Testen mit Use-Cases

Chris Rupp (Chris.Rupp@sophist.de) und Dr. Stefan Queins (Stefan.Queins@sophist.de)

Abstract

In vielen Unternehmen werden zur Zeit Use-Cases mit dem großen Versprechen eingeführt, dass sich Effekte wie einfache Lesbarkeit, Wiederverwendbarkeit und automatisch entstehende Test-Cases direkt einstellen. Diese Ergebnisse treten aber in keinem Fall von vornherein ein, sie sind ein Resultat zielgerichteter und angepasster Anwendung von Use-Cases. In diesem Artikel soll beschrieben werden, wie systematisch aus Use-Cases Test-Cases abgeleitet werden können. Wir geben konkrete Vorgaben, wie Use-Cases zu formulieren sind und erläutern weitere Notationen, die das Vorgehen unterstützen.

1 Ein komplexes Systemumfeld fordert einen durchdachten Prozess

Ein Großteil technischer Systeme sind sogenannte ERT-Systeme, in welchen Hard- und Software miteinander kombiniert sind.

Um die vermehrte Präsenz von ERT-Systemen in der Realität zu verdeutlichen, eignet sich das Beispiel des Automobils. In diesem Bereich hält Software immer größeren Einzug und ein Ende dieses Trends ist nicht abzusehen.

Es ist nicht gerade einfach, derartige Software-/Hardware-Systeme zu spezifizieren, da diese auf Grund der starken Einbettung und Vernetzung der Einzelsysteme und der großen Anzahl von Ausnahmefällen äußerst komplex sind.

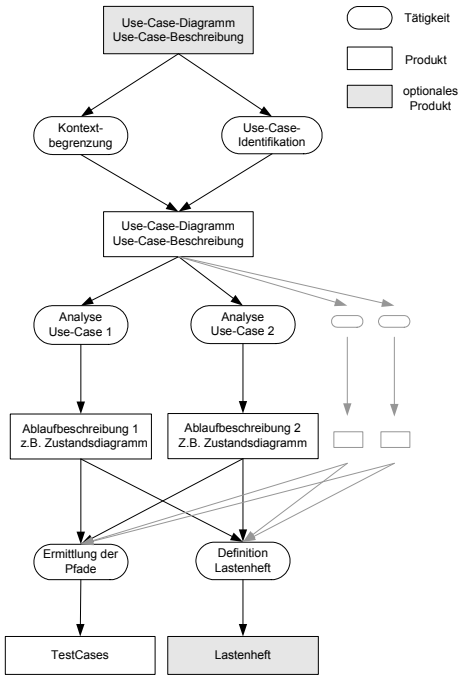

Abbildung 1: Überblick „Vom Use-Case zum Test-Case“

Die systematische Analyse und der darauf basierende Test stellen kritische Schritte in der Entwicklung dar. Dieser Artikel soll die Schritte der Systemanalyse und Ableitung der Test-Cases aus erstellten Analysedokumenten darstellen. In der Abbildung 1 sind die Schritte dargestellt, die einen Überblick über das Vorgehen verdeutlichen sollen. ERT-Systeme kommunizieren meist intensiv mit ihrer Umgebung, können sogar baulich ineinander verankert sein. Es muss daher explizit geklärt werden, was innerhalb und außerhalb der Systemgrenzen liegt. Wir stellen diese Ergebnisse in einem Use-Case-Diagramm dar, welches uns neben der Kontextabgrenzung noch den weiteren Dienst erweist, unser System aus einer Black-Box-Betrachtungsweise in handhabbare Stücke zu zerlegen.

Wir stellen eine Lösung vor, die unserer Meinung nach die meisten Vorteile vereint und verhindert, dass wir bei komplexen technischen Systemen sofort im Chaos der Ausnahmebedingungen versinken. Im Automobilbereich haben wir es häufig mit reaktiven Systemen zu tun. Deshalb drängten sich für die detailliertere Spezifikation des Systemverhaltens Zustandsdiagramme auf. Sie liefern eine gute Grundlage für die Spezifikation des Systemverhaltens auf verschiedenen Detaillierungsstufen und dienen als formale Basis, um automatisierte Test-Cases erzeugen und reduzieren zu können.

