Von Use Cases zu Test Cases: Eine systematische Vorgehensweise

Mario Friske, Holger Schlingloff
Fraunhofer FIRST
Kekuléstraße 7
D-12489 Berlin
{mario.friske,holger.schlingloff}@first.fhg.de

Kurzfassung

Anwendungsfälle (Use Cases) dienen oftmals nicht nur als Grundlage für den Systementwurf, sondern auch für System- und Abnahmetests. Die Ableitung der Testfälle geschieht jedoch oft intuitiv und unsystematisch. In dieser Arbeit beschreiben wir eine Methode zur systematischen Erzeugung von Testfällen aus Anwendungsfällen. In einem ersten Schritt wird der semantische Bezug zwischen Use Case Elementen und Systemfunktionen hergestellt. In einem zweiten Schritt werden die Use Cases aufbereitet und in Aktivitäts-Diagramme überführt, die dann mit automatischen Testgenerierungswerkzeugen weiter verarbeitet werden können. Im Gegensatz zu Ansätzen, die auf der vollautomatischen Analyse natürlicher Sprache basieren, erlaubt unsere Vorgehensweise, alle normalerweise benötigten Sprachelemente zu verwenden. Im Gegensatz zu informellen oder leitfadenbasierten Methoden kann unsere Methode gut durch automatisierte Werkzeuge unterstützt werden.

1 Einleitung

Ein wesentliches Merkmal der modellbasierten Entwicklung eingebetteter Steuergeräte ist es, dass die Qualitätssicherung parallel zum gesamten Entwurfsprozess durchgeführt wird. Bereits frühzeitig im modellbasierten Entwicklungsprozess wird aus den informell formulierten Anforderungen ein ausführbares Modell erstellt, welches dann in mehreren Schritten bis zu einem Implementierungsmodell verfeinert wird. Parallel dazu erfolgt die Erstellung ausführbarer Testfälle, mit denen das Modell in jedem Reifestadium getestet wird.

Für die Ableitung der Testfälle gibt es dabei mehrere Möglichkeiten. Zum einen kann das Modell selbst genutzt werden, um daraus Testsequenzen zu generieren (siehe z.B. [SHS03]). Aus einer Simulation des Systemmodells zusammen mit einem Modell der vor-

gesehenen Systemumgebung werden Eingangsdaten für die Sensoren generiert und die zu erwartenden Reaktionen an den Aktuatorausgängen gemessen. Die so gewonnenen Ereignisfolgen werden für den Test der nächsten Entwicklungsstufe verwendet. Sie enthalten die Stimuli für das zu testende System und dienen gleichzeitig als Testorakel für das Systemverhalten. Diese Vorgehensweise bietet sich insbesondere im letzten Entwicklungsschritt an: Aus dem Implementierungsmodell werden so Testfälle für Hardware-in-the-Loop Tests gewonnen, mit denen das korrekte Zusammenspiel der generierten Software mit dem eingebetteten Zielprozessor untersucht wird.

Eine andere Möglichkeit, Testfälle zu erhalten, besteht darin, die ursprünglichen Anforderungsbeschreibungen zu verwenden. Aus den funktionalen Benutzeranforderungen werden dabei (möglichst systematisch) unmittelbar Testfälle erzeugt, mit denen die ausführbaren Modelle getestet werden. Auf diese Weise kann bereits das allererste grobe Architekturmodell systematisch auf Übereinstimmung mit bestimmten Anforderungen überprüft werden. Ebenso wie das Systemmodell selbst unterliegen bei dieser Methode auch die Testfälle einer Entwicklung und Anpassung an die einzelnen Entwicklungsstufen. Die Vorgehensweise unterstützt vor allem System- und Abnahmetests, da die Benutzersicht auf das Gesamtsystem im Vordergrund steht.

