Modellbasierte Testentwicklung - Verwendung von Aktivitätsdiagrammen zur grafischen Entwicklung von Testfällen

Claus Gittinger

eXept Software AG
Zeppelinstraße 4
74357 Bönnigheim
info@exept.de

Abstrakt: Die stetig wachsende Komplexität von Hard- und Softwaresystemen, sowie steigender Funktionsumfang bei stets kürzer werdenden Produktzyklen führte im letzten Jahrzehnt zum zunehmenden Einsatz von modellbasierten Technologien in der Entwicklung. Dagegen werden für das Testen oft textbasierte Script- oder kompilierte Programmiersprachen eingesetzt. Diese stellen hohe fachliche Anforderungen an die Testentwickler – müssen sie doch neben dem Domänenwissen über das zu testende System zusätzlich über fundierte Kenntnisse der Programmiersprache(n) verfügen. In den vergangenen Jahren führte das zu einem stetig zunehmenden Aufwand für Entwicklung, Pflege und Weiterentwicklung von Tests und Testszenarien. Das hier skizzierte Testsystem „expecco“ kombiniert die Vorteile von auf UML-basierenden Testbeschreibungen mit den kurzen Turn-Around-Zeiten interaktiver scriptbasierter Systeme.

1 Einführung

Das hier beschriebene Testsystem „expecco“, dient zur Definition und Ausführung von Soft- und Hardwaretests. Diese werden als Blackbox-Tests formuliert, indem das Verhalten der Außenwelt des getesteten Systems auf einer hohen Abstraktionsebene graphisch formuliert wird [SL05]. Zur Beschreibung des Testvorgangs werden als Sprachmittel ausschließlich Aktivitätsdiagramme verwendet. Dadurch sind auch Domänenexperten, Produktplaner bzw. Ersteller von Anforderungsprofilen in der Lage, Testszenarien zu verstehen, zu modifizieren und zu erstellen – ohne dass Programmierkenntnisse vorausgesetzt werden. Da Testbeschreibungen unmittelbar interpretiert werden, kann der Testablauf im Detail angezeigt und live animiert werden. Laufende Tests können angehalten, im Einzelschrittmodus ausgeführt und sogar während des Ablaufs modifiziert werden. Eine langwierige Generierung von Testprogrammen entfällt. Über umfangreiche Bibliotheken stehen sowohl Basismechanismen als auch firmen- bzw. projektspezifische (Aktivitäts-) Bausteine zur Verfügung.

2 “Modellbasiertes Testen” - Welches Modell?

Unter „modellbasiertem Testen“ wird im Allgemeinen die Generierung bzw. Abarbeitung von Testfällen/Testprogrammen aus einem Modell verstanden. Allerdings dient hier nicht selten das aus den Anforderungen entstandene „Entwicklermodell“, welches auch in der Entwicklungsabteilung in konkrete Software umgesetzt wird, als Grundlage zur Generierung von Testfällen. Diese Vorgehensweise hat aber den Nachteil, dass keine Fehler im Modell selbst erkannt werden. Wir plädieren daher für eine eigene, separate Modellierung der Tests selbst, und nennen dieses im Folgenden “Testmodell” - im Unterschied zum Entwicklermodell. Dieses Testmodell bildet die externen Schnittstellen sowie das extern messbare Verhalten des Entwicklermodells ab.

Zum Test werden jedem Usecase bzw. Requirement ein oder mehrere Testfälle zugeordnet, welche diese validieren. Die Menge aller so beschriebenen Testfälle werden in einem Testplan zusammengefasst und stellen somit das „Testmodell“ dar [Wi00], [FS05].

