Chris Rupp, Stefan Queins

Vom Use-Case zum Test-Case – strategische Partner für Ihren Projekterfolg

Abstract

Herrscht bei Ihnen im Unternehmen auch gerade die Vision, in Zukunft alles mittels Use-Cases zu spezifizieren? In vielen Unternehmen werden Use-Cases derzeit mit viel Heilsversprechungen eingeführt und dabei Effekte wie einfache Lesbarkeit, automatisch herauspurzelnde Testfälle und gute Wiederverwendbarkeit versprochen. All die Effekte treten keineswegs automatisch sondern nur bei sehr zielgerichteter und angepasster Anwendung von Use-Cases auf. Wir beschreiben in diesem Artikel ein Vorgehen, systematisch Testfälle aus Use-Cases abzuleiten. Hierzu geben wir konkrete Vorgaben an, wie Use-Cases für diesen Zweck zu formulieren sind und welche weiteren Notationen deren Anwendung unterstützen. Um ein sehr realitätsgetreues Szenario aufzuzeigen, setzen wir nicht auf der grünen Wiese auf. Wir erstellen Use-Cases und Test-Cases für einen Teil eines komplexen Gesamtsystems, wobei die Aufteilung des Systems in einzelne Teilsysteme bereits vorgegeben wird.

1. Ein komplexes Systemumfeld fordert einen durchdachten Prozess

Gehören Sie zu der vergessenen Mehrheit der Menschen, die sich mit der Entwicklung technischer Systeme befasst? Beim Lesen gängiger Computerliteratur, insbesondere zum Thema Objektorientierung und Unified Modeling Language (UML [UML]), könnte man zu der Ansicht gelangen, dass die meisten Computer Windows verwenden und in Desktopgehäusen untergebracht sind. In der Realität übersteigt die Anzahl der technischen Systeme, z.B. der eingebetteten Echtzeitsysteme (RTE-Systeme), die Anzahl der deutlich sichtbareren PC-Verwandten allerdings bei weitem. Ein europäischer oder amerikanischer Haushalt hat im Normalfall einen, vielleicht sogar zwei PCs zu bieten. Dem stehen Dutzende elektronischer Geräte gegenüber, die Prozessoren und Software enthalten. Ihr Automobil ist ein brillantes Beispiel, um den Einzug von RTE-Systemen in unsere Realität zu verdeutlichen. Der Anteil an Software stieg in den letzten Jahren kontinuierlich an und dieser Trend wird, wie Abbildung 1 zeigt, weiter anhalten. Schuld daran sind Funktionalitäten, die Ihre Sicherheit und Ihren Fahrkomfort sicherstellen, wie z.B. ABS, ESP oder elektronische Einparkhilfen, aber auch Navigations- und Web-Informationssysteme. Dieser Trend ist derzeit allerdings nicht nur für den Automobilbereich typisch, sondern spielt sich in vielen Bereichen wie z.B. auch in der Haushaltselektronik oder Steuerungs- und Automatisierungstechnik ab.

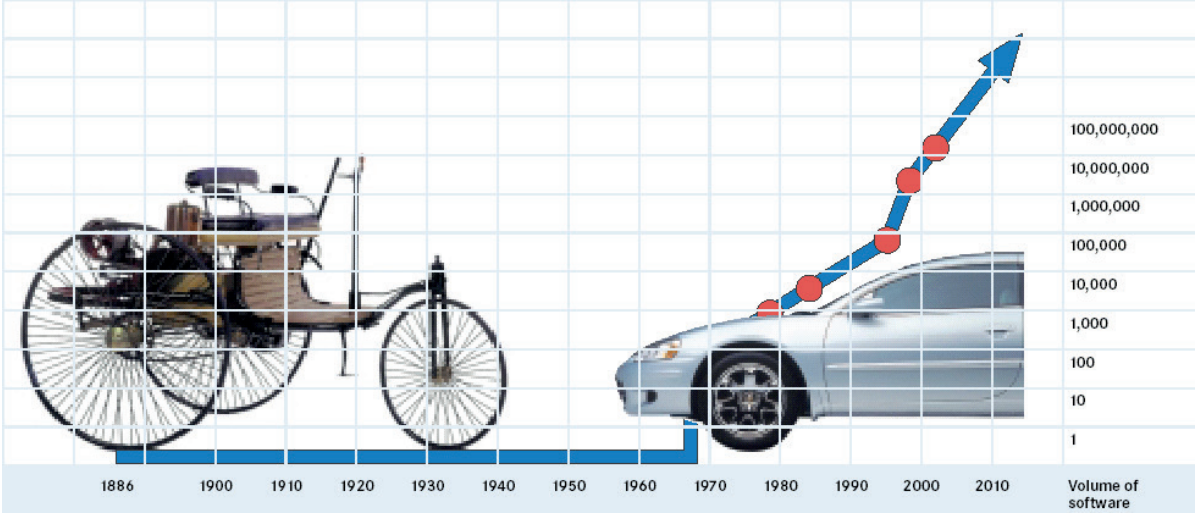

Abbildung 1: Software als wichtiger Bestandteil eines Automobils am Beispiel DaimlerChrysler [High02]

Der Prozess, derartige Software-/Hardware-Systeme zu spezifizieren, ist auf Grund der starken Einbettung und Vernetzung der einzelnen Systeme, der hohen Komplexität und der großen Anzahl an Ausnahmebedingungen nicht gerade einfach. Deshalb ist gerade hier ein Vorgehen von Interesse, das Sie systematisch von der ersten groben Abgrenzung Ihres Systems von der Umwelt bis hin zum Test begleitet. Die systematische Analyse des Systems und der darauf basierende Test stellen momentan kritische Schritte in dieser Prozesskette dar. Im Rahmen von Forschungsprojekten mit Kunden im Bereich eingebetteter Systeme wurden die Schritte eines derartiges Vorgehens diskutiert, erprobt und dokumentiert. Als Ergebnis der gemeinsamen Arbeit wurden Vorgaben entwickelt, wie Use-Cases, deren Beschreibung, Zustandsautomaten und Aktivitätsdiagramme in der Systemanalyse eingesetzt werden sollten um Analyseergebnisse geeignet zu dokumentieren, aber auch um eine gute Basis für den Test zu bieten. In einem weiteren gemeinsamen Anwendungsprojekt wurde der Ansatz inzwischen verifiziert. Der Artikel stellt die Schritte Systemanalyse und Ableitung von Testfällen aus den erstellten Analysedokumenten vor. Abbildung 2 gibt einen groben Überblick über das Gesamtvorgehen, dessen Schritte wir im Folgenden näher betrachten und an dem Beispiel einer Scheibenwischersteuerung verdeutlichen.