2 Das Beispiel: Die Scheibenwischersteuerung

In diesem Artikel haben wir uns für die Scheibenwischersteuerung als geläufiges Beispiel entschieden. Wir haben zur besseren Übersicht ein reales Scheibenwischersystem auf vier Nachbarsysteme und eine geringe Anzahl von Ereignissen reduziert. Bedient wird das System mit dem Lenkstockhebel. Die Funktionalität wurde auf kontinuierliches Wischen, Tippwischen und Intervallwischen eingegrenzt. Die Energieversorgung ist für die richtige Spannung zuständig, der Wischersensor teilt die aktuelle Position des Wischers mit (Parkposition oder im Sichtfeld des Fahrers).

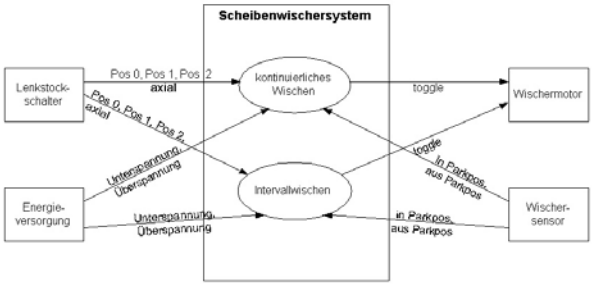

Abbildung 2: Vereinfachtes Use-Case-Diagramm des Scheibenwischersystems

3 Der erste Schritt: Use-Cases

Wir betrachten das System von außen, als Black-Box, und suchen Akteure, von denen wir wissen wollen, was sie von unserem System erwarten. Diese Akteure sind in unserem Fall Nachbarsysteme, die (wie gefordert) mit dem System kommunizieren und Erwartungen an sein Verhalten haben.

3.1 Kontextabgrenzung und Use-Case-Identifikation

Wir müssen Systemabgrenzungen und Schnittstellenpräzisierungen durch UML-konforme Diagramme darstellen, da die UML in der aktuellen Version keine Kontextdiagramme enthält. Mittels Use-Case-Diagrammen lässt sich der logischen Kontext (Kommunikation mit Nachbarsystemen) gut abgrenzen. Folgende Konventionen sollten beachtet werden:

- Exakte Benennung des Systems
- Suchen aller Nachbarsysteme, auch solche, die nur Ereignisse liefern oder nur empfangen
- Echte Akteure als Initiator kennzeichnen (Stereotyp <<initiator>>)
- Strichmännchen nur für menschliche Akteure, das Klassensymbol für Nachbarsysteme
- Finden und dokumentieren aller möglichen Ereignisse aller Akteure, um alle möglichen Test-Cases ableiten zu können
- Nehmen Sie die Sicht der Nachbarsysteme ein, um die Erwartungen des Nachbarsystems an Ihr System zu finden
- Spezifizieren Sie Informationen über Ein- und Ausgaben mit Assoziationen zwischen System und Umgebung, wie im Klassendiagramm
- Bei vielen Ereignissen oder Zusatzinformationen sollten Sie eine Tabelle mit allen Informationen hinzufügen
- Trennen Sie zwei Use-Cases, wenn unterschiedliche Vor- oder Nachbedingungen vorhanden sind (siehe [ABr02], [Oes01])

3.2 Erstellung der Use-Case-Beschreibung

Um Use-Cases zu beschreiben werden die essentiellen Schritte – logische und technologieneutrale Einzelschritte, die für die Ausführung des Prozesses unerlässlich sind – der Use-Cases und Ausnahmefälle gesucht, wobei nur eine Konzentration auf die Auslöser dieser Ausnahmefälle stattfinden sollte.