3 Testfallbeschreibung als Aktivitätsdiagramme

Auf Entwicklerseite ist der Einsatz von UML (Unified Modeling Language) weit verbreitet. Es liegt daher nahe, diese Notation auch zur Beschreibung von Testvorgängen zu nutzen. Allerdings sind die Anforderungen an eine solche Notation im Testumfeld und insbesondere bei Blackbox-Tests nicht notwendigerweise mit denen der Entwickler vergleichbar. Bei der Definition von Testabläufen spielen dynamische Aspekte wie Nebenläufigkeit, Lastverteilung und Reaktion auf falsche Eingabedaten eine größere Rolle als statische wie Vererbung, Objektlayout oder interne Zustände [AD97].

Aus der Menge der in UML enthaltenen Diagrammtypen sind insbesondere Aktivitätsdiagramme in der Lage, auch dynamische Aspekte semantisch vollständig zu beschreiben, wobei natürlich vorausgesetzt wird, dass das Verhalten der Ein- und Ausgänge (Pins), sowie die einem Bearbeitungsschritt zugrunde liegende Aktivität hinreichend genau festgelegt wurden. Aktivitätsdiagramme können mit Einschränkungen [SH05] als Oberklasse von Flussdiagrammen und Petrinetzen betrachtet werden, und haben damit eine entsprechende mathematische Basis. So können Deadlock-Situationen, Abhängigkeiten, Vollständigkeit, Zyklen und andere Eigenschaften relativ einfach validiert werden. Gleichzeitig eignen sie sich auch zur Modellierung komplexer, insbesondere parallel ausgeführter Vorgänge.

Ein Aktivitätsdiagramm besteht aus einer Menge von (Verarbeitungs-) Schritten sowie Verbindungen ihrer Ein- und Ausgänge. Jedem Schritt ist eine Aktivität zugeordnet, welche durch den Datenfluss über die Verbindungen synchronisiert wird. Erst durch die Verfügbarkeit der zu einem Schritt benötigten Daten wird für diesen eine Aktivität erzeugt. Diese liest die Eingangswerte, führt die Aktion durch, und reicht schließlich seinerseits Resultate an andere Verarbeitungsschritte weiter. Die Daten können im einfachsten Fall einzelne Bits, boolesche oder numerische Messwerte oder Trigger-

informationen sein. Natürlich ist es auch möglich, komplexe Datenstrukturen, Tabellen, Dokumente, Datenbankhandles oder Objektinstanzen weiter zu reichen. Aktivitätsdiagramme werden sowohl von Programmierern als auch von Nichtprogrammierern benutzt und verstanden. Sie werden daher auch zur Dokumentation, zum Informationsaustausch und als Diskussionsbasis verwandt. Das folgende stark vereinfachte Beispiel zeigt diese Vorteile anhand eines Programmfragments zum Auslesen eines Messwertes. Das Aktivitätsdiagramm beschreibt den selben Vorgang wie der Programmtext:

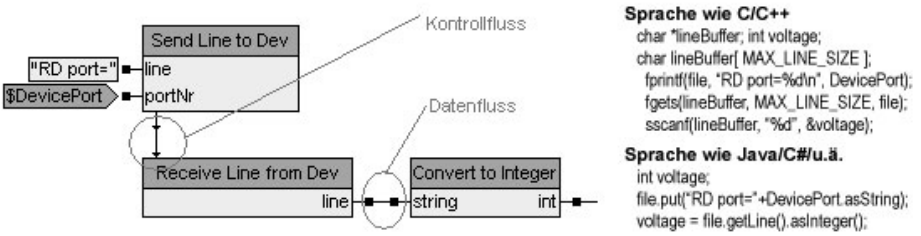

Abbildung 1: Aktivitätsdiagramm vs. Programmierung

4 Elementare und Zusammengesetzte Bausteine

Einzelne Schritte können ihrerseits als Unteraktivitätsdiagramm oder als Elementarfunktion realisiert werden. Ein Unteraktivitätsdiagramm fasst mehrere Aktionen in einem einzelnen Baustein zusammen, womit ein Äquivalent zu Unterprogrammen oder Software-ICs zur Verfügung steht. Ein elementarer Schritt wird als eingebaute Grundfunktion, Aufruf einer existierenden Bibliotheksfunktion (DLL-Call), als Remote-Procedure-Call (SOAP, COM, DCOM) oder als Skript für einem eingebauten oder externen Skriptinterpreter realisiert.