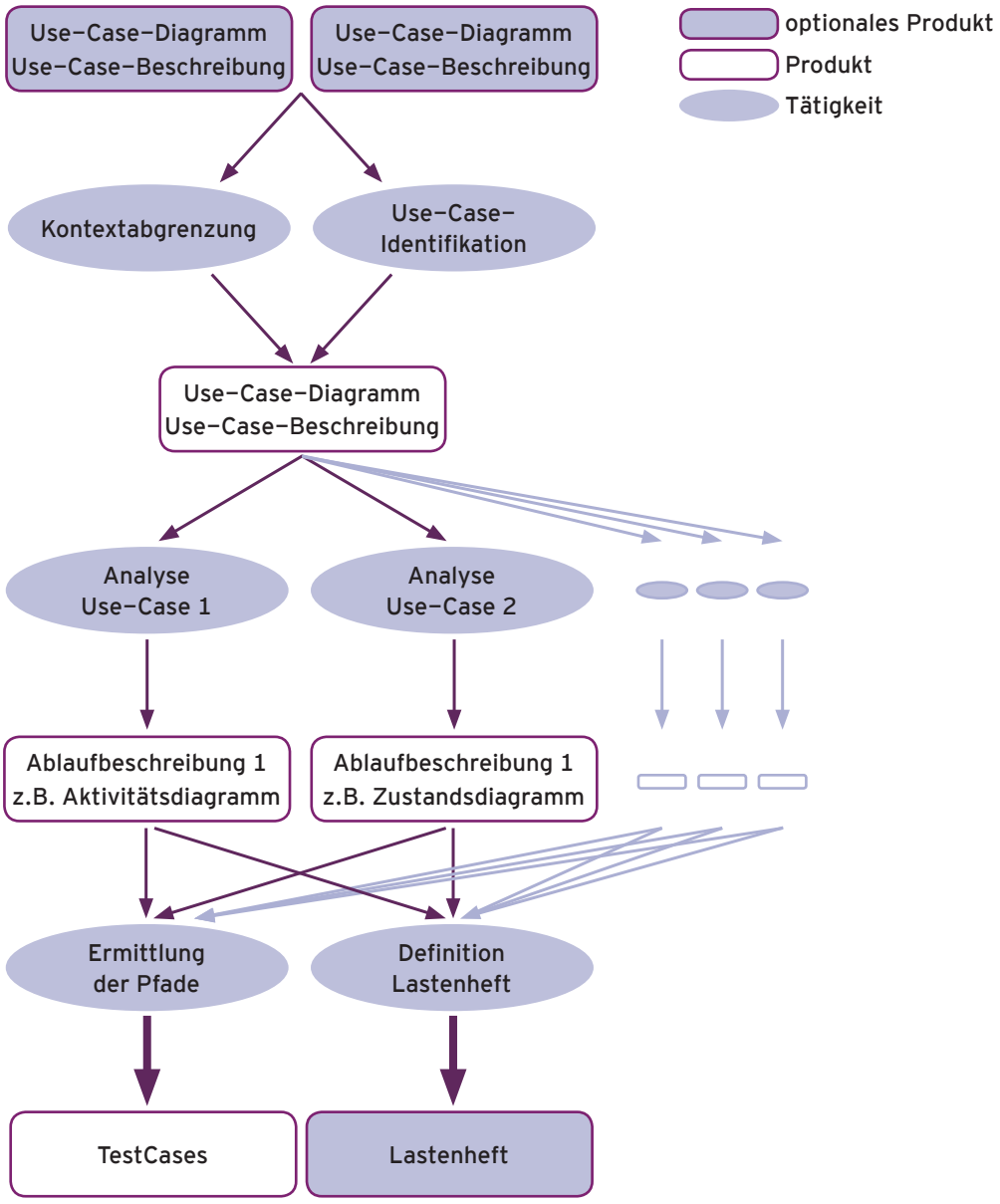

Abbildung 2: Der Weg vom Use-Case zum Test-Case im Überblick

Wir haben bereits festgestellt, dass RTE-Systeme mehr beinhalten als nur Software und meist intensiv mit ihrer Umgebung kommunizieren – sogar baulich in ihr verankert sind. Bei der Systemkonzeption muss daher explizit geklärt werden, was außerhalb und was innerhalb der Systemgrenzen liegt. Die Ergebnisse dieser Überlegung bezeichnen wir als Kontextdiagramm. Hierfür nutzen wir ein Use-Case-Diagramm, welches uns neben der Kontextabgrenzung auch noch den Dienst erweisen, unser System aus einer Black-

Box-Betrachtungsweise in handhabbare Stücke zu zerlegen. Bei der textuellen Beschreibung der Use-Cases experimentierten wir mit unterschiedlichen Beschreibungsarten und Detaillierungsstufen. Kapitel 6 stellt die Lösung vor, die unserer Meinung nach die meisten Vorteile vereint und verhindert, dass wir bei komplexen technischen Systemen sofort im Chaos der Ausnahmebedingungen versinken. Betrachtet man reaktive Systemen, so drängten sich für eine detailliertere Spezifikation des Systemverhaltens Zustandsdiagramme auf. Sie erwiesen sich als gute Basis für die Spezifikation des Systemverhaltens auf unterschiedlichen Detaillierungsstufen und als formale Basis, um automatisiert Testfälle erzeugen, reduzieren, und produzieren zu können.

2. Das Beispiel: Die Scheibenwischersteuerung

Als Beispiel, welches sich durch diesen Artikel zieht, haben wir uns für eine Scheibenwischersteuerung entschieden, die jeder von seinem PKW kennt. Das Scheibenwischersystem ist in das Kraftfahrzeug eingebettet. Bedient wird das System vom sogenannten Scheibenwischerhebel (SWH) aus. Die Funktionalität haben wir auf kontinuierliches Wischen (Pos 2), Tippwischen (Pos 3) und Intervallwischen (Pos 1) reduziert. Eine Energieversorgung ist für die richtige Spannung zuständig und der Wischersensor teilt uns mit, ob der Wischer in der Parkposition ist oder sich im Sichtfeld des Fahrers befindet. Für das reale System, welches wir im Rahmen unserer Forschungsprojekt betrachteten, ergaben sich zwölf Nachbarsysteme mit insgesamt 22 auf das System auftreffenden und 16 vom System produzierten Ereignissen. Das hier verwendete Beispiel ist natürlich deutlich kleiner und beschränkt sich auf vier Nachbarsysteme und nur wenige Ereignisse.

3. Der erste Schritt: Use-Cases

Die meisten Systeme, die wir heute erstellen oder weiterentwickeln, umfassen viele Systemprozesse und sind relativ umfangreich und komplex – zu komplex, um sie „am Stück“ zu begreifen. Deshalb müssen wir unser System systematisch in mundgerechte, verdaubare Bissen zerteilen. Dazu hat Ivar Jacobson mit seiner Idee der Use-Cases (Systemprozesse) einen hervorragenden Beitrag geleistet [Jac92], die von der UML aufgegriffen und adaptiert wurde. Reiz dieses Ansatzes ist die Außenbetrachtung des Systems – das System bleibt (zunächst) eine *Black Box*. Use-Case-Analyse bedeutet, dass wir Akteure *außerhalb* des Systems suchen und von diesen wissen wollen, was sie von unserem System erwarten. Durch diese Herangehensweise kommt man fast wie von selbst zu einer fachlichen, logischen Zerlegung des Systems, ohne über seine interne Struktur nachdenken zu müssen.

