Testfallerzeugung aus Use-Case-Beschreibungen

Mario Friske
Fraunhofer FIRST
Kekuléstraße 7
D-12489 Berlin
mario.friske@first.fhg.de

Einleitung. Funktionale Anforderungen an Softwaresysteme werden heutzutage oft durch Use Cases erfasst. Neben UML-Diagrammen werden dazu auch textuelle Use-Case-Beschreibungen verwendet, insbesondere zur Spezifikation von Vorgangsbearbeitungssystemen. Diese Use-Case-Beschreibungen sind nicht nur Ausgangspunkt für die Systementwicklung, sondern sie können auch zur Erstellung von Testfällen für den Systemtest genutzt werden.

Im Folgenden wird zunächst die bei Fraunhofer FIRST verwendete Vorgehensweise zur manuellen Testfallerstellung aus Use-Case-Beschreibungen erläutert. Anschließend wird dargestellt, wie ein diese Methode unterstützendes Werkzeug unter Verwendung von Techniken der Model-Driven Architecture (MDA) realisiert werden kann.

Manuelle Testfallerstellung aus Use-Case-Beschreibungen. Jeder Schritt in einer Use-Case-Beschreibung beschreibt entweder eine Systemfunktion und den zugehörigen Aktor oder eine Systemreaktion. Sofern die Schritte komplexer sind, lassen sie sich in aufeinanderfolgende Systemfunktionen und Systemreaktionen auflösen. Um die Testfallerstellung vorzubereiten, wird zunächst eine Liste aller Systemfunktionen und Systemreaktionen erstellt.

Im ersten Schritt der Testfallerstellung werden die Use-Case-Beschreibungen in logische Testfälle überführt. Diese repräsentieren jeweils ein zu testendes Szenario. Auf der logischen Ebene werden nur die erforderlichen Äquivalenzklassen festgelegt, jedoch keine konkreten Testdaten.

Im zweiten Schritt werden aus den logischen Testfällen konkrete Testfälle erzeugt. Dazu werden konkrete Daten, d.h. Repräsentanten der zuvor festgelegten Äquivalenzklassen, eingefügt. Weiterhin wird festgelegt, welche GUI-Funktionen zum Aufruf der Systemfunktionen zu nutzen sind. Testfälle auf konkreter Ebene erfordern keine Entscheidungen seitens des ausführenden Testers mehr.

Im dritten Schritt werden die konkreten Testfälle in ausführbare Testskripte überführt. Dazu wird zunächst für jede Systemfunktion eine entsprechende Funktion in der Skriptsprache erstellt. Diese Funktionen ermöglichen eine direkte Umsetzung der textuellen konkreten Testfälle in modular aufgebaute, ausführbare Testskripte.

Die sofortige Durchführung aller drei Transformationsschritte ist nur bei entsprechendem Reifegrad der zu testenden Applikation sinnvoll. Unsere Erfahrungen haben gezeigt, dass in entwicklungsbegleitenden manuellen Tests auf der Grundlage logischer Testfälle ein Großteil der signifikanten Fehler, d.h. fehlende Funktionalitäten bzw. Fehlinterpretationen der textuellen Spezifikation, gefunden wird. Dabei wird die Auswahl der benutzten Daten und GUI-Funktionen dem Tester überlassen, weshalb Testläufe auf der Basis logischer Testfälle im Allgemeinen nicht vollständig reproduzierbar sind. Erst nach erfolgreichem Durchlauf der logischen Testfälle werden konkrete Testfälle erstellt und ausgeführt, um geforderte Abdeckungskriterien zu erreichen. Die nachfolgende, aufwändige Erstellung ausführbarer Testskripte ist nur bei hinreichender Stabilität der Oberfläche und einer großen Anzahl zu erwartender Regressionstests sinnvoll.

Das stufenweise Vorgehen hat sich bei funktionalen Tests auf der Basis von Anwendungsfallbeschreibungen bewährt. Problematisch war dabei jedoch, dass alle Transformationsschritte manuell durchgeführt werden mussten, da keine adäquate Werkzeugunterstützung vorhanden war.

Modellbasierte Entwicklung und modellbasierter Test. Modellbasierte Entwicklungsverfahren sind Gegenstand vieler aktueller Forschungs- und Entwicklungsarbeiten. Sie alle haben gemeinsam, dass ein zentraler Bestandteil der Entwicklung Modelle sind, aus welchen Code generiert wird. In der Model-Driven Architecture (MDA) [6] wird die Codegenerierung mithilfe von mehrstufigen Transformationen realisiert, wobei aus einem plattformunabhängigen Model (PIM) zunächst ein plattformspezifisches Model (PSM) erzeugt wird, aus welchem anschließend der Code generiert wird, siehe Abbildung 1.

