GI-2.1.7. TAV AK-TOOP

Protokoll des Treffs am 15. September 2000 in Darmstadt

Anwesende

Falk Fraikin, TU Darmstadt,
Thomas Leonhardt, TU Darmstadt,
Uwe Hehn, 3Soft GmbH,
Wolfgang Henapl, TU Darmstadt (bis 12:30 Uhr),
Stefanie Ulrich, Dresdner Bank,
Mario Winter, FernUniversität Hagen sowie
einige Studierende der TU Darmstadt.

Ablauf

$10^{\underline{30}}$ - $10^{\underline{45}}$ Uhr	Begrüßung
$10^{\underline{45}}$ - $11^{\underline{45}}$ Uhr	Vortrag mit Werkzeugdemonstration Falk Fraikin: Test mit Sequenzdiagrammen.
$11^{\underline{45}}$ -12:$^{\underline{45}}$ Uhr	Vortrag Mario Winter: SCORES: Test der Anforderungsspezifikation und Systemtest.
$13^{\underline{30}}$ - $14^{\underline{30}}$ Uhr	Diskussion/Brainstorming: Test von Komponenten?
$14^{\underline{30}}$ - $15^{\underline{00}}$ Uhr	Festlegung nächster Treff und Wrap Up.

Themen

Vortrag mit Werkzeugdemonstration Falk Fraikin: Test mit Sequenzdiagrammen

In der Fachliteratur werden Sequenzdiagramme eher selten als geeignetes Mittel zur Testspezifikation genannt. Betrachtet man jedoch das Testen im Kontext eines UML-basierten Entwicklungsprozesses und legt Wert auf möglichst frühes, entwicklungsbegleitendes Testen, so verfügen Sequenzdiagramme über erhebliches Potential. Abgesehen von Ihrer weiten Verbreitung sind hier im wesentlichen 3 Vorteile zu nennen:

1. Sie ermöglichen eine leicht verständliche, effektive Schnittstellenspezifikation
2. Abstraktion/Verfeinerung wird auf intuitive Weise unterstützt (Abb. 1)
3. Darstellung aller für eine Komponente relevanten Interaktionen (Projektion) (Abb. 2)

Insbesondere 3. legt nahe, dass es unter bestimmten Voraussetzungen mittels Sequenzdiagrammen möglich ist, Unit Tests so parallel zur Entwicklung durchzuführen, dass der Integrationstest für die getesteten Einheiten entfallen kann. Das bedeutet, dass der Test- und der Entwicklungsprozess für diese Komponenten gleichzeitig abgeschlossen werden kann.

Am Fachgebiet Praktische Informatik der TU Darmstadt wird derzeit SeDiTeC (Sequence Diagram Test Center) - ein entsprechendes Werkzeug für die Programmiersprache Java - entwickelt. Es benötigt zum einen Daten über Klassen, Methoden und Sequenzdiagramme, die zu diesem Zweck aus einem CASE-Tool (momentan besteht eine Schnittstelle zu Together) extrahiert werden. Zum anderen werden die eigentlichen Testdaten (Parameter und Rückgabewerte der Methoden in den Sequenzdiagrammen) benötigt. Derzeit erfolgt dies durch XML-Dateien,

an einer grafischen Eingabemöglichkeit wird gearbeitet.