Als Jacobson Use-Cases einführte, dachte er an Softwareprozesse. Wir übertragen diese Idee auch auf die System-/Produktebene. System-Use-Cases stellen Systemprozesse quer durch Hard- und Software dar. Akteure in der Systemumgebung initiieren derartige Systemprozesse. Bei unserem Beispiel befinden wir uns in einem Systemumfeld, das durch vorhergehende Designentscheidungen entstanden ist und das für uns nicht veränderbar ist. Wir haben – wie die Kontextabgrenzung zeigen wird – Nachbarsysteme, die mit unserem System auf eine festgelegte Art und Weise kommunizieren und Erwartungen an sein Verhalten haben. Genau diese Erwartungen werden auch bei der Abnahme des Systems getestet.

3.1. Kontextabgrenzung und Use-Case-Identifikation

Die UML stellt uns zwar viele Diagrammarten zur Verfügung, enthält aber in der aktuellen Version kein Kontextdiagramm. Zahlreiche Autoren, die auf die Systemabgrenzung und Schnittstellenpräzisierung gegenüber der Umwelt viel Wert legen, simulieren dieses Diagramm durch UML-konforme Diagrammtypen.

Bei der Abgrenzung des Kontexts unterscheiden wir zwischen logischem und physikalischem Kontext. Beim logischen Kontext liegt der Fokus auf der Kommunikation mit den Nachbarsystemen, beim physikalischen Kontext auf den physikalischen Kanälen und Übertragungsmedien, die das System mit den Nachbarsystemen verbindet. Egal, welche Art der Darstellung Sie für die Kontextabgrenzung wählen: Sie müssen auf jeden Fall die Nachbarsysteme Ihres Systems identifizieren.

Die Kontextabgrenzung mittels Use-Case-Diagrammen ist der von vielen Vorgehensmodellen empfohlene Einstieg in die Systementwicklung. Mittels Use-Case-Diagrammen lässt sich der logische Kontext Ihres Systems gut abgrenzen, weswegen wir in unserem Forschungsprojekt auch damit starten.

Im Rahmen der Kontextabgrenzung schlagen wir Ihnen folgende Konventio¬nen für die Nutzung von Use-Case-Diagrammen vor, die teilweise über die UML-Standardempfehlungen hinausgehen:

- Benennen Sie Ihr System exakt, um dem Gegenstand Ihrer Entwicklung Identität zu geben.
- Zeichnen Sie nicht nur Akteure ein, d. h. nicht nur die Nachbarsysteme, die Prozesse auslösen, sondern **alle** Nachbarsysteme, mit denen Ihr System kommuniziert. Viele UML-Bücher suggerieren, nur die Akteure einzuzeichnen, die Prozesse auslösen. Bei der Verwendung von Use-Cases zur Kontextabgrenzung müssen auch Nachbarsysteme dargestellt werden, die nur Eingaben an unser System liefern oder Ergebnisse empfangen.
- Kennzeichnen Sie die echten Akteure als Initiator (Stereotyp <<initiator>>).
- Verwenden Sie Strichmännchen nur für Menschen, die mit dem System kommunizieren.
- Nutzen Sie das Klassensymbol für andere Arten von Nachbarsystemen und nutzen Sie spezifische Stereotypen, um die Arten von Akteuren zu unterscheiden.
- Finden und dokumentieren Sie alle Ereignisse aller Akteure, die auf Ihr System zukommen können. Dies ist eine wichtige Vorbedingung, um auch alle erforderlichen Testfälle ableiten zu können und das Verhalten des Systems vollständig zu spezifizieren.
- Zeichnen Sie die Systemprozesse aus der Sicht Ihrer Akteure ein. Da die Akteure bei den meisten technischen Systemen Nachbarsysteme sind, sollten Sie auch deren Sicht einnehmen. Wichtig ist, welche Erwartung ein Nachbarsystem an Ihr System hat. Die Sicht des Endnutzers ist hier oft durch Designentscheidungen in Erwartungen von Nachbarsystemen heruntergebrochen worden.
- Informationen über die Ein- und Ausgaben, die zwischen dem System und sei¬ner Umgebung ausgetauscht werden, werden im Use-Case-Diagramm normalerweise weggelassen oder formlos als Notizen an den Assoziationen notiert. Die UML bietet aber die Möglichkeiten zur genaueren Spezifikation der Assoziation zwischen Systemumgebung und System. Grundsätzlich kann die Beziehung zwischen Use-Case und Akteur mit allen näheren Angaben einer Assoziation (zum Beispiel mit Multiplizitäten oder Rollen) versehen werden – wie im Klassendiagramm.
- Verwenden Sie bei einer großen Anzahl von Ereignissen, die die Systemgrenze passieren oder bei vorhandenen Zusatzinformationen zu den Ereignissen (z.B. Parameter, physikalische Werte) eine Tabelle, in der Sie die Ereignisse mit allen Informationen aufführen. Notieren Sie dann im Use-Case-Diagramm nur einen essenziellen Namen für das Ereignis und packen Sie alle weiteren Informationen redundanzfrei in die Tabelle.
- Für die Abgrenzung der Use-Cases untereinander, also das Schneiden der Use-Cases, existieren mehr oder weniger restriktive Richtlinien (siehe [Abr02], [Oes02]), auf die wir hier nicht im Detail eingehen. Jedoch hat es sich für den Einsatz als Ausgangspunkt für das Herleiten der Test-Cases als sinnvoll herausgestellt, zwei Use-Cases dann voneinander zu trennen, wenn sie verschiedene Vor- oder Nachbedingungen besitzen.

Ein gutes Use-Case-Diagramme leistet Ihnen zwei wichtige Dienste. Es dokumentiert die Kontextabgrenzung Ihres Systems und visualisiert die gefundenen Systemprozesse.

