Electronic version of an article published in **Koschke, R.; Herzog, O.; Rödiger, K.-H.; Ronthaler, M. (Hrsg): Informatik 2007. Informatik trifft Logistik. Bd. 2. Beiträge der 37. Jahrestagung der Gesellschaft für Informatik e.V. (GI), 24.-27. September 2007 in Bremen. LNI-P-110, pp. 404-409**

Testfallgenerierung aus semi-formalen Use Cases

Timea Illes-Seifert, Lars Borner, Barbara Paech

Lehrstuhl für Software Systeme
Universität Heidelberg, Institut für Informatik
Im Neuenheimer Feld 326
69120 Heidelberg
illes@informatik.uni-heidelberg.de
borner@informatik.uni-heidelberg.de
paech@informatik.uni-heidelberg.de

Abstract: Zahlreiche wissenschaftliche Arbeiten thematisieren die Ableitung von Testfällen aus Anforderungen, insbesondere aus Use Cases. Die entwickelten Methoden werden jedoch nur selten in Anforderungsmanagement- oder Testwerkzeugen integriert. In dieser Arbeit stellen wir einen leichtgewichtigen semi-formalen Ansatz vor, der die automatische Ableitung von Testfällen aus Use Cases ermöglicht und leicht in bestehende Anforderungsmanagementwerkzeuge integriert werden kann. Hierbei werden logische Testfälle aus Use Cases unter Berücksichtigung des Kontrollflusses und der vom Use Case verarbeiteten Daten und ihrer Bedingungen abgeleitet.

1 Einleitung

Use Cases (UC) haben sich als Mittel zur Anforderungsermittlung und –spezifikation im objektorientierten Umfeld bewährt [Co00]. Zahlreiche wissenschaftliche Ansätze zur Ableitung von Testfällen aus UCs wurden bereits vorgeschlagen, ein Überblick von typischen Ansätzen findet sich in [IP06a]. Es existieren zahlreiche Requirements Engineering- [Ho05] und Testwerkzeuge [Il06] mit ihren jeweiligen Stärken im „eigenen“ Anwendungsbereich. Dennoch gibt es wenige Werkzeuge, die einen solchen Übergang unterstützen. Die Verknüpfung zwischen Requirements Engineering- (RE) und Testwerkzeugen beschränkt sich meistens auf die Nachvollziehbarkeit zwischen Anforderungen und Testfällen.

RE Werkzeuge sind für die Unterstützung der Arbeit der Anforderungsingenieure konzipiert. Sie erleichtern die Spezifikation der Anforderungen in Form von UCs und dienen somit vor allem als Kommunikationsmedium zwischen Anforderungsingenieuren und Kunden. Aus diesem Grund werden häufig Textfelder zur Spezifikation der UC-Inhalte verwendet[1]. Hieraus resultieren Probleme bei der automatisierten Ableitung von Testfällen aus UCs. Ein Hauptproblem stellt die *geringe Strukturierung* der UCs in Werkzeugen dar. Komplexe Alternativen sowie der Kontrollfluss innerhalb eines UCs

[1] Ein Beispiel für die zu beschreibenden Inhalte eines UCs kann in [BD03] gefunden werden.

können nur sehr schwer innerhalb der Textfelder spezifiziert werden. Die so erstellten UCs sind sehr fehleranfällig und schwer wartbar. Ein weiteres Problem resultiert aus der Tatsache, dass Tester als „Stakeholder“ der UC-Spezifikation nicht berücksichtigt werden. Somit fehlen in den UCs spezielle *Informationen*, die für den Test notwendig sind. Insbesondere können *Schritte und Daten häufig nicht getrennt* spezifiziert werden. Aufgrund der aufgezählten Probleme, ist die automatische Ableitung von Testfällen aus UC-Spezifikationen nicht möglich. Testfälle müssen folglich manuell spezifiziert werden, wodurch eine hohe *Redundanz* zwischen den UC Beschreibungen und Testfallspezifikationen entsteht, vor allem was die auszuführenden Schritte betrifft.

In dieser Arbeit stellen wir einen leichtgewichtigen Ansatz zur automatischen Ableitung von logischen Testfällen aus semi-formalen UCs vor. In einem ersten Schritt (Kapitel 2) identifizieren wir Anforderungen an ein UC-Modell, das die Testfallgenerierung aus nicht formalen UCs ermöglicht und stellen ein solches UC-Modell vor. Anschließend skizzieren wir unseren Ansatz zur Testfallgenerierung (Kapitel 3). Abschließend stellen wir verwandte Arbeiten vor (Kapitel 4), fassen die Ergebnisse zusammen und geben einen Ausblick auf weitere Arbeiten (Kapitel 5).