Name	kontinuierliches Wischen	
Akteure	Lenkstockschalter, Energieversorgung, Wischermotor, Wischersensor	
Auslösendes Ereignis	Lenkstockschalter in Position 2 oder Lenkstockschalter axial	
Kurzbeschreibung	Vom Lenkstockschalter wird ein kontinuierliches Wischen angefordert. Das System setzt diesen Wunsch in Steuersignale für den Wischermotor um.	
Vorbedingungen	Zündung mind. in Stellung 1 (Radiostellung), es wird nicht gewischt oder der Wischer befindet sich im Intervallwischen.	
Essenzielle Schritte	**Intention der Systemumgebung**	**Reaktion des Systems**
	Lenkstockschalter -> Pos 2 (Kontinuierliches Wischen)	Die normale Wischfunktion mit einer vorgegebenen Basiswischergeschwindigkeitsstufe W=WBASIS wird aktiviert. Das Event toggle an den Wischermotor wird erzeugt, falls Wischer in Parkposition.
	Lenkstockschalter -> axial (Tippwischen)	Falls der Wischer in Parkposition ist und der Lenkstockschalter sich nicht in Pos 2 befindet, wird das Event toggle an den Wischermotor erzeugt. Ein Wischzyklus mit einer vorgegebenen Basiswischergeschwindigkeitsstufe W=WBASIS wird aktiviert. Sobald der Wischer am Ende des Wischzyklus sich in Parkposition befindet wird das Event toggle an den Wischermotor erzeugt.
	Lenkstockschalter -> Pos 0 (Wischen deaktivieren) oder Lenkstockschalter -> Pos 1 (Intervallwischen)	Der kontinuierliche Wischvorgang wird beendet. Das Event toggle an den Wischermotor wird erzeugt, sobald Wischer in Parkposition.
Ausnahmefälle	• Liegt die Spannung U für t=tSPANNUNG_AUS unter U=ULOW1 (Energieversorgung -> Unterspannung), so wird der Wischer wegen Unterspannung abgeschaltet. Ein Einschalten ist erst wieder bei einer Spannung über U=ULOW2 für t=tSPANNUNG_EIN möglich. • Beträgt die Spannung U für t=tSPANNUNG_AUS einen Wert über UHIGH2 (Energieversorgung -> Überspannung), so wird der Wischer wegen Überspannung abgeschaltet. Ein Einschalten erfolgt erst wieder bei einer Spannung unter UEIN2 für t=tSPANNUNG_EIN.	
Nachbedingung	Das System kehrt in vorherigen Zustand zurück (Aus oder Intervallwischen)	

Die Beschreibung der Use-Cases sollte mindestens folgende Punkte enthalten:

- Name
- Akteure
- Auslösendes Ereignis
- Kurzbeschreibung
- Vorbedingung
- Essentielle Schritte
- Trigger der Ausnahmefälle
- Nachbedingung

4 Die Formalisierung: UML-Verhaltensdiagramme

Das hohe Abstraktionsniveau des Black-Box-Verhaltens des erstellten Systems soll nun in eine formale Notation überführt werden. Aus jedem Use-Case erstellen wir eine Prozessablaufbeschreibung, die das Verhalten des Use-Cases losgelöst vom Gesamtsystem betrachtet.

Als Notation wird ein Aktivitätsdiagramm (Ablauforientierte Prozesse) oder ein Zustandsdiagramm (Reaktive Prozesse) vorgeschlagen. Wir werden uns hier auf Zustandsdiagramme beschränken.

Da es nicht möglich ist, automatisch Use-Cases in Zustandsdiagramme zu überführen, muss dieser Schritt manuell durchgeführt werden. Dieses Vorgehen ist in [HRu02] detailliert erläutert.

Wir schlagen vor, zunächst nur die essentiellen Schritte zu überführen. Die Detaillierung durch Hinzufügen von Ausnahmefällen (Verfeinerung) wird später vorgestellt.

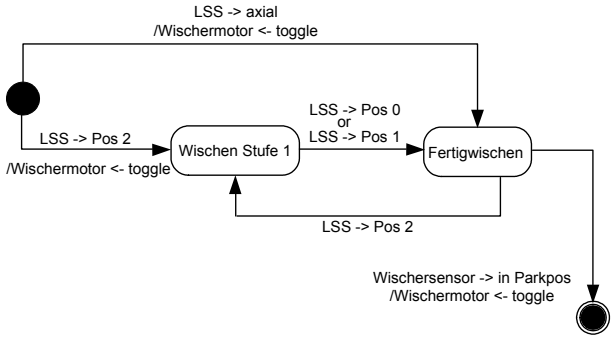

Abbildung 3: Zustandsdiagramm „Kontinuierliches Wischen“

5 Die Generierung: Vom Pfad zum Test-Case

Unsere nächste Aufgabe besteht darin, aus den Zustandsdiagrammen der Use-Cases möglichst effizient Test-Cases herzuleiten. Die zu testenden Abläufe entsprechen den Pfaden in den Verhaltensdiagrammen, wobei wir die eingehenden Ereignisse als Auslöser für die Übergänge ansehen. Anfangs- und Endpunkte dieser Pfade sind durch Modellierung vorgegeben (Start- und Endzustand im Graphen).