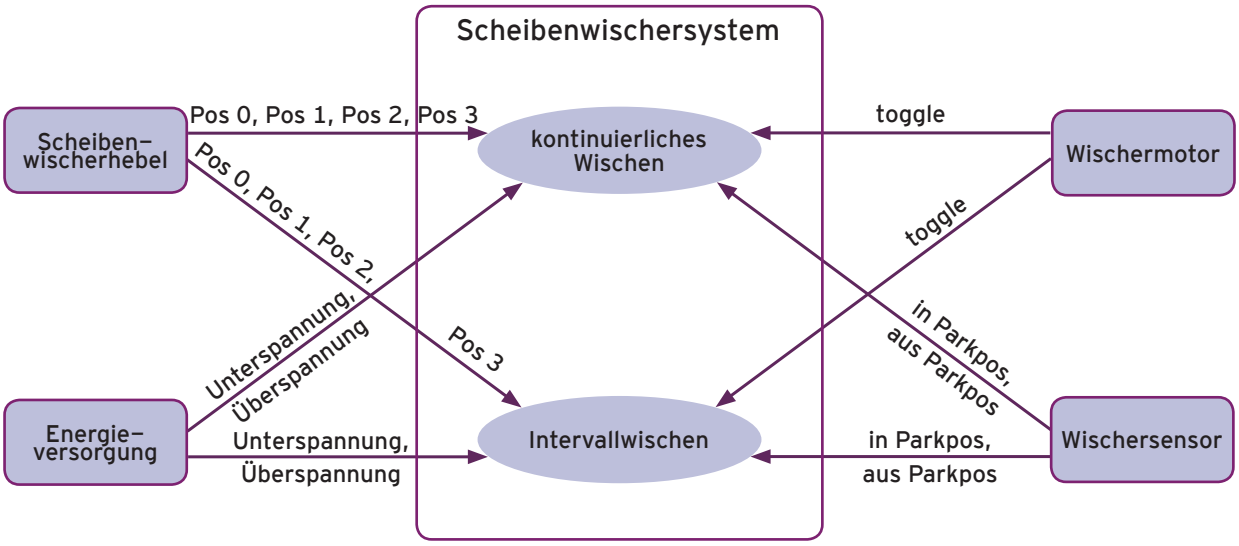

Abbildung 3: Vereinfachtes Use-Case-Diagramm des Scheibenwischersystems

3.2. Erstellung der Use-Case-Beschreibung

Jeder gefundene Use-Case stellt einen Teilprozess des Systems dar. Im folgenden Schritt werden diese Teilprozesse analysiert, wobei zunächst die essenziellen Schritte der einzelnen Use-Cases identifiziert und dokumentiert werden. Unter den essenziellen Schritten verstehen wir logische, technologieneutrale Einzelschritte, die für die Ausführung des betrachteten Prozesses unerlässlich sind und das normale Verhalten jenseits von Design- und Realisierungsentscheidungen abbilden.

Zusätzlich werden die Ausnahmefälle gesucht, allerdings ohne den Anspruch, deren Auswirkungen vollständig zu definieren. Da später aus den Use-Cases auch die Test-Cases hergeleitet werden sollen, müssen im Rahmen der Analyse alle Auslöser für Fehlersituationen erfasst werden. Eine vollständige Beschreibung aller Fehlerszenarien in textueller Form würde allerdings zu seitenlangen Beschreibungen führen, die aufwendig zu dokumentieren, redundant, und nicht wartbar wären. Deswegen konzentrieren wir uns hier auf die Auslöser und wählen eine geeignetere Notation als Prosa für die Szenarien. Die beschriebenen Use-Cases sollten folgende Punkte enthalten:

- Name
- Akteur
- Auslösendes Ereignis
- Kurzbeschreibung
- Vorbedingungen
- Essenzielle Schritte
- Trigger der Ausnahmefälle
- Nachbedingung

Diese Punkte können, je nach Projektkontext, um z.B. Auftrittshäufigkeiten, Verfügbarkeits- oder Performanceanforderungen, etc. ergänzt werden.

Für das vorliegende Beispiel wurde unter anderem der folgende Ausschnitt eines Use-Case ermittelt:

Name	kontinuierliches Wischen	
Akteuere	Scheibenwischerhebel, Energieversorgung, Wischermotor, Wischersensor	
Auslösendes Ereignis	Scheibenwischerhebel in Position 2 oder Scheibenwischerhebel in Position 3	
Kurzbeschreibung	Vom Scheibenwischerhebel 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**
	Scheibenwischerhebel -> 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.
	Scheibenwischerhebel -> Pos 3 (Tippwischen)	Falls der Wischer in Parkposition ist und der Scheibenwischerhebel 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.
	Scheibenwischerhebel -> Pos 0 (Wischen deaktivieren) oder Scheibenwischerhebel -> 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 (Energiever-sorgung -> 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).	

4. Die Formalisierung: UML-Verhaltensdiagramme

Nachdem wir nun die Use-Cases in der oben beschriebenen Weise definiert haben, besitzen wir auf einem hohen Abstraktionsniveau eine vollständige Beschreibung des Black-Box-Verhaltens unseres Systems. Diese natürlichsprachlichen Beschreibung überführen wir nun in eine formale Notation. Dieser Bearbeitungsschritt erlaubt uns eine kompakte und übersichtliche Darstellung des Verhaltens unseres Systems. Danach können wir das bisher hohe Abstraktionsniveau verlassen und zu einer realitätsnahen, d.h. detaillierten Beschreibung übergehen. Weiterhin haben wir dadurch die Chance, die Weiterbearbeitung in einer systematischen Weise durchzuführen.

Für jeden einzelnen Use-Case erstellen wir eine Prozess¬ab¬lauf¬beschreibung, die das Verhalten des jeweiligen Use-Case losgelöst von dem Rest des Systems wiederspiegelt. Als Notation schlagen wir entweder Zustandsdiagramme oder Aktivitätsdiagramme vor. Welche dieser beiden Notationen Sie verwenden sollten, hängt von mehreren Faktoren ab. Entscheidend jedoch ist hier der Typ des zu beschreibenden Prozesses.

Ablauforientierte Prozesse erkennen Sie daran, das ihre essenziellen Schritte in zeitlich logische Abfolgen gebracht werden können. Häufig erkennen Sie das bereits bei den natürlichsprachlichen Formulierungen ihrer Stakeholder. Begriffe wie „anschließend, danach, nach Beendigung, im Folgenden...“ in der Systemprozess-Beschreibung lassen auf zeitliche Folgen schließen. Können Sie die Beschreibung als kleine Geschichte (Story) erzählen, liegt meist ein Ablauf vor. Für die ablauforientierten Prozesse empfiehlt sich meist eine Beschreibung durch Aktivitätsdiagramme, da in ihnen die Abläufe sehr gut dargestellt werden können.

Reaktive Prozesse werden hingegen immer wieder durch Ereignisse beeinflusst. Das können Benutzereingaben, Signale von Nachbarsystemen, Geräten, Timern oder anderen Systemprozessen sein. Die Prozesse befinden sich in einem spezifischen Zustand und reagieren dementsprechend. Sie durchlaufen keine fest vorherbestimmte Aktionsketten. Charakteristische Beschreibungsmerkmale in Stakeholderbeschreibungen sind hierfür „ein Ereignis trifft ein, im Zustand ..., reagiert auf ..., wenn das passiert, dann ..., verweilt bis, ...“. Reaktive Systeme lassen sich unserer Erfahrung nach am geeignetsten mit Zustandsdiagrammen modellieren.