2 Das UC Modell

Basierend auf den oben beschriebenen Problemen ergeben sich die nachfolgenden Anforderungen an ein gemeinsames UC-Modell für den Anforderungsingenieur und den Tester, welches die automatische Ableitung von Testfällen erlaubt. Es sollte einerseits *leichtgewichtig* sein und die *natürlichsprachliche* Spezifikation beibehalten (**A1**), um als Kommunikationsmedium zwischen Anforderungsingenieuren und Kunden verwendet werden zu können. Andererseits sollte es für Testzwecke geeignet sein. Dies schließt eine *bessere Strukturierung* der UC-Spezifikation (**A2**), insbesondere die *Trennung von Daten und Schritten* (**A2.1**), eine *explizite Darstellung von Alternativen* (**A2.2**) sowie deren *Bedingungen* (**A2.3**) ein.

Das von uns entwickelte UC Modell basiert auf dem in [BD03] vorgestellten UC Modell, bestehend aus Vorbedingungen, Aktorschritten, Systemschritten, Nachbedingungen sowie Ausnahmen, und erfüllt die oben identifizierten Anforderungen. Das entstandene Modell ist in Abbildung 1 dargestellt. Ein UC besteht aus einer *Vorbedingung*, gefolgt von einem Aktorschritt. Dem Aktorschritt können wiederum weitere Aktor- und/oder Systemschritte folgen. Innerhalb eines *Aktorschritts* können mit Hilfe von *Aktorauswahlen* die verschiedenen Handlungsalternativen, die dem Aktor in diesem Schritt zur Verfügung stehen, modelliert werden. Ihnen kann ein weiterer Aktor- oder ein Systemschritt folgen ***(A2.2).*** Ein *Systemschritt* beschreibt, wie das System auf die Aktion des Aktors reagiert. Dieses Verhalten wird durch *Bedingungen*, enthalten in einer Systemauswahl, beeinflusst ***(A.2.3)***. Ein Systemschritt kann, ähnlich einem Aktorschritt, eine Menge von Systemauswahlen enthalten ***(A2.2)***. Innerhalb einer Systemauswahl können Bedingungen auf Daten durch boolesche Operatoren miteinander verknüpft werden ***(A.2.3)***. Nur wenn das Ergebnis dieser Operation „TRUE“ ergibt, wird der dazugehörige „Nachfolgeschritt“ ausgeführt (entweder ein weiterer System- oder ein Aktorschritt) oder das Ende des UCs (durch ein „reguläres“ Ende oder durch das

Auftreten eines Ausnahmenfalls) erreicht. Das Erreichen eines „regulären“ Endes bedeutet, dass die im UC definierte *Nachbedingung* erfüllt ist. Beim Auftreten einer *Ausnahme*, wird der UC beendet, ohne dass die Nachbedingung erfüllt wird. Innerhalb einer Aktorauswahl kann der Aktor *Daten* eingeben. Je nachdem, welche Bedingung in den Systemauswahlen für die eingegebenen Daten zutrifft, ergibt sich der Nachfolgeschritt ***(A.2.1)***. Somit ist die Beschreibung komplexer Kontrollflüsse möglich. Alle Inhalte der UC Modellelemente werden weiterhin natürlichsprachig verfasst ***(A1)***. Ein ausführliches UC - Beispiel zur Illustrierung der beschriebenen Modellierungskonzepte findet sich in [IBP07].