Ein vollständiger Test setzt voraus, dass alle möglichen Pfade vom Start- zum Endpunkt betrachtet werden. Auftretende Zyklen sollten zunächst nur einmal durchlaufen werden. Für jeden ermittelten Pfad des Zustandsdiagramms kann dann ein Test-Case identifiziert werden, wobei wir uns an die Testdokumentenspezifikation ANSI/-IEEE 829 halten. Die Vor- und Nachbedingungen des Test-Case werden aus dem entsprechenden Use-Case abgeleitet.

6 Die Komplettierung: Verfeinerung und Komposition

Die bisherigen Test-Cases, welche nur die essentiellen Schritte der Use-Cases enthalten, sind nicht sehr gehaltvoll. Deshalb erweitern wir das Vorgehen und verfeinern die Zustandsdiagramme, damit realitätsnahe Test-Cases entstehen. Da wir außerdem das Gesamtverhalten des Systems betrachten, müssen wir eine Verbindung zwischen den Use-Cases und somit auch zwischen den Zustandsdiagrammen herstellen.

6.1 Verfeinerung

Sie können die Überführung der Use-Cases in Zustandsdiagramme iterativ-inkrementell durchführen. Entscheiden Sie, ob Sie zuerst die Use-Cases sehr abstrakt beschreiben und diese dann in einer weiteren Iteration präzisieren oder ob Sie zuerst nur einen Teil der Use-Cases (z.B. essentielle Schritte) betrachten und diese in ein Zustandsdiagramm überführen, welches Sie dann inkrementell um die Ausnahmefälle erweitern.

6.2 Komposition

Use-Cases, die durch include- oder extend-Beziehungen verbunden sind, ermöglichen z.B. das Anstoßen eines Use-Cases aus der Bearbeitung eines Anderen heraus. Außerdem kann eine gleichzeitige Abarbeitung von Use-Cases von außen her getriggert werden (quasi-parallele Abarbeitung oder mehrfache Instanziierung). Aus diesem Grund ist eine getrennte Betrachtung der Use-Cases nicht ausreichend, um für das Gesamtverhalten des Systems umfassende Test-Cases abzuleiten. Die formale Beschreibung der Use-Cases durch Zustandsdiagramme erlaubt uns ein systematisches Vorgehen, um die Use-Cases zusammenzuführen und das Verhalten des Gesamtsystems zu beschreiben. Durch das Übereinanderlegen (Komposition) einzelner Zustandsdiagramme entsteht eine Systemablaufbeschreibung. Dabei müssen alle möglichen Kombinationen von Zuständen betrachtet werden. Kombinierte Zustände, die in der Realität nie erreicht werden, können jedoch gestrichen werden.

6.3 Kombination der Verfeinerung und Komposition

Es stellt sich die Frage, ob Sie Zustandsdiagramme einzeln verfeinern und danach miteinander kombinieren, oder ob zuerst eine Komposition stattfinden soll und die daraus resultierende Systemablaufbeschreibung verfeinert wird. Die frühe Komposition hat zum Vorteil, dass nur noch ein Zustandsdiagramm verfeinert werden muss. Diese Beschreibung ist aber um einiges komplexer als die Verfeinerung der einzelnen Zustandsdiagramme. Die späte Komposition hat den Vorteil der relativ einfachen Verfeinerung. Die eigentliche Komposition ist aber wesentlich komplexer, da detaillierte Zustandsdiagramme miteinander kombiniert werden müssen. Welchen Weg Sie gehen wollen, hängt im Wesentlichen davon ab, ob und welche abstrakteren Zustandsdiagramme Sie zum Herleiten von Test-Cases benutzen.