Für unser Beispiel, die Scheibenwischersteuerung, haben wir, da es sich in die Gruppe der reaktiven Systeme eingliedert, die Zustandsdiagramme zur weiteren Beschreibung gewählt. Deswegen werden wir im Weiteren ausschließlich von Zustandsdiagrammen sprechen, wobei sich die getroffenen Aussagen jederzeit auch auf die Aktivitätsdiagramme übertragen lassen.

Die Überführung der Use-Cases in Zustandsdiagramme kann leider nicht automatisch durchgeführt werden, so dass Sie hierbei gefordert sind. Je besser jedoch die Beschreibung der Use-Cases ist, je genauer Sie sich an die Vorgaben bei der Erstellung der Use-Cases gehalten haben, desto leichter wird Ihnen dieser Schritt fallen. Ein detaillierte Anleitung für diesen Schritt finden Sie in [RuH02].

Wir schlagen an dieser Stelle vor, zunächst die essenziellen Schritte aus der Use-Case-Beschreibung in ein Zustandsdiagramm zu überführen. Dieses repräsentiert das Verhalten des Prozesses vollständig, wenn auch bis hierhin nur auf einem abstrakten Niveau. Es ermöglicht Ihnen jedoch eine grobe Sicht auf die Grundfunktionalität des Prozesses. Die Detaillierung der Zustandsdiagramme durch Hinzufügen der Ausnahmefälle sehen wir als eine Verfeinerung an, die in Kapitel 6 vorgestellt wird.

Abbildung 4 zeigt ein Zustandsdiagramm, welches die essenziellen Schritte des Use-Cases Kontinuierliches Wischen abbildet.

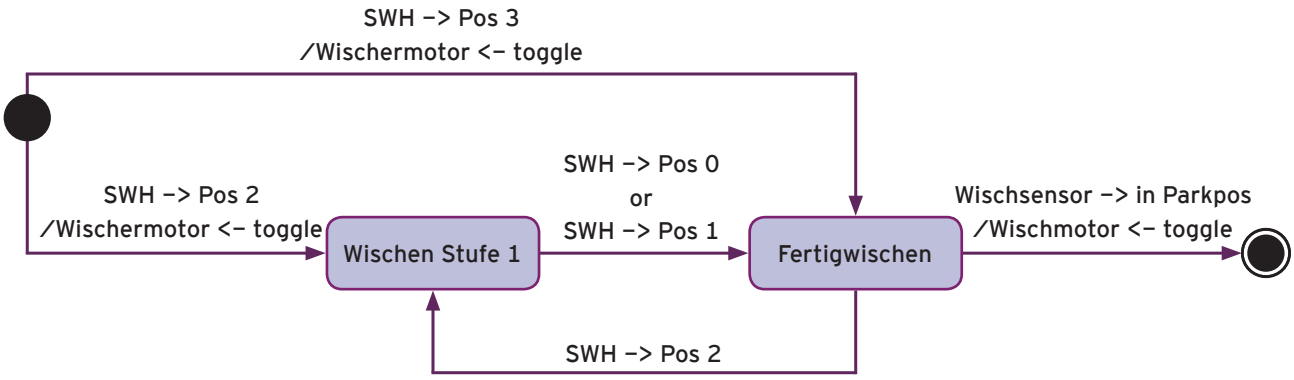

Abbildung 4: Zustandsdiagramm des kontinuierlichen Wischens

Sind die einzelnen Use-Cases Zustandsdiagramme definiert, haben wir, unabhängig von dem betrachteten Abstraktionsniveau, eine formale Grundlage geschaffen, um teilweise automatisiert Test-Cases für die

einzelnen Prozesse herzuleiten. Für das Herleiten der Test-Cases, die das Verhalten des Gesamtsystems überprüfen, benötigen wir Kombinationen der einzelnen Use-Cases bzw. der Zustandsdiagramme. Dieses Vorgehen beschreiben wir in Kapitel 6.

5. Die Generierung: Vom Pfad zum Test-Case

Nachdem für jeden gefundenen Use-Case zumindest ein Zustandsdiagramm vorliegt, besteht unsere nächste Aufgabe darin, daraus effizient Test-Cases herzuleiten. Die Kunst besteht darin, die zu testenden Abläufe aus den Beschreibungen abzuleiten.

Diese zu testenden Abläufe entsprechen dabei Pfaden in den Zustandsdiagrammen, wobei wir die eingehenden Ereignisse als Auslöser für die Übergänge ansehen. Die Anfangs- und Endpunkte dieser Pfade sind durch die Modellierung vorgegeben (der Start- und Endzustand in einem Zustandsgraphen).

Um einen vollständigen Test Ihres Systems zu ermöglichen, müssen Sie prinzipiell alle möglichen Pfade vom Start- zum Endpunkt betrachten. Diese Anzahl kann jedoch eingeschränkt werden, ohne dass Sie einen geringeren Grad an Testabdeckung in Kauf nehmen müssen (siehe Kapitel 7).

Beachten Sie bei der Ermittlung der Pfade, dass in den Diagrammen potentiell Zyklen auftreten können. Zunächst wird von der relativ einfachen Behandlung dieser Zyklen ausgegangen: Für die in unser Beispiel ermittelten Pfade nehmen wir an, dass jeder Zyklus nur einmal durchlaufen wird!

Testpfade aus dem Zustandsdiagramm „kontinuierliches Wischen“ (essenzielle Schritte), Abbildung 5			
Pfad 1	**Zustand**	**Ereignis**	**Aktionen**
Schritt 1	Initial	SWH -> Pos 2	Wischermotor <- toggle
Schritt 2	Wischen Stufe 1	SWH -> Pos 0	
Schritt 3	Fertigwischen	Wischersensor -> in Parkpos	Wischermotor <- toggle
Pfad 2	**Zustand**	**Ereignis**	**Aktionen**
Schritt 1	Initial	SWH -> Pos 2	Wischermotor <- toggle
Schritt 2	Wischen Stufe 1	SWH -> Pos 1	
Schritt 3	Fertigwischen	Wischersensor -> in Parkpos	Wischermotor <- toggle
Pfad 3	**Zustand**	**Ereignis**	**Aktionen**
Schritt 1	Initial	SWH -> Pos 3	Wischermotor <- toggle
Schritt 2	Fertigwischen	Wischersensor -> in Parkpos	Wischermotor <- toggle
Pfad 4	**Zustand**	**Ereignis**	**Aktionen**
Schritt 1	Initial	SWH -> Pos 3	Wischermotor <- toggle
Schritt 2	Fertigwischen	SWH -> Pos 2	
Schritt 3	Wischen Stufe 1	SWH -> Pos 0	
Schritt 4	Fertigwischen	Wischersensor -> in Parkpos	Wischermotor <- toggle

Abbildung 5 : Testpfade für das kontinuierliche Wischen

Für jeden aus dem Zustandsdiagramm ermittelten Pfad können Sie nun einen Test-Case definieren. Dabei richten wir uns im Wesentlichen nach der in der ANSI/IEEE Testdokumentationsspezifikation 829 gegebenen Notation. Als Beispiel in Abbildung 6 dient der erste Pfad aus der Abbildung 5. Wir haben hier nur wichtigsten Beschreibungsmerkmale dargestellt. Die Vor- und Nachbedingungen können Sie aus den entsprechenden Bedingungen des zugehörigen Use-Case ableiten.

