Formalisierung der funktionalen Anforderungen mit visuellen Kontrakten und deren Einsatz für modellbasiertes Testen

Gregor Engels, Baris Güldali, Stefan Sauer
Software Quality Lab, Universität Paderborn
{engels,bguldali,sauer}@s-lab.upb.de

1 Testbare Anforderungen

Die Hauptaufgabe der Anforderungsanalyse ist es, Kundenwünsche zu verstehen und sie in einer Form aufzunehmen, so dass weitere Beteiligte im Software-Entwicklungsprozess diese für die anschließenden Entwicklungsaktivitäten nutzen können. Dafür müssen die funktionalen und nichtfunktionalen Anforderungen klar und verständlich formuliert sein. Die Formulierung muss außerdem eindeutig und einheitlich sein, damit die Anforderungen des Kunden von Entwicklern richtig verstanden werden. Heutzutage geschieht die Aufnahme der Kundenanforderungen meist in Form einer informalen Prosabeschreibung, die, wenn nicht sorgfältig geschrieben, zu Missverständnissen führen kann. Besonders wichtig ist die Qualität der Anforderungsspezifikation, wenn daraus weitere Softwareartefakte abgeleitet werden sollen, wie z.B. Entwurfsmodelle oder Testfälle. Je „formaler“ die Anforderungsspezifikation geschrieben wird, desto besser sind die Kommunikation zwischen den Anforderungsanalysten, den Entwicklern und den Testern und die Wiederverwendung der Anforderungsspezifikation in den weiteren Entwicklungsphasen.

Testbarkeit ist eine wichtige Eigenschaft von Softwareartefakten und wird in [5] als „das Maß an Nutzbarkeit der Softwareartefakte für die Testaktivitäten“ beschrieben. Techniken wie *Modellbasiertes Testen* (MBT) beschreiben, wie Software-Modelle für Aktivitäten des funktionalen Tests systematisch genutzt werden können. Dabei werden viele Testaktivitäten wie Testfallgenerierung und Testbewertung mit Hilfe von Modellen automatisiert. Wichtig ist dabei, dass die Modelle für die Nutzung in Testaktivitäten geeignet bzw. *testbar* sind. Nach dieser Definition kann die Testbarkeit der Anforderungsspezifikation damit gemessen werden, wie gut die Anforderungsspezifikation während der Testphase eingesetzt werden kann. Eine testbare Anforderungsspezifikation muss ausreichend Informationen enthalten, damit die Tester wissen, welche Eingaben nötig sind, um die Software zu kontrollieren und welche erwarteten Ausgaben die Software liefern muss.

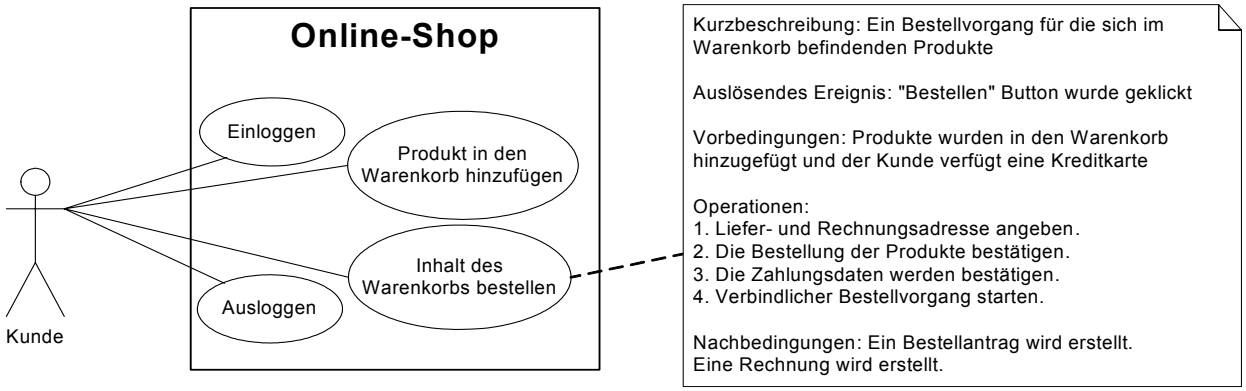

Abbildung 1: Anwendungsfall-Beschreibung

Die Spezifikation der funktionalen Anforderungen an die Software wird häufig mit Anwendungsfällen durchgeführt, die eine systematische Aufnahme von funktionalen Anforderungen ermöglichen und in UML [9] mit Anwendungsfall-Diagrammen modelliert werden. Abbildung 1 zeigt ein vereinfachtes Anwendungsfall-Diagramm für eine Online-Shop-Anwendung. Der Anwendungsfall „Inhalt des Warenkorbs

bestellen“ wird näher beschrieben. Die Beschreibung des Anwendungsfalls enthält Informationen u.a. zu benötigten Vorbedingungen, zu auszuführenden Operationen und zu erwarteten Nachbedingungen.