7 Der Ausblick

Die Komplexität der Modelle ist sicherlich ein Problem, welches Ihnen sicherlich auch auffällt. Es bietet sich eine rechnergestützte Findung der Pfade an. Dabei kann zusätzlich eine Interaktion des Testers mit einfließen, um aktuell nicht relevante Testfälle ausschließen.

Bei endlichen Automaten kann das Testen reduziert werden, indem jeder Übergang nur mindestens einmal besucht werden muss. Diese Reduktion findet aber keine Anwendung bei erweiterten endlichen Automaten, da bei diesen der Weg, wie man in einen Zustand gelangt ist, eine wichtige Rolle spielt.

Der Tester hat die Möglichkeit, sich zwischen wenigen langen, beziehungsweise vielen kurzen Wegen zu entscheiden. Viele kurze Testfälle bieten eine leichtere Fehlerlokalisierung, es ist aber eine immer wiederkehrende Wiederherstellung der Testumgebung notwendig. Wenige lange Testpfade reduzieren die Gesamtanzahl der Test-Cases. Um möglichst lange oder kurze Testpfade zu finden, bietet es sich an, Heuristiken und Wegefindungsalgorithmen [Len94] anzuwenden.

Literaturliste:

ABr02 Steve Adolph und Paul Bramble
The Agile Software Development Series
Patterns for Effective Use Cases
Addison-Wesley, 2002

HRu02 Peter Hruschka, Chris Rupp
Agile Softwareentwicklung für Embedded und Real-Time Systems mit der UML
Hanser Verlag, 2002

Len94 Lengauer, Thomas
Combinatorial algorithms for integrated circuit layout
Teubner Verlagsgesellschaft und Wiley, 1994

Oes01 Bernd Oestereich
Objektorientierte Softwareentwicklung – Analyse und Design mit der Unified Modeling Language
5. Auflage, Oldenburg Verlag, 2001

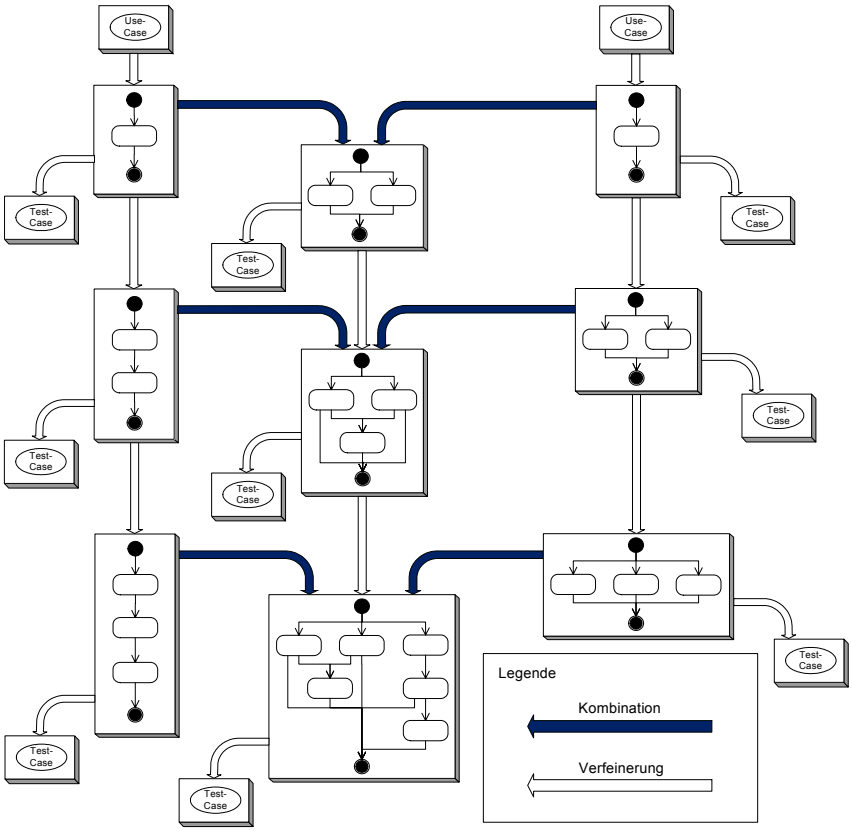

Abbildung 4: Hierarische Test-Case-Erstellung