Mit den heute verfügbaren UML-basierten Technologien lässt sich eine vollständige Codegenerierung gänzlich ohne notwendige manuelle Ergänzungen oder Nachbesserungen des Codes jedoch noch nicht erreichen. In [7] wurde dieses Ziel, *models only*, als Modelling Maturity Level (MML) 5 definiert. Das derzeit üblicherweise erreichbare Level an Präzision von Modellen ist MML 4, *precise models*, d.h. den Programmierern obliegen keine geschäftsprozessspezifischen Entscheidungen mehr, sondern sie müssen eine Umsetzung realisieren, die der vorgegebenen Spezifikation entspricht. Genau dieses Level an Modellpräzision ist notwendige Voraussetzung für den modellbasierten Test, bei dem zu entscheiden ist, ob die Implementie-

rung eine Realisierung der Spezifikation ist. Beim modellbasierten Test lässt sich ebenfalls ein mehrstufiges Verfahren anwenden, indem aus dem PIM zunächst plattformunabhängige Testfälle (PIT) erzeugt werden, welche in plattformspezifische Testfälle (PST) transformiert werden [2]. Die PST können anschließend in ausführbare Testskripte überführt werden.

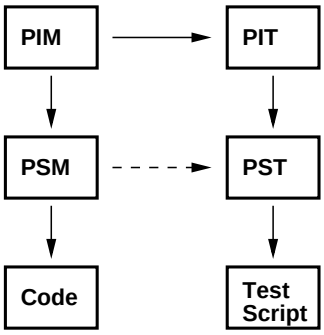

Abbildung 1: Mehrstufige Transformationen beim Model-Driven Testing

In Use-Case-Beschreibungen werden typische Nutzer-System-Interaktionen auf plattformunabhängige Weise spezifiziert, dementsprechend können sie als Teil des PIM betrachtet werden. Die daraus erstellten logischen Testfälle enthalten ebenfalls noch keine Plattformabhängigkeiten, wie GUI-Funktionen, sie sind plattformunabhängige Testfälle (PIT). Die daraus generierten konkreten Testfälle enthalten Abhängigkeiten von der konkreten Realisierung der grafischen Oberfläche, entsprechend sind sie plattformspezifische Testfälle (PST).

MDA-basierte Werkzeugunterstützung Obwohl, wie eingangs beschrieben, bei der Testfallerstellung aus Use-Case-Beschreibungen ebenfalls eine mehrstufige Transformation verwendet wird, lassen sich die MDA-basierten Verfahren nicht sofort anwenden, da die Use-Case-Beschreibungen in der Regel nicht präzise genug sind. Deshalb ist zunächst eine Aufbereitung der Use-Case-Beschreibungen notwendig, um die geforderte Präzision zu erreichen. Dazu werden in einem werkzeuggestützten, interaktiven Prozess den einzelnen Schritten der Use-Case-Beschreibungen die benutzten Systemfunktionen und -reaktionen zugeordnet und als Metadaten gespeichert. Dabei wird eine Liste aller Systemfunktion und -reaktionen erstellt.

Bislang hat sich noch kein einheitliches Format für Use-Case-Beschreibungen durchgesetzt. Um die werkzeuggestützte Aufbereitung der Use-Case-Beschreibungen zu ermöglichen, ist es deshalb notwendig, dem Werkzeug eine Beschreibung des Aufbaus der Use-Case-Beschreibungen zu geben. Zur Beschreibung des Modelles einer Use-Case-Beschreibung, des Metamodelles, kann die Meta-Object-Facility (MOF) [5] verwendet werden.

Use-Case-Beschreibungen lassen sich in die vier Modellierungsebenen der OMG folgendermaßen einordnen: Szenarien sind die Instanzen (M0), eine Use-Case-Beschreibung beschreibt die Vielzahl der möglichen Szenarien, d.h. sie ist das Modell (M1). Die Beschreibung des Aufbaus der Use-Case-Beschreibungen ist das Metamodell (M2).

Auf die gleiche Art und Weise lassen sich auch logische und konkrete Testfälle einordnen: Ein konkreter Testfall ist eine Instanz (M0) eines logischen Testfalls (M1), die Beschreibung des Aufbaus der Testfälle ist das Metamodell (M2).

Die Zuordnung zu den einzelnen Ebenen ist in Abbildung 2 tabellarisch dargestellt.

Modell-ebene	Transformations-Quelle	Transformations-Ziel
M3	MOF	MOF
M2	Modell der Use-Case-Beschreibung	Modell der Test-fallbeschreibung
M1	Use-Case-Beschreibung	logischer Testfall
M0	Szenario	konkreter Testfall

Abbildung 2: Einordnung von Use-Case-Beschreibungen und Testfällen in die Modellebenen

Da sich Use-Case-Beschreibungen und Testfälle in die vier Modellebenen einordnen lassen, lassen sich die eingangs beschriebenen manuellen Überführungen als automatisierte Tranformationen unter Verwendung von MOF-basierten Transformationswerkzeugen realisieren. Entsprechend können bekannte Algorithmen zur Erstellung von Testfällen aus Use Cases, beispielsweise die in [1] vorgestellten Verfahren, in die Transformationsbeschreibungssprachen dieser Werkzeuge übertragen werden.