Test-Case ID: WIWA01	**Test-Case Name:** Standardwischen Geschwindigkeitsstufe 1		**Getestetes System:** Frontscheibenwischersteuerungssystem
Zugehörige Use-Cases: Kontinuierliches Wischen			
Test Typ: Funktional	**Priorität:** 2	**Autor:** SQ	**Datum:** 14.11.2002
Vorbedingungen: Zündung mind. in Stellung 1 (Radiostellung), es wird nicht gewischt oder der Wischer befindet sich im Intervallwischen.			
Nachbedingungen: Das System kehrt in den vorherigen Zustand zurück (Aus, Intervallwischen).			

Schrittnr.	**Zustand / Aktivität**	**Ereignis**	**Akteure**	**Erwartetes Ergebnis**
1	Initial	SWH -> Pos 2 : Wischen Stufe 1	SWH	Start Wischen
2	Wischen Stufe 1	SWH -> Pos 0 : Wischen deaktivieren	SWH	
3	Fertigwischen	Wischer in Parkposition; Wischermotor <- toggle	Wischersensor	Wischermotor aus; Wischer in Parkposition

Abbildung 6: Beispielhafter Test-Case

6. Die Komplettierung: Verfeinerung und Komposition

Wie Sie leicht erkennen können, ist der Test-Case aus Abbildung 6 nicht sehr gehaltvoll. Dies liegt in den wenigen Informationen begründet, die in den essenziellen Schritten des Use-Case und damit auch in dem Zustandsdiagramm beinhaltet sind.

Deswegen führen wir eine Erweiterung des Vorgehens ein, das Ihnen eine systematische Verfeinerung der Zustandsdiagramme erlaubt und so zu realitätsnaheren Test-Cases führt.

Darüber hinaus haben wir bisher noch nicht den Zusammenhang zwischen den Use-Cases betrachtet. Da wir aber das Verhalten des Gesamtsystems betrachten wollen (und nicht einzelne Teile losgelöst voneinander) müssen wir eine Verbindung zwischen den Use-Cases und damit zwischen den beschreibenden Zustandsdiagrammen herstellen. Dies ist die zweite Dimension, in die wir unser Vorgehen erweitern.

Verfeinerung der Zustandsdiagramme:

Sehen Sie ein Problem bei der Überführung der Use-Case-Beschreibung in Zustandsdiagramme? Erscheint Ihnen dieser Schritt als sehr komplex und deswegen als schwierig? Uns auch! Unser Vorgehen erlaubt Ihnen, diesen Schritt iterativ-inkrementell durchzuführen. Sie können sich entscheiden, die Use-Cases zunächst sehr abstrakt zu beschreiben, um sie dann in einer erneuten Iteration zu präzisieren. Sie können aber auch in einem ersten Schritt die nur einen Teil der Use-Cases betrachten (z.B. die essenziellen Schritte) und diese in ein Zustandsdiagram überführen. Dieses Diagramm würden Sie dann inkrementell erweitern, in dem Sie einzelne Ausnahmefälle (Über-/ Unterspannung) in das Diagramm einbauen.

Komposition der Zustandsdiagramme:

Die getrennte Betrachtung der einzelnen Use-Cases ist im Allgemeinen nicht ausreichend, um daraus aussagekräftige und umfassende Test-Cases abzuleiten. Use-Cases, die durch include- und extend-Beziehungen miteinander verbunden sind, ermöglichen z.B. das Anstoßen eines Use-Case aus der Bearbeitung eines anderen heraus. Weiterhin kann die gleichzeitige Abarbeitung mehrerer Use-Cases von außen getriggert werden, da in der Realität mehrere Use-Cases (quasi-) parallel oder auch ein Use-Case mehrfach instanziiert werden kann.

Bisher haben wir nur die einzelnen Use-Cases getrennt voneinander betrachtet; nun möchten wir diese zusammenfügen, um dadurch das Verhalten des Gesamtsystems zu beschreiben. Diese Komposition werden wir aber nicht auf der Basis der Use-Cases durchführen. Die formalere Beschreibung der Prozessablaufbeschreibungen durch Zustands- oder Aktivitätsdiagramme erlaubt uns auch hier ein systematisches Vorgehen. Das Resultat, eine Systemablaufbeschreibung, wird wiederum durch ein Zustandsdiagramm beschrieben. Es entsteht durch ein Übereinanderlegen, eine Komposition, der einzelnen Zustandsdiagramme.

Da die Use-Cases-Beschreibungen nicht definieren, wann und in welchen Kombinationen sie instanziiert werden, müssen wir bei der Komposition der Zustandsdiagramme prinzipiell alle möglichen Kombina-

tionen von Zuständen betrachten. Dabei können wir jedoch die kombinierten Zustände streichen, die in der Realität niemals erreicht werden. Wir haben hier einen kreativen Entwicklungsschritt, der sicher nicht vollständig automatisch werden kann.

Kombination der Verfeinerung und der Komposition:

Prinzipiell kann ein Zustandsdiagramm auf einer gewünschten Abstraktionsebene auf verschiedenen Wegen ermittelt werden. Es stellt sich die Frage, ob die einzelnen, beteiligten Zustandsdiagramme zunächst verfeinert und dann kombiniert werden sollen oder ob zuerst die Komposition und dann die entstandene Systemablaufbeschreibung verfeinert werden sollte. Die frühe Komposition (d.h. der Wechsel in die mittlere Spur weit oben in Abbildung 7) hat zwar den Vorteil, dass ab diesem Punkt nur noch ein Zustandsdiagramm verfeinert werden muss, jedoch ist diese Beschreibung natürlich um einiges komplexer als die einzelnen Zustandsdiagramme, aus denen es zusammengesetzt wurde. Demgegenüber hat die späte Komposition den Vorteil, relativ einfache Beschreibungen zu verfeinern, wobei zusätzlich nur der betrachtete Ausschnitt aus dem Gesamtverhalten bei der Verfeinerung relevant ist. Jedoch wird die eigentliche Komposition komplexer, da die komplexeren Zustandsdiagramme kombiniert werden müssen. Welchen Weg Sie in Abbildung 7 einschlagen, um zu den gewünschten Beschreibungen zu kommen, hängt im Wesentlichen davon ab, ob und welche abstrakteren Zustandsdiagramme Sie zum Herleiten von Test-Cases ausnutzen möchten.