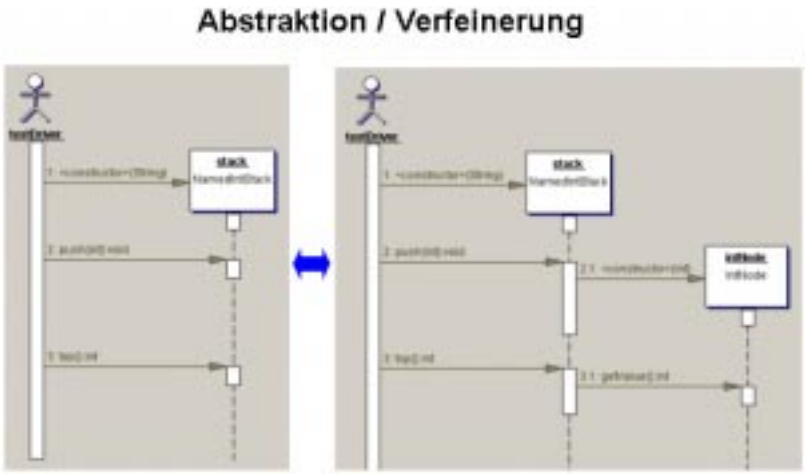

Abb. 1: Abstraktion und Verfeinerung von Sequenzdiagrammen

Diese Testdaten werden von SeDiTeC zur Ausführung und zum Testen der implementierten Klassen gemäß den spezifizierten Sequenzdiagrammen verwendet. Die Ergebnisse dieser Tests werden, ggf. mit den gefundenen Abweichungen, in HTML dargestellt. Um ein möglichst frühes Testen zu unterstützen, ist SeDiTeC in der Lage, für Klassen, die zwar in den Sequenzdiagrammen spezifiziert aber noch nicht implementiert sind, voll funktionsfähige (im Sinne der erfolgten Spezifikation) Teststümpfe zu generieren. Dies erlaubt es, eine Komponente/Klasse vollkommen unabhängig vom übrigen System zu entwickeln und zu testen.

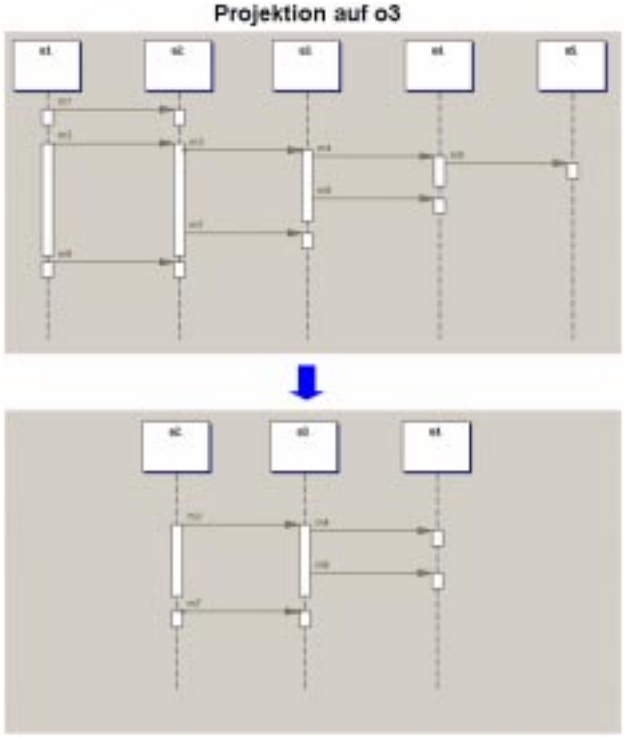

Abb. 2: Projektion auf Objekt-Teilmengen

Literatur: Falk Fraikin, Thomas Leonhardt: Top-Down Testen auf der Basis von Sequenzdiagrammen, Diplomarbeit, TU Darmstadt, 1999

Vortrag Mario Winter: Scores: Test der Anforderungsspezifikation und Systemtest

Zur Präzisierung objektorientierter Anforderungsspezifikationen wird SCORES (Systematic Coupling of Requirements Specifications) eingeführt. SCORES überbrückt die Abstraktionslücke zwischen Anwendungsfällen und dem Klassenmodell durch angepasste Aktivitätsdiagramme, welche die Modellierung von Abläufen bzw. „Kontrollfluss“ im Anwendungsfall-Konzept gestatten. Das Konzept der *Makroaktion* setzt die include-, extend- und Generalisierungsbeziehungen zwischen Anwendungsfällen auf die entsprechenden Aktivitätsdiagramme fort und ermöglicht somit erstmals die präzise Semantikdefinition dieser Beziehungen.

Die Anforderungsermittlung mit SCORES erfasst die Kontextuelle-, Interaktions- und Systeminterne Information und berücksichtigt die Qualitätssicherung bereits in den frühen Phasen der Entwicklung objektorientierter Software, da sie den nahtlosen, verfolgbaren *Übergang von den umgebenden Geschäftsprozessen über Anwendungsfälle zum (Domänen-) Klassenmodell* ermöglicht. Hierauf aufbauend unterstützt SCORES sowohl die Verifikation des Klassenmodells gegen das Anwendungsfallmodell und umgekehrt als auch die Validierung der Anwendungsfälle und wichtiger Teile des Klassenmodells.

Die während der Qualitätssicherung bei der Anforderungsermittlung investierte Arbeit zahlt sich in Form unmittelbar wiederverwendbarer Testfälle für den Test des Anwendungssystems gegen die Anforderungsspezifikation aus. Semantik und Granularität des Ansatzes ermöglichen es dabei, im sogenannten SCORES *grey-box Test* ohne größeren Mehraufwand vertraglich geforderte Werte für strukturelle Testkriterien zu erreichen. Die neu eingeführten Konzepte von SCORES erlauben die Formulierung operationaler Testendekriterien, wodurch man erstmalig quantitative Maße für die Steuerung und Kontrolle sowohl des Validierungs- und Verifikationsprozesses als auch des System- und Abnahmetestprozesses erhält.