Anwendungsfall-basiertes Testen hat das Ziel, die Anwendungsfall-Beschreibungen mit ihren Vorbedingungen, Operationen und Nachbedingungen für das Testen systematisch zu nutzen. Anwendungsfälle mit sorgfältig spezifizierten Vorbedingungen, Operationen und Nachbedingungen sind gut testbar. Dabei dienen die Vorbedingungen als Testeingaben und die Nachbedingungen dienen als erwartete Testergebnisse, nachdem die spezifizierten Operationen ausgeführt wurden.

Wir beschreiben in diesem Beitrag ein Verfahren, wie Vor- und Nachbedingungen von Anwendungsfällen mit Hilfe von Modellen formalisiert und für Zwecke des modellbasierten Testens eingesetzt werden. Dabei beschränken wir uns bei der Formalisierung der Vor- und Nachbedingungen auf die fachlichen Daten, die von dem jeweiligen Anwendungsfall zur Ausführung benötigt und nach der Ausführung erzeugt werden. Ein fachliches Datenmodell beschreibt die fachlichen Daten und ihre Beziehungen zueinander. Nach der Implementierung der Anwendungsfälle werden diese formalen Vor- und Nachbedingungen zur automatisierten Testfallgenerierung, Testausführung und Testbewertung eingesetzt.

2 Visuelle Kontrakte

Zur Formalisierung der Vor- und Nachbedingungen von Anwendungsfällen benutzen wir *visuelle Kontrakte*. Visuelle Kontrakte wurden als eine UML-basierte Spezifikationsmethodik an der Universität Paderborn entwickelt [7]. Sie beschreiben Vor- und Nachbedingungen durch zwei UML-Objektdiagramme. Die Objektdiagramme sind getypt über einem fachlichen Datenmodell, modelliert durch ein UML-Klassendiagramm, das die fachlichen Daten und deren Zusammenhänge beschreibt (Abbildung 2, links).

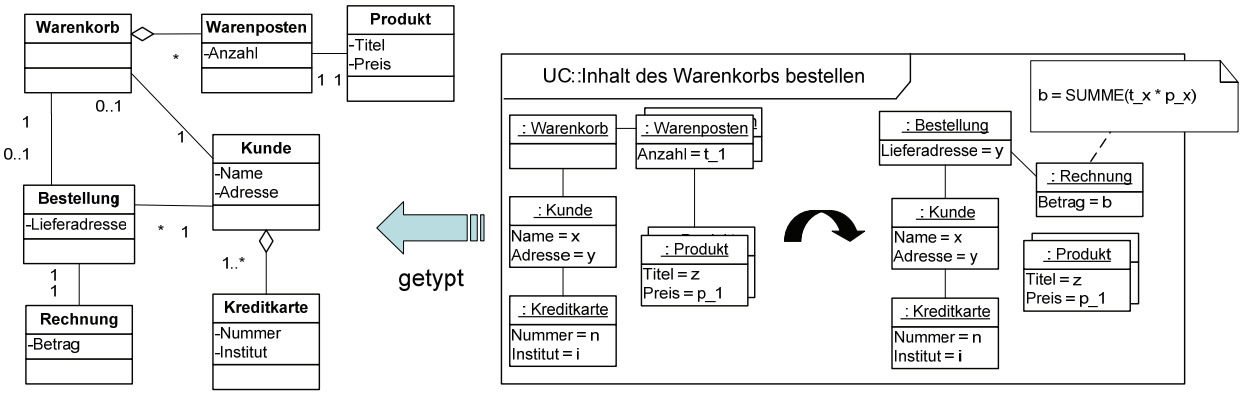

Abbildung 2: Fachliches Datenmodell für Online-Shop (links), Visueller Kontrakt (rechts)

Abbildung 2 (rechts) zeigt ein Beispiel für einen visuellen Kontrakt, der die Vor- und Nachbedingungen des Anwendungsfalls aus Abbildung 1 formalisiert. Inspiriert von der Idee des Design-by-Contract (DbC) [8] spezifiziert die linke Seite des Kontrakts die Vorbedingung und damit die Erwartungen der Software an fachlichen Daten, um den Anwendungsfall ausführen zu können. Die rechte Seite des Kontrakts spezifiziert die Nachbedingung und damit die nach der Ausführung des Anwendungsfalls garantierten und somit nachfolgenden Anwendungsfällen zur Verfügung gestellten fachlichen Daten, falls die Vorbedingung erfüllt wird.