Beispielhafte Anwendung Das Verfahren soll nun an einem Beispiel veranschaulicht werden. Grundlage der Testfallerstellung soll ein Use Case eines Online-Buchungssystems sein:

```
Basic Path
----------
1. The use case starts when the customer
   selects Place Order.
2. The system displays the Place Order
   screen.
3. The customer enters his name
   and address.
4. The customer enters product codes for
   products to be ordered.
   ...
```

In diesem Beispiel wird in den Schritten 1, 3 und 4 eine Systemfunktion aufgerufen und in Schritt 2 erfolgt eine Systemreaktion. In Abbildung 3 ist ein stark vereinfachtes Modell dieser Use-Case-Beschreibung

dargestellt. Es zeigt, dass der Use Case aus Schritten besteht, welche entweder eine Systemfunktion oder eine Systemreaktion sind. Jeder Schritt kann einen weiteren Schritt als Nachfolger haben.

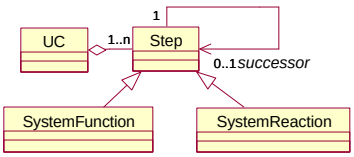

Abbildung 3: Vereinfachtes Modell einer Use-Case-Beschreibung

Die Use-Case-Beschreibung soll nun in den in Abbildung 4 dargestellt logischen Testfall übertragen werden. Das in Abbildung 5 dargestellte, vereinfachte Modell dieses logischen Testfalls zeigt, dass jeder Schritt aus einer Aktion und zugehöriger Systemreaktion besteht, welche beide jeweils null oder einmal pro Schritt vorhanden sein können.

No	Action	Expected Result
1	customer selects "Place Order"	system displays place order screen
2	customer enters name and address	
3	customer enters product codes	
4	...	

Abbildung 4: Logischer Testfall

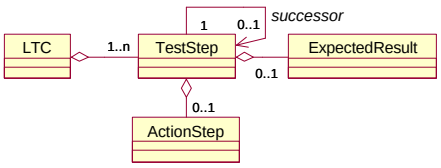

Abbildung 5: Vereinfachtes Modell des logischen Testfalls

Die Transformation der Use-Case-Beschreibung (PIM) in die logischen Testfälle (PIT) erfolgt nach folgenden Transformationsregeln:

- Für jeden Schritt vom Typ *SystemFunction* im PIM wird eine *Action* im PIT erzeugt.
- Für jeden Schritt vom Typ *SystemReaction* im PIM wird ein *ExpectedResult* im PIT erzeugt.
- Falls der Nachfolger eines Schrittes vom Typ *SystemFunction* ein Schritt vom Typ *SystemReaction* ist, dann werden beide dem gleichen Testschritt zugeordnet, ansonsten werden sie zwei aufeinanderfolgenden Testschritten zugeordnet.

Diese Transformationsregeln lassen sich nun weiter detaillieren und, wie in [3] beschrieben, in eine Transformationsbeschreibungssprache übertragen. Analog lassen sich Tranformationsregeln zum Überführen der logischen Testfälle in konkrete Testfälle erstellen. In diesem Schritt ist es notwendig, die Algorithmen zum Erreichen vorgegebener Überdeckungskriterien umzusetzen. Entsprechend sind die erforderlichen Transformationen wesentlich komplexer als die hier skizzierten.

Status und Ausblick Es wurde beschrieben, wie aus Use-Case-Beschreibungen Testfälle erzeugt werden können. Weiterhin wurde skizziert, wie unter Verwendung von MDA-Techniken ein unterstützendes Werkzeug realisiert werden kann. Die Anforderungsanalyse und das Use-Case-Modell einer solchen Applikation ist in [4] dargestellt. Der Einsatz eines solchen Werkzeuges wird sowohl eine effizientere Erstellung von Testfällen ermöglichen als auch zu einer Erhöhung der Systematik und folglich zu einer verbesserten Qualität der erzeugten Testfälle beitragen.

Zurzeit wird das Erfassen notwendiger GUI-Informationen mittels Capture/Replay-Tools konzipiert und die Umsetzung des Transformationsalgorithmus mittels eines Transformationswerkzeuges realisiert. Anschließend wird ein prototypisches Werkzeug realisiert und die Methode evaluiert werden.

Literatur

[1] Robert V. Binder. *Testing Object-Oriented Systems: Models, Patterns, and Tools*. Object Technology Series. Addison Wesley, 1999.

[2] Reiko Heckel and Marc Lohmann. Towards model-driven testing. *Electronic Notes in Theoretical Computer Science*, 2003.

[3] Anneke Kleppe, Jos Warmer, and Wim Bast. *MDA Explained: The Model Driven Architecture - Practice and Promise*. Addison-Wesley, 2003.

[4] Mike Mader. Designing tool support for use-case-driven test case generation. Diplomarbeit, FHTW Berlin, 2004.

[5] Object Management Group. *Meta Object Facility (MOF) Specification*. OMG, http://www.omg.com/, 2003.

[6] Object Management Group. *Model Driven Architecture (MDA)*. OMG, http://www.omg.com/mda/, 2003.

[7] Jos Warmer and Anneke Kleppe. *The Object Constraint Language: Getting Your Models Ready for MDA*. Addison-Wesley, 2nd edition, 2003.