Ein wichtiger Punkt bei dieser Vorgehensweise ist die Systematik der Erstellung von Tests aus den im Pflichtenheft beschriebenen Anforderungen. Zur Beschreibung funktionaler Anforderungen werden beim objektorientierten Softwareentwurf oftmals *Use Cases* verwendet. Für geschäftsprozessunterstützende Softwaresysteme hat sich insbesondere die in [Coc00] und [SW01] definierte Darstellungsform durchgesetzt. Ein Use Case ist dabei eine Beschreibung typischer Nutzer-System-Interaktionen in natürlicher Sprache oder tabellarischer Notation. Use Cases sind oftmals Teil des Kontraktes zwischen Auftraggeber und Auftragnehmer und bilden daher eine Grundlage für die Systementwicklung. Im Bereich eingebetteter Systeme werden Pflichtenhefte dagegen vielfach nicht direkt in Form von Use Cases formuliert, sondern durch Mischformen aus tabellarischen und informellem Text. Implizit sind jedoch auch solche Dokumente häufig durch die Anwendersicht strukturiert und können daher zur Erstellung von Use Cases genutzt werden. In [DPB03] sind Richtlinien zur Erstellung von Use Cases aus informellen Anforderungsbeschreibungen für eingebettete Systeme angegeben.

Use-Case-Beschreibungen lassen auch als Ausgangspunkt für funktionale Systemtests verwenden. Da die Formulierung von Use Cases jedoch in natürlicher Sprache erfolgt, ist es beim heutigen Stand der Technik nicht möglich, sie vollständig automatisch in Testfälle zu transformieren. Es existiert zur Zeit nicht einmal ein standardisiertes Format, in dem Use-Case-Beschreibungen notiert werden. Ein wichtiges Problem ist daher die Aufbereitung von Use-Case-Beschreibungen für den Systemtest.

Zur Lösung dieses Problems existieren mehrere Ansätze. Zum einen gibt es Versuche, die Ausdrucksmächtigkeit natürlicher Sprachen einzuschränken [Sch98]. Zum anderen können Use Cases auf relevante Formulierungen und Schlüsselwörter untersucht werden. Bei der (manuellen) Erstellung von Testfällen kann diese Information benutzt werden [McC03].

In dieser Arbeit schlagen wir eine interaktive Vorgehensweise zur Aufbereitung von Use-

Case-Beschreibungen vor, die diese beiden Ansätze vereint. Zunächst werden den einzelnen Schritten im Use Case die entsprechenden Systemfunktionen und -reaktionen zugeordnet und der Kontrollfluss formalisiert. Anschließend werden die Use Cases in eine formalere Notation überführt, von der aus sie mit automatischen Testfallgenerierungsalgorithmen weiterverarbeitet werden können.

2 Use-Case-Beschreibungen und ihre Formalisierung

Bei der Anforderungsanalyse großer Softwaresysteme werden in der Regel Teams von Experten aus den unterschiedlichsten Fachrichtungen eingesetzt, die parallel und verteilt einzelne Use Cases erstellen und zur Menge der Anforderungen hinzufügen. Daraus ergeben sich oft *inkonsistente*, *mehrdeutige* und *unvollständige* Spezifikationen, die nicht zur automatischen Testfallerzeugung genutzt werden können [Pos96].

Daher müssen textuelle Anforderungen in Use Cases in semantisch eindeutiger Weise formalisiert werden, bevor daraus automatisch Testfälle erstellt werden können. An eine Methodik zum Formalisieren von Anforderungen werden unterschiedliche Ansprüche gestellt, abhängig davon, ob das Ziel die Entwicklung oder der Test des Systems ist. Werden Anforderungen für die Systementwicklung formalisiert, dürfen dabei noch keine Entwurfsentscheidungen getroffen werden. Zwischen informellen und formalisierten Anforderungen darf keine Verfeinerungsbeziehung erzeugt werden, wie sie zwischen Anforderungen und Design besteht. Die formalisierten Anforderungen müssen noch jede mögliche Realisierung zulassen, welche die informell notierten Anforderungen des Auftragebers an das System erfüllt.

Das Ziel des Systemtests ist es zu validieren, ob eine konkrete Realisierung die gestellten Anforderungen erfüllt. Mit der zu prüfenden Implementierung muss nur eine mögliche Verfeinerung der Spezifikation betrachtet werden – alle Entwurfsentscheidungen sind schon gefallen. Dementsprechend muss eine Methodik zur Formalisierung von Anforderungen für den Systemtest obige Forderung nicht erfüllen. Bei der Formalisierung für den Test lassen sich sogar design- und implementierungsspezifische Informationen aus dem Systementwurf für die Formalisierung nutzen.