5 Testentwicklung und Ausführung

Das Erstellen von Tests ist ein kreativer und oft explorativer Prozess [Bi95]. Daher werden von Testentwicklern mehr noch als von „normalen Entwicklern“ kurze Turn-Around-Zeiten und interaktive Programmierhilfen gewünscht. Das „expecco“ Testsystem wurde daher als interpretatives System konzipiert, in dem Änderungen auch während der Testausführung möglich sind.

6 Automatische Erzeugung von Traces und Testberichten

Da sämtliche Datenflüsse zwischen den Aktivitäten optional vermerkt werden, können diese leicht visualisiert oder in einer Datei bzw. Datenbank gesichert werden. Zusätzlich

dienen sie zur Generierung eines detaillierten Testberichts als HTML-, XML- oder PDF-Dokument und zur Erzeugung von automatischen Soll-Ist-Vergleichen.

7 Import aufgezeichneter Szenarien

Verschiedene Fremdformate können mittels Import zur automatischen und halbautomatischen Erzeugung von Aktivitätsdiagrammen dienen. Besonders erwähnenswert ist hierbei der Import von Aufzeichnungen einer Web-Sitzung, wie sie z.B. das freie "Selenium"-Tool liefert. So aufgezeichnete Webaktivitäten können in wiederverwendbare Teilaktivitäten (Login, Bestellung, usw.) zerlegt, neu arrangiert und durch semantische Tests erweitert werden. Die gesammelten Daten werden automatisch archiviert und dienen zur Analyse, Ermittlung des Projektfortschrittes, zur Erstellung von Statistiken oder zur Generierung von Prüfberichten (Ausgangskontrolle). Aus der Zuordnung von Testfällen zu Projektanforderungen entsteht eine zeitnahe Übersicht des aktuellen Entwicklungs- und Fehlerstandes [BV06].

8 Erfahrungen und Zusammenfassung

Das „expecco" Testsystem wird in verschiedenen Projekten zur automatischen Testausführung, sowie zur Installation und Konfiguration von Geräten im Bereich Telekommunikation, Anlagen- und Maschinenbau, Medizintechnik sowie Automobiltechnik und Webanwendungen eingesetzt. Testfälle sind selbstdokumentierend und die zugrunde liegenden Konzepte werden schnell verstanden – dies insbesondere auch von Nichtprogrammierern. Somit wurde in allen Anwendungsfällen die Kommunikation zwischen den Domänenexperten und Testern einerseits und den Entwicklern andererseits spürbar verbessert, insbesondere da diese innerhalb weniger Stunden in der Lage waren, Testszenarien zu verstehen, zu modifizieren und zu erweitern. Gleichzeitig konnten diese den Entwicklern sofort als Feedback zur Reproduktion von Fehlern dienen, wodurch die mittlere Fehlerbehebungszeit spürbar reduziert wurde.

Literatur

[AD97] Larry Apfelbaum; John Doyle: Model Based Testing Software Quality Week Conferences, Mai 1997

[Bi95] Robert V. Binder: Object Oriented Testing: Myth and Reality Object Magazine, Mai 1995

[BV06] Andreas Bartsch; Stefan Vogel: expecco Whitepaper, 2006, www.expecco.de

[FS05] Mario Friske; Holger Schlingloff: Von Use Cases zu Test Cases, TU Braunschweig Report, 2005

[SH05] Harald Störrle; Jan Hendrik Hausmann: Towards a Formal Semantics of UML 2.0 Activities Software Engineering 2005

[Wi00] Mario Winter: Qualitätssicherung für objektorientierte Software: Anforderungsermittlung und Test gegen die Anforderungsspezifikation, 2000