SCORES wurde innerhalb einer Projektgruppe an der TU Darmstadt eingesetzt, wobei folgende Fragestellungen aufgetreten sind:

- Frage: Wie modelliert man Aktivitäten, die alleine vom System ohne Beteiligung von Akteuren ausgeführt werden?
  Antwort: Durch Interaktionen, denen kein Akteur zugeordnet ist. Auf feingranularer Ebene durch Sequenzdiagramme in Szenarien.
- Frage: Wozu genau ist der „Klassenbereich“ eines Anwendungsfalls/einer Aktion da?
  Antwort: Der „Klassenbereich“ beinhaltet alle Klassen, von denen Instanzen bei der „Ausführung“ des Anwendungsfalls/der Aktion eine Rolle spielen, d.h. die erzeugt, modifiziert, abgefragt oder gelöscht werden. Nur die Klassen im Klassenbereich dürfen in den textuellen Spezifikationen (Vor- und Nachbedingungen des Anwendungsfalls/der Aktion) vorkommen. Konkrete Objektkonstellationen werden dann für die Szenarien und die Ausführung von Wurzeloperationen in Szenarien angegeben.
- Frage: Dürfen Makroaktionen auch auf „partielle“ Anwendungsfälle (ohne echten Fortschritt für einen Akteur) verweisen?
  Antwort: Ja, wenn jeder „aufgerufene“ Anwendungsfall aus einer zusammenhängenden Folge von Aktionen mit klar definiertem Anfangspunkt bzw. Endpunkten besteht.
- Frage: Wie werden „Daten-“Abhängigkeiten zwischen Anwendungsfällen berücksichtigt (z.B. über gemeinsam benutzte Instanzen etc.)?
  Antwort: Einerseits durch nicht-leere Durchschnitte der Klassenbereiche (notwendige Bedingung), andererseits durch „künstliche“ Ober-Anwendungsfälle bzw. deren Aktivitätsdiagramme. Diese beinhalten nur Makroaktionen, welche die abhängigen Anwendungsfälle referenzieren, und deren Reihenfolge die erlaubten Reihenfolgen der abhängigen Anwendungsfälle.

Literatur: M. Winter: Qualitätssicherung für objektorientierte Software - Anforderungsermittlung und Test gegen die Anforderungsspezifiktion, Dissertation, Universität Hagen, Sept. 1999

H.-W. Six, G. Kösters, M. Winter: Validation and Verification of Use Cases and Class Models, Proc. 6th REFSQ, Stockholm, 4-6 Juni 2000

Test komponentenbasierter Software?

Im Brainstorming wurden folgende Punkte erarbeitet, die während des nächsten Treffens weiter diskutiert werden:

Was? Java Enterprise Beans, ActiveX, DOM, Bibliotheken, OCX, VisualBasic, ...

Eine Definition:

> A software component is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third parties. [Szyperski1998]

Merkmale? Black-Box Verwendung, keine Generalisierung, Schnittstellenspezifikation, Introspektion, Umgebung, Konfigurierbarkeit, Zusammenspiel, Trennung von Entwicklersicht und Benutzersicht, ...

Bzgl. der Trennung von Entwicklersicht und Benutzersicht wurden die Punkte

> „Die Komponente soll das tun, was sie soll“ (Spezifikation) und
> „Die Komponente soll das tun, was der Benutzer will“ (Anforderung)

gegenübergestellt.

Als mögliche Vorgehensweise beim Test komponentenbasierter Programme wurde diskutiert:

1. Eigene Anforderungen auf die Spezifikationen der Komponenten abbilden.
2. Die benötigten Operationen der Komponenten einzeln gegen die eigenen Anforderungen testen.
3. Die Operationsfolgen (woher?) testen.
4. Die Interoperabilität der verwendeten Komponenten testen.

Literatur:

[Szyperski1998] Clemens Szyperski: Component Software, acm Press, Addison Wesley, Reading, Mass. 1998

Nächstes Treffen des Arbeitskreises

Das nächste Treffen des AK-TOOP findet anlässlich des 16. Treffens der Fachgruppe vom 15.-16. Februar 2001 in der Nordakademie in Elmshorn bei Hamburg. Bitte per eMail bei M. Winter anmelden (Mario.Winter@FernUni-Hagen.de, ggf. inkl. weiteren Themenvorschlägen).