Die Kernforderungen an die Formalisierung für den Systemtest lassen sich folgendermaßen zusammenfassen:

1. Beseitigung bzw. Verringerung des Interpretationsspielraumes, sowohl für den Kontrollfluss als auch für die einzelnen Schritte
2. Herstellung des Bezuges zur Implementierung
3. Erhaltung der expliziten Repräsentation der Szenarien

Als potentielles Zielformat für die Formalisierung von Use-Case-Beschreibungen sind prinzipiell alle Formalismen zur Darstellung von Interaktionen geeignet. Typische Vertreter dieser Kategorie sind *Message Sequence Charts* (MSC) [OMG03a], UML2.0 MSC

[OMG03b], *Life Sequence Charts* LSC [DH01], Activity-Diagramme [OMG03a] und Statecharts [Har87], [OMG03a].

Darüber hinaus gibt es Beschreibungstechniken, die speziell auf die Darstellung szenarienbasierter Nutzer-System-Interaktionen ausgerichtet sind, wie Use-Case-Schrittgraphen [Win99], Templates mit strukturiertem Text [Rup02], glossarbasierte Templates [RH04] und Strukturierungsvorgaben für Activity-Diagramme [HVFR04].

Nicht alle dieser Beschreibungstechniken eignen sich als Zielformat für die Formalisierung von Use-Case-Beschreibungen für den Systemtest, sofern obige Forderungen erfüllt werden sollen. Strukturierter Text ist nicht geeignet, da er einerseits viel zu aufwendig zu erzeugen ist, andererseits aber auch nur ein Zwischenformat ist. In Statecharts sind die in Use-Case-Beschreibungen repräsentierten Szenarien zwar noch implizit enthalten aber nicht mehr explizit dargestellt. In MSC lassen sich Szenarien zwar explit repräsentieren, jedoch nur mit linearen Kontrollflüssen.

In den UML2.0 MSC ist diese Einschränkung beseitigt worden. Eine weitere geeignete Repräsentationsform sind Activity-Diagramme, insbesondere bei Verwendung von Strukturierungsvorgaben, wie sie in [HVFR04] dargestellt sind.

Ein Großteil der in der Literatur dargestellte Überführungsverfahren von Use-Case-Beschreibungen fokussiert sich auf eine Formalisierung für die Systementwicklung. Wesentliches Ziel dieser Verfahren ist es, die textuelle Ausgangsspezifikation in eine formalere Darstellung zu überführen, welche fortan ausschließlich als Basis des Entwicklungsprozesses verwendet wird. In diese Kategorie fallen die meisten Verfahren, welche auf semantischer Textanalyse basieren und strukturierten Text erzeugen. Weiterhin existieren richtlinenbasierte Verfahren zur manuellen Überführung z.B. in Statecharts [DKvK+02]. Einige Formalisierungsmethodiken sind auch speziell auf den Systemtest ausgerichtet, z.B. das in [RG99] dargestellte Statechart-basierte Verfahren.

3 Systematische Überführung von Use Cases in Test Cases

In der modellbasierten Entwicklung [OMG04] wird zwischen plattformunabhängigen Modellen (PIM) und plattformspezifischen Modellen (PSM) unterschieden, aus welchen mithilfe mehrstufiger Transformationen der Code generiert wird, siehe Abbildung 1.

Im Test lassen sich ebenfalls plattformunabhängige Testfälle (PIT) und plattformspezifische Testfälle (PST) unterscheiden, in der Literatur auch oft als *logische* und *konkrete Testfälle* bezeichnet [SL02]. Die PST können anschließend in ausführbare Testskripte transformiert werden.