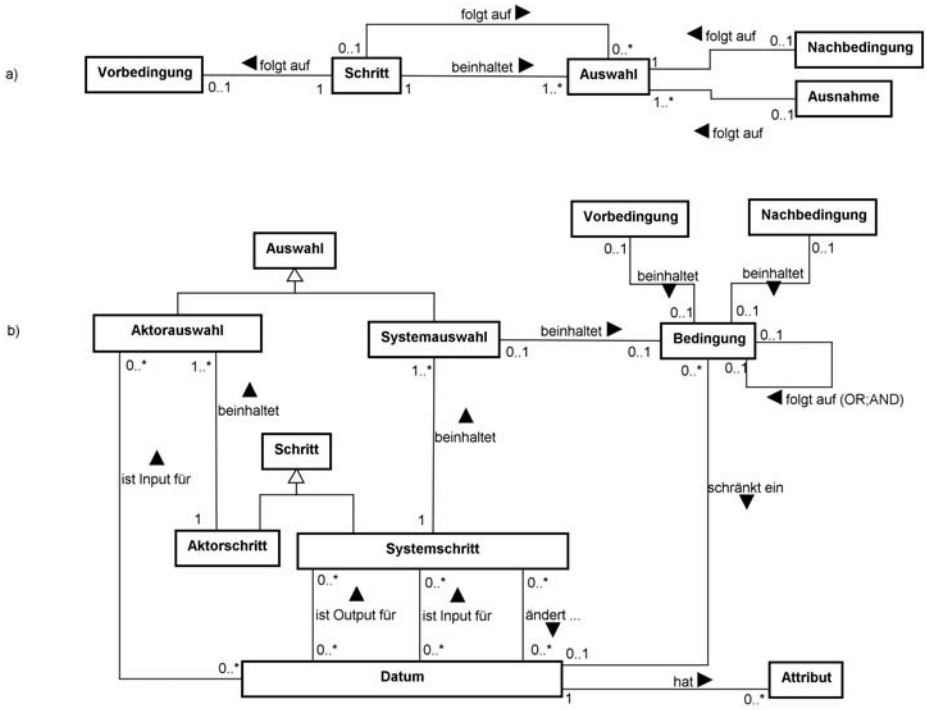

Abbildung 1: Metamodell der UC Spezifikation in Sysiphus

3 Testfallableitung

Das UC-Modell unterstützt die Ableitung von Testfällen nach verschiedenen Testtechniken: kontrollflussbasierte und kombinatorische Techniken.

Kontrollflussbasierte Technik. Um Testfälle auf der Basis der kontrollflussbasierten Testtechnik zu generieren, wird das UC Modell in einen Kontrollflussgraphen transformiert, wobei Schritte (Aktor- und Systemschritte), die Vor-, Nachbedingung und die Ausnahmen auf Knoten abgebildet werden. Auswahlen (Aktor- und Systemauswahlen) werden durch Kanten im Kontrollflussgraphen repräsentiert. Ein entsprechender Algorithmus, der die automatische Transformation des UC Modells in einen Kontrollflussgraphen durchführt ist in [IBP07] zu finden. Die Ableitung der Testfälle wird auf Basis des so konstruierten Kontrollflussgraphen durchgeführt. Ein Testfall stellt hierbei einen möglichen Pfad[2] durch den Kontrollflussgraphen dar. Folgende Überdeckungskriterien[3] bezogen auf den Kontrollflussgraphen können definiert werden: *Aktorschrittüberdeckung* (jeder Aktorschritt[4] muss in mindestens einem Testfall der Testsuite[5] enthalten sein), *Systemschrittüberdeckung* (jeder Systemschritt[4] muss in mindestens einem Testfall der Testsuite enthalten sein), *Ausnahmeüberdeckung* (jeder Ausnahmeknoten[4] muss in mindestens einem Testfall der Testsuite enthalten sein), *Endknotenüberdeckung* (jeder Endknoten – inklusive Ausnahmeknoten – muss in mindestens einem Testfall der Testsuite enthalten sein), *Schrittüberdeckung* (jeder Schritt[4] muss in mindestens einem Testfall der Testsuite enthalten sein), *Kantenüberdeckung* (in Anlehnung an die Zweigüberdeckung [SL06] - jede Kante muss in mindestens einem Testfall der Testsuite enthalten sein). Abbildung 2 stellt drei Überdeckungskriterien im Vergleich dar. Eine vergleichende Darstellung aller Überdeckungskriterien sowie ein Algorithmus zur automatischen Ableitung von Testfällen aus dem Kontrollflussgraph finden sich in [IBP07].