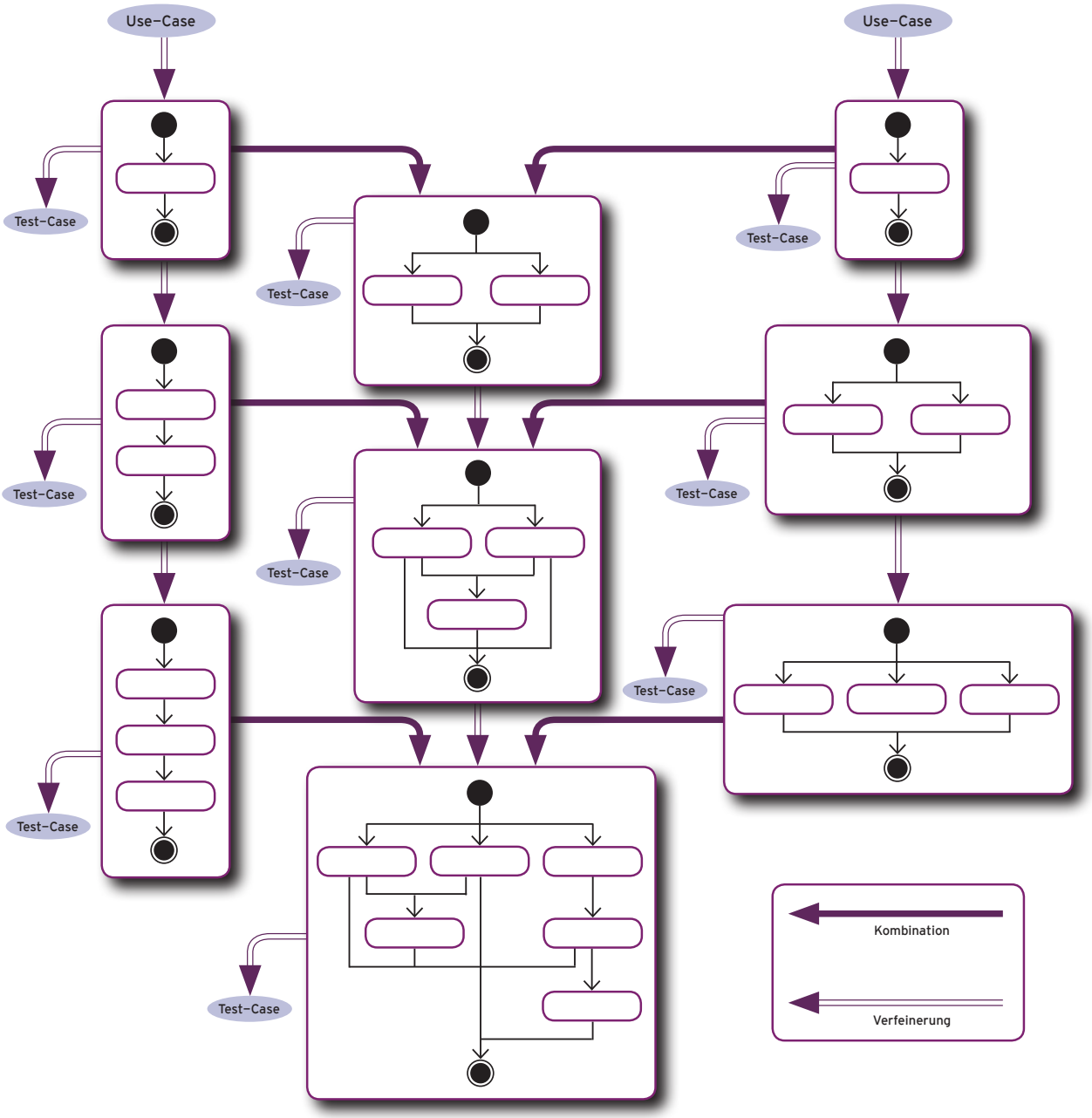

Abbildung 7: Hierarchische Test-Case Erstellung

Für unser reales System haben wir sehr detaillierte Zustandsdiagramme erstellt und aus mehreren dieser Diagramme auf verschiedenen Abstraktionsebenen Test-Cases erzeugt. Weiterhin haben wir Kompositionen auf verschiedenen Abstraktionsebenen durchgeführt. Das komplexeste Zustandsdiagramm entstand durch die Komposition von drei Zustandsdiagrammen auf niedriger Abstraktionsebene und besitzt 14 Zustände mit 57 Übergängen. Daraus konnten wir 33 unterschiedliche Testpfade mit im Schnitt jeweils sechs Schritten (bzw. Zustandsübergänge) ableiten.

7. Die Optimierung: rechnergestützte Testpfadermittlung

Das Vorgehen bis hierhin erlaubt es uns, prinzipiell beliebig komplexe Zustandsgraphen zu ermitteln, die das Systemverhalten spezifizieren, und daraus Test-Cases herzuleiten. Ein Problem, das Ihnen sicher sofort offensichtlich ist, ist die Komplexität der Modelle. Dies legt nahe, einen Teil der Arbeit vom Entwickler auf den Computer abwälzen. Insbesondere beim Erzeugen der Testfälle für einen vollständigen Test minimiert eine Rechnerunterstützung den Aufwand dramatisch. Für einen Test, der aus den Zustandsgraphen auf abstrakter Ebene resultiert, mag eine manuelle Dokumentation der Testfälle noch durchführbar sein. Für die unteren Verfeinerungsebenen muss die Herleitung der Test-Cases jedoch automatisiert werden. Zustandsdiagramme als formale Grundlage dieses Schritts erlaubt es uns zusätzlich, Einschränkungen in der Testabdeckung zuzulassen, den Tester bei dieser Tätigkeit zu unterstützen und diese Einschränkungen bei der Herleitung der Test-Cases zu berücksichtigen.

Mit den hier eingeführten Begriffen der Test-Cases und der Testpfade hängt der Aufwand zum Testen eines Systems sowohl von der Anzahl der Test-Cases als auch von der Länge der Testpfade in einem Test-Case ab. Mit der Annahme, dass bei der Abarbeitung von Test-Cases immer wieder ähnliche Aktivitäten durchgeführt werden müssen, bietet es sich an, die Zahl der Test-Cases zu verringern. Dadurch verringert sich die Anzahl der wiederkehrenden Aktivitäten und dadurch auch der Gesamtaufwand in der Testphase. Möchten Sie dennoch alle möglichen Übergänge im System testen, können die Pfade mit dieser Randbedingung definiert werden.

