist auch ein Trend zu verzeichnen, der von den DSL weg zu den abstrakten Modellierungssprachen führt (siehe Weilkiens, 2008).

Literatur

[IX März 2009] *IX Studie: Modellbasiertes Testen*. Heise Zeitschriften Verlag GmbH Co KG, März 2009. – URL http://www.heise.de/kiosk/special/ixstudie/09/01/

[Andreas Spillner 2005] ANDREAS SPILLNER, Tilo L.: *Basiswissen Softwaretest*. dpunkt.verlag GmbH, 2005. – ISBN 3-89864-358-1

[Boberg 2008] BOBERG, Jonas: Early fault detection with model-based testing. In: *ACM* (2008)

[Conrad 2005] CONRAD, Mirko: Systematic Testing of Embedded Automotive Software - The Classification-Tree Method for Embedded Systems (CTM/ES). In: BRINKSMA, Ed (Hrsg.) ; GRIESKAMP, Wolfgang (Hrsg.) ; TRETMANS, Jan (Hrsg.): *Perspectives of Model-Based Testing*. Dagstuhl, Germany : Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany, 2005 (Dagstuhl Seminar Proceedings 04371). – URL http://drops.dagstuhl.de/opus/volltexte/2005/325. – ISSN 1862-4405

[ETSI 21.06.2009] ETSI: *Webseite der ETSI*. 21.06.2009. – URL http://www.etsi.org/WebSite/homepage.aspx

[GmbH 21.06.2009] GMBH, Testing Technologies I.: *TTworkbench*. 21.06.2009. – URL http://www.testingtech.com/products/ttworkbench.php

[Hafner 21.06.2009] HAFNER, Martina: *Modellierungssprache für die Kfz-Software Entwicklung*. 21.06.2009. – URL http://www.elektronikpraxis.vogel.de/

themen/embeddedsoftwareengineering/analyseentwurf/articles/95528/

[Hartman 21.06.2009] HARTMAN, Alan: *Webseite des Projektes AGEDIS*. 21.06.2009. – URL http://www.agedis.de/documents/ModelBasedTestGenerationTools.pdf

[IBM 21.06.2009] IBM, (Telelogic): *IBM Rational Rapsody*. 21.06.2009. – URL http://www.telelogic.com/products/rhapsody/index.cfm

[INCOSE 21.06.2009] INCOSE: *Webseite der INCOSE*. 21.06.2009. – URL http://www.incose.org/

[für Informationstechnik 21.06.2009] INFORMATIONSTECHNIK, Bundesregierung für: *Das V-Modell XT*. 21.06.2009. – URL http://www.cio.bund.de/DE/IT-Methoden/V-Modell_XT/v-modell_xt_node.html

[ITU 21.06.2009a] ITU: *Webseite der ITU*. 21.06.2009. – URL http://www.itu.int/net/home/index.aspx

[ITU 21.06.2009b] ITU, ETSI: *TTCN-3*. 21.06.2009. – URL http://www.ttcn-3.org/

[Korff 2008] KORFF, Andreas: *Modellierung von eingebetteten Systemen mit UML und SysML*. Spektrum Akademischer Verlag Heidelberg, 2008. – ISBN 978-3-8274-1690-2

[Kyandoghere 21.06.2009] KYANDOGHERE, Kyamakya: *Systems Modelling Language Automotive Open System Architecture*. 21.06.2009. – URL vi.uni-klu.ac.at/publications/studentwork/2.pdf

[LIST 21.06.2009] LIST, CEA: *Papyrus UML*. 21.06.2009. – URL http://www.papyrusuml.org/

[MARTE 21.06.2009] MARTE, OMG: *The official OMG MARTE Web site*. 21.06.2009. – URL http://www.omgmarte.org/

[MOTION 21.06.2009] MOTION: *Modellbasiertes Testen*. 21.06.2009. – URL http://www.fokus.fraunhofer.de/de/motion/ueber_motion/arbeitsthemen/modellbasiertes_testen/index.html

[OMG 21.06.2009] OMG: *Webseite der OMG*. 21.06.2009. – URL http://www.omg.org/

[Paul Baker 2007] PAUL BAKER, Jens Grabowski Oystein Haugen Ina Schieferdecker Clay W.: *Model-Driven Testing*. Springer-Verlag Berlin, Oktober 2007. – ISBN 978-3-540-72562-6

[Stürmer 2007] STÜRMER, Ingo: Systematic Testing of Model-Based Code Generators. In: *IEEE Explore* (2007), September

[Weilkiens 2008] WEILKIENS, Tim: *Systems Engineering mit SysML/UML*. dpunkt.verlag GmbH, 2008. – ISBN 978-3-89864-577-5

[Zander-Nowicka 21.06.2009] ZANDER-NOWICKA, Justyna: *Model-based Testing of Real-Time Embedded Systems in the Automotive Domain*. 21.06.2009. – URL http://www.irb.fraunhofer.de/bookshop/artikel.jsp?v=229241

Glossar

AGEDIS Automated Generation and Execution of Test Suites in Distributed Component-based Software

AUTOSAR AUTomotive Open System ARchitecture

DSL Domain specific language - eine Modellierungssprache mit einer bestimmten Syntax und Symantic für eine bestimmte Domäne (z.B. Modellica)

ETSI European Telecommunications Standards Institute ETSI (21.06.2009)

INCOSE International Council of System Engineering INCOSE (21.06.2009)

ITEA Information Technology of European Advanced

ITU International Telecommunication Union ITU (21.06.2009a)

MARTE Modeling and Analysis of Real-time and Embedded systems

Mathlab ist ein Tool zur Modellierung. Es ist gut für numerische oder technische Berechnungen, hällt eine interaktive Umgebung für Algorythmenentwicklung und die Visualisierung und Analyse von Daten bereit.

MATT Matlab Automated Testing Tool

OMG Object Management Group - ist ein internationales Konsortium welches Standards für objektorientierte Sprachen entwickelt

Simulink ein Tool mit eigener DSL aus der elektrotechnischen Branche mit hierarchischen Blockschaltdiagrammen für den Entwurf und die Simulation von zeitdiskreter und zeitkontinuierlicher Systemen

SUT System under Test - das zu testende System

TTCN-3 Testing and Test Control Notation (die 3 steht für die Verionsnummer)

UTP UML Testing Profile - ein Profil der UML zum Modellieren von Testscenarien

Index