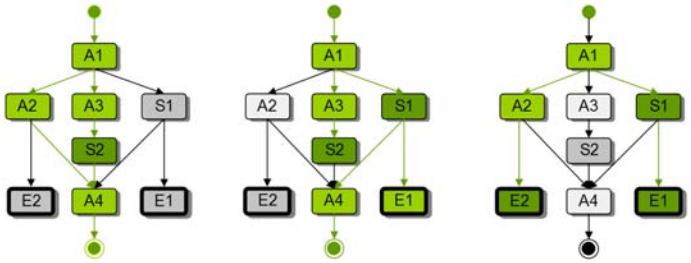

Abbildung 2: Unterschiedliche Überdeckungskriterien im Kontrollflussgraph (von links nach rechts: Aktor-, Systemschrittüberdeckung, Ausnahmeüberdeckung)

Kombinatorische Technik. Die Ableitung von Testfällen nach kombinatorischen Testtechniken erfolgt in drei Schritten. Im ersten Schritt werden alle Daten (enthalten in

---

[2] Ein *Pfad* ist eine Sequenz von Knoten im Kontrollflussgraphen, vom Startknoten (Vorbedingung) zu einem Endknoten (Nachbedingung oder Ausnahme).
[3] Ein Überdeckungskriterium gibt an, welche und wie viele Testfälle aus zu erstellen sind. Es bezieht sich nicht auf die durch Ausführung der Testfälle erreichte Überdeckung von Codezeilen.
[4] der durch einen entsprechenden Knoten im Kontrollflussgraphen repräsentiert wird
[5] Menge von Testfällen, die ein gegebenes Überdeckungskriterium erfüllt.

den Aktor- und Systemauswahlen) identifiziert. Im zweiten Schritt werden die Einschränkungen (enthalten in den Systemauswahlen) auf die Daten identifiziert. Abschließend können Datenkombinationen durch Vorgabe von Überdeckungskriterien (z.B. *each used, pair wise* oder *t-wise* [GOA05]) erstellt werden. Eine derart abgeleitete Kombination repräsentiert einen Testfall.

4 Verwandte Arbeiten

Zwei Gruppen von Ansätzen zur Ableitung von Testfällen aus UCs können unterschieden werden. *Modellexplorationsansätze* belassen UCs in ihrer ursprünglichen Form und schlagen intuitive Vorgehensweisen zur Definition von Testfällen aus UCs vor. Ein Beispiel ist in [Ah02] beschrieben. Diese Ansätze bieten den Vorteil, dass nur ein Modell sowohl zur Anforderungsspezifikation als auch zur Ableitung von Testfällen erstellt und gewartet werden muss. Im Vergleich zu unserer Vorgehensweise bieten die vorgeschlagenen Ansätze aber keine für eine automatische Generierung von Testfällen ausreichende Strukturierung von UCs an. *Modelltransformationsansätze* schlagen eine Vorgehensweise vor, die aus einer textuellen UC Beschreibung ein „Zwischenmodell“ ableitet, welches als Ausgangspunkt für die Generierung von Testfällen dient. Hierfür werden die Modelle nach bestimmten Überdeckungskriterien traversiert, um Testfälle abzuleiten. Als Zwischenmodelle dienen beispielsweise Zustandsautomaten [RQ03] oder Aktivitätsdiagramme [RQ03, BL02]. Andere Arbeiten schlagen eigene Modelle vor, wie Schrittgraphen [Wi99], Abhängigkeitsgraphen [RG03], mit OCL annotierte Zustandsdiagramme [Ne06] oder Zielgraphen [ARS05]. Der Vorteil der Modelltransformationsansätze ist die leichte Generierbarkeit von Testfällen. Im Vergleich zu unserem Ansatz, werden hierfür zwei unterschiedliche Modelle notwendig, die separat gewartet und weiter entwickelt werden müssen. In [IP06b] werden typische Ansätze in Form von Mustern zusammengefasst. Weitere Ansätze untersuchen den Nutzen einer frühen Ableitung von Testfällen aus UCs während der Anforderungserhebungsphase, beispielsweise in [RR98].

5 Zusammenfassung und Ausblick

In dieser Arbeit haben wir einen Ansatz zur semi-formalen Beschreibung von UCs vorgestellt, der eine automatische Ableitung von logischen Testfällen unter Berücksichtigung gegebener Überdeckungskriterien ermöglicht. Diese Testfälle können in einem anschließenden Schritt manuell mit konkreten Daten angereichert werden, um ausführbare Testfälle zu erstellen. Unser Ansatz zeigt, dass durch eine geeignete Strukturierung der UCs eine Automatisierung der Testfallableitung erreicht werden kann, ohne auf formale Sprachen zurückgreifen zu müssen. Weiterhin werden alle in Abschnitt 2 beschriebenen Anforderungen erfüllt, wobei zusätzlich das Aufdecken von Fehlern im Kontrollfluss der UCs (z.B. Sackgassen, Endlosschleifen) sowie die Unterstützung von verschiedenen Testtechniken mit unterschiedlichen Überdeckungskriterien ermöglicht werden. Der leichtgewichtige Ansatz ermöglicht keine automatische Generierung von konkreten Testdaten oder von ausführbarem Testcode, bietet aber dennoch den großen Vorteil, die Zusammenarbeit zwischen