Bei der Testpfadermittlung können Sie nach unterschiedlichen Kriterien vorgehen. Eine mögliche Heuristik ist, zunächst die kürzesten Pfade in dem Graphen zu finden. Dies führt in Abbildung 8 zu den Pfaden (1,2,3,5), (1,2,3,4,5) und (1,2,6,7,3,5). Eine Testpfadermittlung mit dieser Heuristik führt uns zu vielen kurzen Test-Cases. Fehler bei der Durchführung derartig kurzer Test-Cases sind leicht zu lokalisieren.

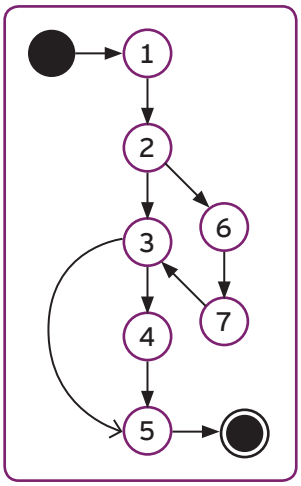

Abbildung 8: Beispielgraph

Andererseits können Sie zunächst die längsten Pfade suchen, die dabei besuchten Übergänge markieren und dann dieselbe Suche auf dem reduzierten Graphen starten. Dies führt zu den Pfaden (1,2,6,7,3,4,5) und (1,2,3,5). Wenn Sie diese Heuristik anwenden, so haben Sie die Anzahl der Testpfade bei gleichbleibendem Überdeckungsgrad reduziert. Detailliertere Erläuterungen zu Wegefindungsalgorithmen finden Sie unter [Len94].

Möchten Sie nun weniger Übergänge testen, so bedeutet dies zwar eine weniger gute Testabdeckung, aber gleichzeitig natürlich eine Ersparnis an Testaufwand. Um dies zu unterstützen, müssen die im aktuellen Kontext interessanten Pfade bestimmt werden.

Nun stellt sich die Frage, welche der Pfade als interessant bezeichnet werden. Dies wird primär von den Eingaben, die auf den jeweiligen Pfaden zu leisten sind, abhängig gemacht. Zum Beispiel kann es vorkommen, dass aufgrund der Verfügbarkeit von Testeinrichtungen nur eine Teilmenge aller Eingaben realisiert werden kann. Oder Sie entscheiden sich, den Testfokus auf eine bestimmte Eigenschaft des Systems (z.B. das Verhalten der Scheibenwischersteuerung bei Überspannung) zu legen. Ähnliches gilt für die Ausgaben des Systems, die entsprechend eingeschränkt werden können.

Grundsätzlich werden durch eine Beschränkung der Ein-/Ausgaben manche der Testpfade nicht durchführbar. Auf die aus den vollständigen Graphen ermittelten Testpfade können Sie hierbei nicht aufsetzen. Abhilfe schafft die Betrachtung der Zustandsgraphen, die entsprechend der Auswahl der interessanten Ein-/Ausgaben zunächst um die nicht erreichbaren Knoten und Übergänge reduziert werden. Danach kann dann die Pfadermittlung wie oben beschrieben starten.

Der Computer kann Sie bei der Auswahl der interessanten Ein-/Ausgaben ebenfalls unterstützen. Da die Reduktion der Graphen und die Suche nach Testpfaden im Allgemeinen schnell genug durchführbar ist, kann der Schritt der Auswahl der Ein-/Ausgaben interaktiv gestaltet werden. So erhalten Sie ein sofortiges Feedback über die Auswirkungen Ihrer Einschränkungen auf die Testabdeckung.

8. Die Zusammenfassung: zwei Partner des Erfolgs

Das hier beschriebene Vorgehen zum modellbasierten Herleiten von Testfällen bietet sowohl für die Analyse als auch für den Test des Systems Vorteile. Test-Cases lassen sich bereits in einer sehr frühen Phase systematisch aus Systembeschreibungen und nicht, wie herkömmlich, erst aus der weiteren Entwicklung, herleiten. Ein weiterer Vorteil ist die automatisierbare Herleitung der Testfälle im Laufe des Vorgehens. Durch eine systematische Kombination der Zustandsautomaten und der daraus generierten Testfälle kann der Test-Horizont erweitert werden. Hierdurch werden systemübergreifende Tests möglich. Somit kann z. B. nicht mehr nur die Scheibenwischersteuerung für sich, sondern das gesamte Fahrzeug getestet werden, sofern sein Verhalten auf diese Weise modelliert wird.

Ein wesentlicher Nebeneffekt beim Einsatz von Use-Cases ist zudem eine natürliche Ausweitung der Systembetrachtung. Eine einzelne Funktion wird nicht mehr nur isoliert betrachtet, sondern das Zusammenspiel verschiedener Funktionen rückt stärker in das Zentrum der Betrachtung. Führt man sich vor Augen, dass eine Vielzahl an Problemen genau in diesem Bereich zu finden ist, ist uns durch den Einsatz von Use-Cases damit ein mächtiges Instrument zur frühzeitigen Thematisierung dieser potentiellen Problemfelder an die Hand gegeben.

Die Ergebnisse der Analyse bilden somit eine sehr gute Ausgangsbasis für die weitere Entwicklung oder für die Beauftragung des Systems an einen Entwicklungspartner.

Literaturliste:

Abr02	Steve Adolph und Paul Bramble The Agile Software Development Series: Patterns for Effective Use Cases Addison-Wesley. 2002
High02	Klaus Grimm, Hightech Report 1/2002, Artikel Soft and Safe
Jac92	Jacobson, Ivar Object-Oriented Software Engineering – A Use Case Driven Approach. 1. Auflage, Addison Wesley Longman Verlag. 1992.
Len94	Lengauer, Thomas Combinatorial algorithms for integrated circuit layout Stuttgart: Teubner; Chichester: Wiley, 1994.
Oes01	Oestereich, Bernd Objektorientierte Softwareentwicklung – Analyse und Design mit der Unified Modeling Language 5. Auflage, Oldenburg Verlag. 2001
RuH02	Peter Hruschka, Chris Rupp Agile Softwareentwicklung für Embedded und Real-Time Systems mit der UML Hanser Verlag. 2002
UML	Unified Modeling Language (UML) v1.5, UML 2.0 (Draft) http://www.omg.org/technology/documents/formal/uml.htm

Autoren:

Chris Rupp (chris.rupp@sophist.de) liefert durch Ihre Publikationen und Vorträgen immer wieder wichtige Impulse für die Bereiche Requirements Engineering und Objektorientierung.

Erfindungen von Ihr und den SOPHISTen legten die Basis des modernen Requirements Engineering. Chris ist Geschäftsführerin der SOPHIST GROUP.

Dr. Stefan Queins (stefan.queins@sophist.de) hat nach seinem Studium im Bereich der domänenspezialisierten Entwicklung promoviert. Er ist seitdem als Berater und Trainer bei der SOPHIST GROUP tätig.

Sein Aufgabengebiet liegt in der Analyse und dem Design komplexer technischer Systeme, z.B. im Automobilbereich.