Die einfache, intuitive Interpretation eines visuellen Kontrakts ist, dass jedes Objekt, welches nur auf der rechten Seite des Kontrakts vorhanden ist (:Bestellung, :Rechnung), neu erzeugt wird. Jedes Objekt, welches nur auf der linken Seite des Kontrakts vorhanden ist, wird gelöscht (:Warenkorb, :Warenposten). Objekte, welche sowohl auf der linken als auch der rechten Seite des Kontrakts stehen, sind von den Änderungen des Anwendungsfalls nicht betroffen. Dieser Formalisierung liegt die Theorie von Graphtransformationen zugrunde [3]. Es können auch Multiobjekte zur Darstellung von mehreren Objekten vom selben Typ eingesetzt werden (:Warenposten, :Produkt). Durch Parametrisierung der Objektattribute und Hilfs-

operatoren, wie z.B. Addition oder Multiplikation, können einfache Berechnungen ausgedrückt werden. Sprachen wie OCL können eingesetzt werden, um die Ausdrucksmächtigkeit der visuellen Kontrakte zu erhöhen (z.B. für die Berechnung des Rechnungswertes durch die Summe aller Preise der Warenposten).

3 Modellbasiertes Testen

Modellbasiertes Testen (MBT) beschreibt, wie Modelle systematisch für Testaktivitäten eingesetzt werden können. In [4] werden die drei wichtigen Aufgaben von MBT wie folgt definiert:

1. Generierung von Testfällen aus Modellen
2. Generierung von Testorakeln aus Modellen
3. Generierung der Testausführungsumgebung aus Modellen

Wir zeigen als nächstes, wie die durch die visuellen Kontrakte formalisierten Anforderungen für die oben aufgelisteten Aufgaben eingesetzt werden können. Als erstes zeigen wir, wie Testfälle aus visuellen Kontrakten generiert und ausgeführt werden.

3.1 Generierung und Ausführung von Testfällen

Ein Testfall besteht aus Testeingaben und erwarteten Testausgaben. Die Testeingaben für einen Anwendungsfall werden durch die Vorbedingung des visuellen Kontrakts bestimmt. Die Generierung der Testeingaben aus visuellen Kontrakten findet während der Testfallspezifikation in zwei Schritten statt [2]:

1. Generierung der Objekte, die in der Vorbedingung spezifiziert werden
2. Generierung der konkreten Werte für die Objektattribute

Im ersten Schritt werden die Objekte in der Vorbedingung und deren Verlinkung generiert. Für die Generierung der Multiobjekte werden Techniken wie Äquivalenzklassenbildung oder Grenzwertanalyse eingesetzt. In unserem Beispiel werden für die Multiobjekte :Warenposten und :Produkt nach Äquivalenzklassenbildung drei Testfälle generiert. In einem Testfall werden keine Objekte, in einem zweiten Testfall Objekte in einer beliebigen Anzahl generiert. In einem dritten Testfall werden die Objekte in der Anzahl der von der Spezifikation maximal zugelassenen Anzahl der Warenposten generiert.

In dem zweiten Schritt der Testfallspezifikation werden konkrete Attributwerte bestimmt. Auch hier werden Techniken wie Äquivalenzklassenbildung, Grenzwertanalyse oder Zufallsprinzip unter Berücksichtigung der Besonderheiten des Attributtyps verwendet. Für die Bestimmung von sinnvollen Attributwerten macht es durchaus Sinn, diese aus einer bereits existierenden Datenquelle mit fachlichen Daten zu beziehen. Abbildung 3 zeigt ein Beispiel für eine Testeingabe, in dem für die Multiobjekte jeweils zwei Objekte betrachtet wurden.

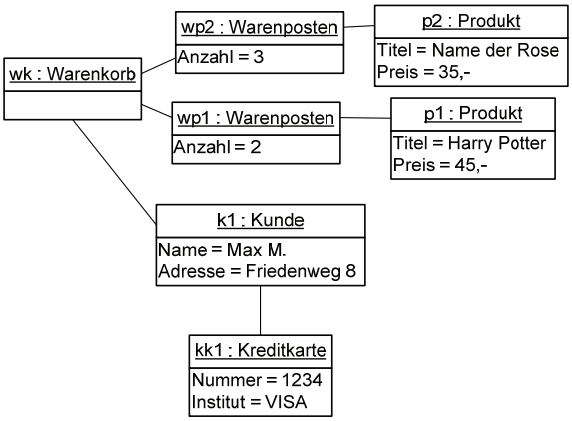

Abbildung 3: Beispiel für eine Testeingabe

Bevor die Testausführung startet, werden die Vor- und Nachbedingungen der visuellen Kontrakte zu Zusicherungen übersetzt und in die Testausführungsumgebung integriert. Diese werden während der Testaus-

führung die Eingaben der Testfälle und die Ausgaben des Anwendungsfalls überwachen und auf Korrektheit überprüfen.