Anforderungsingenieuren und Testdesignern zu verbessern, da beide Rollen auf dem selben Modell arbeiten können. Der in dieser Arbeit skizzierte Ansatz wird aktuell in das bestehende Software Engineering Werkzeug Sysiphus [Sys07] integriert und evaluiert. Weiterhin arbeiten wir an der Integration der kontrollflussbasierten und kombinatorischen Testtechniken, um durch die Informationen, die im Kontrollfluss enthalten sind, unmögliche Testfälle (die durch die Anwendung kombinatorischer Testtechniken entstanden sind) auszuschließen und somit die Testsuite zu minimieren.

Literaturverzeichnis

[Ah02] Ahlowalia, N.: Testing from Use Cases Using Path Analysis Technique, International Conference On Software Testing Analysis & Review, 2002.

[ARS05] Alspaugh, T.A., Richardson, D.J., and Standish, T.A.: Scenarios, State Machines and Purpose Driven Testing, 4th International Workshop on Scenarios and State Machines: Models, Algorithms and Tools (SCESM'05), St. Louis, USA, (2005)

[BL02] Briand, L., Labiche, Y.: A UML-based Approach to System Testing, Technical Report, Carleton University, 2002.

[BD03] Bruegge, B.; Dutoit, A. H.: Object-Oriented Software Engineering Using UML, Patterns, and Java. Prentice Hall, Englewood Cliffs, NJ, 2003.

[Co00] Cockburn, A.: Writing Effective Use Cases, Addison-Wesley Professional, 2000

[GOA05] Grindal, M.; Offutt, J.; Andler, S.: Combination Testing Strategies: a Survey. Software Testing, Verification and Reliability, 2005; S. 167-199.

[Ho05] Hood, C. et al.: iX-Studie 01/2005 Requirements Engineering: Methoden und Techniken, Einführungsszenarien und Werkzeuge im Vergleich, Heise Zeitschriften Verlag, 2006.

[Il06] Illes, T. et al.: Software-Testmanagement Planung, Design, Durchführung und Auswertung von Tests - Methodenbericht und Analyse unterstützender Werkzeuge, Heise Zeitschriften Verlag, 2006.

[IBP07] Illes-Seifert, T., Borner, L., Paech, B.: Generating Test Cases from semi-formal use case description, Technical Report, SWEHD-TR-2007-02, 2007. http://www-swe.informatik.uni-heidelberg.de/research/publications/SWEHD_TR2007_02.pdf

[IP06a] Illes, T., Paech, B.: An Analysis of Use Case Based Testing Approaches Based on a Defect Taxonomy, In (Sacha, K.): Proceedings of the IFIP International Federation for Information Processing, Band 227, Software Engineering Techniques: Design for Quality, S. 211-222, 2006.

[IP06b] Illes, T., Paech, B.: Workshop: From "V" to "U" or: How Can We Bridge the V-Gap Between Requirements and Test?, Software & Systems Quality Conferences 2006, Düsseldorf, 2006.

[Ne06] Nebut, C. et al.: Automatic test generation: a use case driven approach, Transactions on Software Engineering, Band. 32, Ausgabe 3, 2006, S. 140-155.

[RR98] Regnell, B.; Runeson, P.: Combining Scenario-based Requirements with Static Verification and Dynamic Testing, Proceedings of 4th Intl Workshop on Requirements Engineering - Foundation for Software Quality (REFSQ'98), Pisa, Italy, Juni 1998.

[RQ03] Rupp, C., Queins, S.: Vom Use-Case zum Test-Case, OBJEKTspektrum, vol. 4, 2003.

[RG03] Ryser, J., Glinz, M.: SCENT: A Method Employing Scenarios to Systematically Derive Test Cases for System Test, Technical Report, University of Zürich, 2000/03.

[SL06] Spillner, S.; Linz, T.: Basiswissen Softwaretest, dpunkt.verlag, 2006.

[Sys07] Sysiphus, http://sysiphus.informatik.tu-muenchen.de/, 18.06.2007.

[Wi99] Winter, M.: Qualitätssicherung für objektorientierte Software - Anforderungsermittlung und Test gegen die Anforderungsspezifiktion, Dissertation, Dept. of. CS, University of Hagen, Sept. 1999.