Use Cases beschreiben design- und technologieunabhängig typische Nutzerinteraktionen. Bei der Erstellung des PIM werden im Vergleich zu den Use Cases bereits erste Designentscheidungen getroffen. Zum Erstellen von PIT ist Wissen über das gewählte Design notwendig. Deshalb lassen sich Testfälle nicht allein aus den Use Case ableiten, sondern Wissen über die realisierenden Systemfunktionen und verwendeten Datentypen ist erforderlich. In dem hier vorgestellten Verfahren wird interaktiv der Bezug zwischen den se-

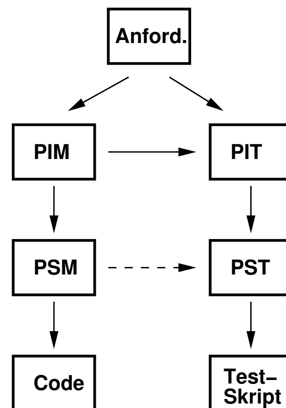

Abbildung 1: Formalisierung von Anforderungen als Transformation im Model-Driven-Testing

mantisch äquivalenten Schritten der Use-Case-Beschreibungen und den parametrisierten Systemfunktionen hergestellt. Anschließend wird diese Information genutzt, um die Use Cases in stereotypisierte Activity-Diagramme zu überführen.

Use Case als Ausgangspunkt. Das Verfahren wird am Beispiel des Use Cases *Record a Message* aus [PL99] dargestellt. Dort wird die UML-basierte Entwicklung eines digitalen Sound-Recorders beschrieben.

```
1. The user selects a message slot from the message
   directory.
2. The user presses the 'record' button.
3. If the message slot already stores a message, it is
   deleted.
4. The system starts recording the sound from the
   microphone until the user presses the 'stop' button,
   or the memory is exhausted.
```

Im Gegensatz zu Beispielen aus der Telekommunikation oder Geschäftsprozessen gibt es in eingebetteten Systemen nur ein eingeschränktes Interaktionsverhalten. Die Komplexität entsteht dabei vor allem durch die Parameter der Interaktion und nicht durch komplizierte Kontrollflüsse. Da allerdings eingebettete und kommunizierende Systeme zunehmend zusammenwachsen, sind die Grenzen fließend.

Bestimmung der Systemfunktionen und -reaktionen. Jegliche Nutzer-System-Interaktion, d.h. sowohl Systemfunktionen und -reaktionen, müssen über die Benutzerschnittstelle übertragen werden. Sofern die Benutzerschnittstelle nur durch ein *Graphical User Interface* (GUI) realisiert ist, genügt eine systematische Analyse der Oberfläche zur Bestimmung von Systemfunktionen und -reaktionen. In eingebetteten Systemen umfasst die

Systemschnittstelle zusätzlich Sensoren und Aktuatoren. Für diese gibt es häufig standardisierte Zugriffsfunktionen, die sich aus dem Systemmodell ablesen lassen.

Jedes ermittelte Element der Systemschnittstelle lässt sich mindestens einer Funktionalität zuordnen. Elemente, welche Eingaben des Systems aufnehmen (Buttons, Eingabefelder, Sensoren etc.) können den Systemfunktionen zugeordnet werden. Rein darstellende Elemente in graphischen Benutzeroberflächen (Fenster, Ausgabedialoge etc.) sowie Aktuatoren werden den Systemreaktionen zugeordnet.

Aus der Realisierung des Beispiel-Use-Case *Record a Message* lassen sich folgende Systemfunktionen ermitteln:

```
Systemfunktionen
================
select_MessageSlot(Slot)
start_Recording()
stop_Recording()
```

In einer konkreten Realisierung werden die abstrakten Systemfunktionen *start_Recording()* und *stop_Recording()* durch den Druck auf die entsprechenden Knöpfe aufgerufen. Es ist jedoch durchaus denkbar, dass auf diese Funktionen auch über andere Wege zugegriffen werden kann, z.B. über eine Fernbedienung oder ein Signal auf einem Multimedia-Bus.

Die aus der Realisierung des Beispiel-Use-Case bestimmten Systemfunktionen sind folgende:

```
Systemreaktionen
================
delete_MessageSlot(Slot)
record_Message(Slot)
```

Ermitteln des Kontrollflusses. Die Formalisierung der Use-Case-Beschreibungen lässt sich in zwei Aspekte trennen. Zum einen erfolgt die Formalisierung des Kontrollflusses, zum anderen die Verbindung der einzelnen Schritte mit den Systemfunktionen und -reaktionen.