Die Testausführung besteht aus drei Schritten: Testvorbereitung, Testaufruf und Testbewertung. Während der Testvorbereitung werden die Testeingaben in der Testumgebung erzeugt, damit die Vorbedingungen des zu testenden Anwendungsfalls erfüllt werden. Zur Zeit des Testaufrufs wird als erstes die Korrektheit der erzeugten Testeingaben durch die Zusicherungen geprüft. Dann werden die Operationen des Anwendungsfalls ausgeführt. Während der Testbewertung werden die Ausgaben des Anwendungsfalls von den Zusicherungen auf Korrektheit bezüglich der Nachbedingung des visuellen Kontraktes geprüft. Bei der Prüfung von Testausgaben dienen die Zusicherungen und damit die visuellen Kontrakte als Testorakel, was eine explizite Generierung der erwarteten Testausgaben während der Testfallspezifikation erübrigt.

3.2 Werkzeugunterstützung

In unserer Forschungsgruppe wurden mehrere Werkzeuge zur Einbindung der visuellen Kontrakte in den Entwicklungsprozess entwickelt. In [6] ist die Visual Contract Workbench (VCW) zum Editieren von Klassendiagrammen und visuellen Kontrakten beschrieben. Die VCW wurde erweitert um einen Plug-in für Testfallgenerierung und Testausführung [1]. Mit dieser Werkzeugunterstützung sind wir in der Lage, aus visuellen Kontrakten abstrakte und ausführbare Testfälle und Zusicherungen zu generieren, die Testfälle im Batch-Modus auszuführen und die Testausgaben zu bewerten.

4 Zusammenfassung

Wir haben in diesem Beitrag einen Ansatz zur Formalisierung der UML-Anwendungsfallsbeschreibungen vorgestellt, um Anwendungsfälle effektiv für Testzwecke einsetzen zu können. Dabei werden die textuellen Beschreibungen der Vor- und Nachbedingungen mit visuellen Kontrakten formalisiert. Die visuellen Kontrakte beschreiben die Änderungen bezüglich der fachlichen Daten nach der Ausführung des Anwendungsfalls. Mit visuellen Kontrakten können während der Testfallspezifikation Testeingaben generiert und während der Testausführung Testausgaben überprüft werden. Für visuelle Kontrakte wurden Werkzeuge entwickelt, die die Einbindung der visuellen Kontrakte in den Entwicklungs- und Testprozess ermöglichen.

5 Literatur

[1] Ellerweg, J.: Komponententest mit visuellen Kontrakten. Diplomarbeit, Universität Paderborn, 2008

[2] Engels, G., Güldali, B., Lohmann, M.: Towards Model-Driven Unit Testing. In Satellite Events at the MoDELS 2006, Revised Selected Papers, LNCS Volume 4364 / 2007, pp. 182 - 192, Springer Berlin / Heidelberg, 2007

[3] Heckel, R., Ehrig, H., Wolter, U., Corradini, A.: Double-Pullback Transitions and Coalgebraic Loose Semantics for Graph Transformation Systems. APCS (Applied Categorical Structures) 9(1) 83–110, 2001

[4] Heckel, R., Lohmann, M.: Towards Model-Driven Testing. Electr. Notes Theor. Comput. Sci. 82(6), 2003

[5] Jungmayr, S.: Reviewing Software Artifacts for Testability. EuroSTAR ’99, Barcelona, Spain, Nov. 8-12, 1999

[6] Lohmann, M.: Kontraktbasierte Modellierung, Implementierung und Suche von Komponenten in serviceorientierten Architekturen. Dissertation, Universität Paderborn, 2006

[7] Lohmann, M.; Sauer, S.; Engels, G.: Executable Visual Contracts. In Proc. of IEEE Symposium on Visual Languages and Human-Centric Computing, 2005, pp. 63-70, 2005

[8] Meyer, B.: Applying “Design by Contract”. IEEE Computer 25(10) 40–51, 1992

[9] Object Management Group: UML Specification V2.1.1. http://www.omg.org/cgi-bin/doc?formal/07-02-05, 2007