Der Kontrollfluss wird zum Teil durch die Struktur des Templates für die Use-Case-Beschreibungen vorgegeben. Oft sind Teile des Kontrollflusses nur textuell beschrieben. Folgende Arten von Kontrollflüssen treten typischerweise in Use-Case-Beschreibungen auf: sequentielle Abfolgen, Schleifen, Fallunterscheidungen, alternative Abläufe, Sprünge sowie Includes weiterer Use Cases.

In einem Use Case Metamodell z.B. [RA98, Figure 3] werden die unterschiedlichen Kontrollflüsse als Realisierungen von *Flow of Actions* repräsentiert. Jedes dieser Konzepte repräsentiert Verbindungen zwischen Schritten, wobei die Anzahl der verbunden Schritte variiert. So werden beispielsweise in einer sequentiellen Abfolge nur jeweils zwei Schritte über eine Vorgänger-Nachfolger-Relation in Beziehung gesetzt. Eine bedingte Verzweigung hingegen verbindet drei Schritte: den Ausgangsschritt, welcher die Bedingung ent-

hält, und die beiden Schritte, mit welchen bei Erfüllung bzw. Nichterfüllung der Bedingung fortgesetzt wird.

Wie schon erwähnt, werden einige dieser Konzepte, z.B. sequentielle Abfolgen und alternative Abläufe, unmittelbar durch die templatebasierte Struktur repräsentiert und lassen sich entsprechend direkt aus der Struktur der Use-Case-Beschreibung ableiten. Andere, textuell repräsentierte Konzepte lassen sich jedoch nicht ohne weiteres ableiten, d.h. ohne Interpretation des Textes.

Ziel der Formalisierung des Kontrollflusses ist es, die einzelnen Schritte der Use-Case-Beschreibungen durch diese Konzepte zu verbinden, d.h. ein *konzeptionelles Modell* aufzubauen. Dieses kann entweder manuell erstellt werden oder werkzeuggestützt interaktiv aufgebaut werden [Mad04] [Fri04].

Schritt	Typ	Funktion oder Reaktion
1	F	select_MessageSlot(Slot)
2	F	start_Recording()
3	R	delete_MessageSlot(Slot)
4a	R	record_Message()
4b	F	stop_Recording()
4c	I	memory_exhaust

Abbildung 2: Beziehungen zwischen Schritten und Systemfunktionen und -reaktionen

Abbilden von Systemfunktionen auf Use-Case-Schritte. Jeder Use Case besteht aus Schritten, welche einen (bei sequentieller Abfolge) oder mehrerere weitere Schritte (bei Fallunterscheidungen, Schleifen, etc.) als Nachfolger haben können. Nachdem beim Ermitteln des Kontrollflusses der Zusammenhang der Schritte untereinander festgestellt wurde, wird nun der Inhalt der einzelnen Schritte betrachtet.

Im Black-Box-Systemtest wird die Eingabe-Ausgabe-Konformität zwischen Systemspezifikation und Systemrealisierung geprüft. In den Testfällen ist festzulegen, was für Eingaben in welcher Reihenfolge durch den Nutzer zu tätigen sind, einschließlich der zugehörigen Systemreaktionen.

Die Abfolge ist durch den Kontrollfluss bestimmt. Nun gilt es noch, die anderen Aspekte eindeutig zu bestimmen, d.h. festzustellen, welcher Aktor welche Systemfunktion in einem Schritt aufruft und welche Systemreaktionen dadurch hervorgerufen werden. Im Hinblick auf den Systemtest ist es ausreichend, einen Schritt entweder als eine Festlegung der auszuführenden parametrisierten Systemfunktion einschließlich dem ausführenden Aktor oder aber als parametrisierte Systemreaktion zu betrachten. Ein darüber hinausgehende Analyse ist aus Sicht des Systemtests ist nicht notwendig.

In dem als Beispiel dienenden Use Case *Record a Message* werden alle Systemfunktionen von dem gleichen Aktor *User* genutzt. In dem in Abbildung 2 dargestellten Ergebnis der Zuordnung ist deshalb der Aktor nicht mehr explizit aufgeführt. Die Spalte *Schritt* referenziert die Use-Case-Schritte. Die Spalte *Typ* bezeichnet den Typ des Schrittes, wobei

„F“ für Systemfunktion, „R“ für Systemreaktion und „I“ für einen internen Schritt steht (in diesem Fall für eine Ausnahme).

Erstellen der Zwischenrepräsentation und Testfälle. Im nächsten Schritt wird das *konzeptionelle Modell* in eine Zwischenrepräsentation übertragen. So wird beispielsweise das Konzept der sequentiellen Abfolge von Schritten in Use-Case-Beschreibungen in Activity-Diagrammen als zwei aufeinanderfolgende Activities dargestellt.

Das Ergebnis der Formalisierung des Beispiels ist in Abbildung 3 dargestellt.

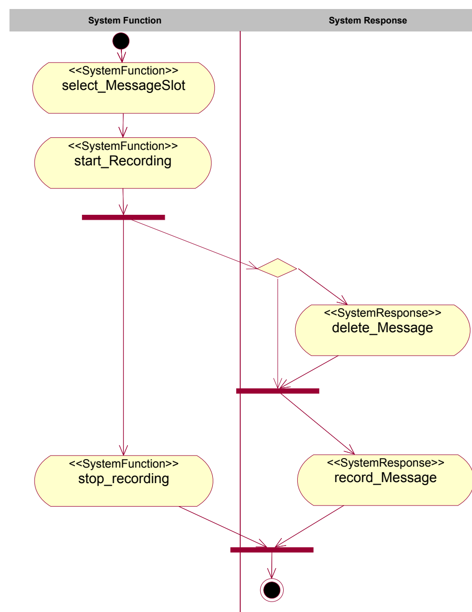

Abbildung 3: Darstellung des Use Case als UML Activity Diagramm

Aus dem so entstandenen Activity-Diagramm lassen sich Testfälle ableiten, indem gemäss festgelegten Überdeckungskriterien Pfade durch den Graphen konstruiert werden. Als Korrektheitskriterium verwenden wir dabei die *Input-Output-Conformance* [Tre96]. Ein- und Ausgaben sind durch Stereotypen gekennzeichnet. Diese können bei der Erstellung von Testfällen genutzt werden. Den abstrakten Systemfunktionen und -reaktionen sind Schnittstellen und Ereignisse zuzuordenen. Für die Systemfunktionen sind implementierungsspezifische Aufrufanweisungen zu erstellen. Für die Systemreaktionen müssen Vergleichsanweisungen zur Überprüfung der tatsächlichen mit den erwarteteten Resultaten festgelegt werden.

Wenn die Testfälle austomatisch ausgeführt werden sollen, sind die Aufruf- und Vergleichsanweisungen in ausführbare Routinen umzusetzen. Im Bereich der eingebetteten Systeme können die Aufrufanweisungen aus komplexen Bussignalen bestehen. Die Anweisungen zur Auswertung können den Vergleich kontinuierlicher Signalverläufe erfor-

derlich machen. So wird in unserem Beispiel eine Funktion zum Vergleich der vorgegebenen mit der aufgezeichneten Tonspur benötigt.

4 Weiteres Vorgehen

In diesem Positionspapier haben wir eine Methode zur systematischen Überführung von Use Cases in Test Cases für den automatisierten Systemtest skizziert. Zur Zeit wird diese Methode bei Fraunhofer FIRST prototypisch implementiert und an kommerzielle Werkzeuge angebunden. Weitere Arbeiten bestehen in der Erweiterung der Ausdrucksmächtigkeit der Anwendungsfallbeschreibungssprache, einer Parametrisierung der Methode für verschiedene formale Notationen, sowie einer durchgängigen Toolkette für die kohärente qualitätsgetriebene modellbasierte Entwicklung eingebetteter Systeme.

Literatur

[Coc00] Alistair Cockburn. *Writing Effective Use Cases*. Addison-Wesley, 2000.

[DH01] Werner Damm und David Harel. LSCs: Breathing Life into Message Sequence Charts. *Formal Methods in System Design*, 19(1):45–80, 2001.

[DKvK+02] Christian Denger, Daniel Kerkow, Antje von Knethen, Maricel Medina Mora und Barbara Paech. Richtlinien - Von Use Cases zu Statecharts in 7 Schritten. IESE-Report Nr. 086.02/D, Fraunhofer IESE, 2002.

[DPB03] Christian Denger, Barbara Paech und Sebastian Benz. Guidelines - Creating Use Cases for Embedded Systems. IESE-Report Nr. 078.03/E, Fraunhofer IESE, 2003.

[Fri04] Mario Friske. Testfallerzeugung aus Use-Case-Beschreibungen. Präsentation auf dem 21. Treffen der Fachgruppe 2.1.7 Test, Analyse und Verifikation von Software (TAV) der Gesellschaft für Informatik (GI). Softwaretechnik-Trends, Band 24, Heft 3, 2004.

[Har87] David Harel. Statecharts: A Visual Formalism for Complex Systems. *Science of Computer Programming*, 8(3):231–274, June 1987.

[HVFR04] Jean Hartmann, Marlon Vieira, Herb Foster und Axel Ruder. UML-based Test Generation and Execution. Präsentation auf der TAV21 in Berlin, 2004.

[Mad04] Mike Mader. Designing Tool Support for Use-Case-Driven Test Case Generation. Diplomarbeit, FHTW Berlin, 2004.

[McC03] James R. McCoy. Requirements use case tool (RUT). In *Companion of the 18th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, Seiten 104–105. ACM Press, 2003.

[OMG03a] OMG. UML Spezifikation Version 1.5. http://www.omg.org/uml/, 2003.

[OMG03b] OMG. UML Spezifikation Version 2.0. http://www.omg.org/uml/, 2003.

[OMG04] OMG. Model Driven Architecture (MDA). http://www.omg.com/mda/, 2004.

[PL99] Ivan Porres Paltor und Johan Lilius. Digital Sound Recorder - A Case Study on Designing Embedded Systems Using the UML Notation. TUCS Technical Report No. 234, Turku Center for Computer Science, 1999.

[Pos96] R. M. Poston. *Automating Specification-Based Software Testing*. IEEE Computer Society, Los Alamitos, 1. Auflage, 1996.

[RA98] Colette Rolland und Camille B. Achour. Guiding the Construction of Textual Use Case Specifications. *Data Knowledge Engineering*, 25(1-2):125–160, 1998.

[RG99] Johannes Ryser und Martin Glinz. A Scenario-Based Approach to Validating and Testing Software Systems Using Statecharts. In *Proc. 12th International Conference on Software and Systems Engineering and their Applications*, 1999.

[RH04] Matthias Riebisch und Michael Hübner. Refinement and Formalization of Semi-Formal Use Case Descriptions. Position Paper at the 2nd Workshop on Model-Based Development of Computer Based Systems: Appropriateness, Consistency and Integration of Models, ECBS 2004, Brno, Czech Republic, 2004.

[Rup02] Chris Rupp. *Requirements-Engineering und -Management*. Hanser, 2002.

[Sch98] Rolf Schwitter. Kontrolliertes Englisch für Anforderungsspezifikationen. Dissertation, University of Zurich, 1998.

[SHS03] Dirk Seifert, Steffen Helke und Thomas Santen. Test Case Generation for UML Statecharts. In Manfred Broy und Alexandre V. Zamulin, Hrsg., *Perspectives of System Informatics (PSI03)*, Jgg. 2890 of *Lecture Notes in Computer Science*, Seiten 462–468. Springer-Verlag, 2003.

[SL02] Andreas Spillner und Tilo Linz. *Basiswissen Softwaretest*. Dpunkt Verlag, 2002.

[SW01] Geri Schneider und Jason P. Winters. *Applying Use Cases: A Practical Guide*. Object Technology Series. Addison-Wesley, Reading/MA, 2. Auflage, 2001.

[Tre96] J. Tretmans. Test Generation with Inputs, Outputs and Repetitive Quiescence. *Software—Concepts and Tools*, 17(3):103–120, 1996.

[Win99] Mario Winter. Qualitätssicherung für objektorientierte Software - Anforderungsermittlung und Test gegen die Anforderungsspezifiktion. Dissertation, University of Hagen, 1999.