VU Softwarequalitätssicherung – Beispiel 3 CashMachine

Angabe

Aufgabenstellung – Überblick

In Beispiel a soll zu einer vorhandenen Spezifikation, unter Verwendung des Interfaces, ein Zustandsdiagramm gezeichnet werden. Kenntnisse der Implementierung des Interfaces sind dazu nicht erforderlich. Danach gilt es, in Beispiel b, alle notwendigen Äquivalenzklassen zu einer vorgegebenen Methode zu finden. Beispiel c befasst sich mit der Erstellung von Testfällen. Spezifikation, Äquivalenzklassen, Interface und Zustandsdiagramm sind in diesem Fall bekannt, Kenntnisse der Implementierung des Interfaces sind auch hier nicht erforderlich. Die Ergebnisse sollen in einem Testbericht dokumentiert werden.
Für die folgenden Beispiele d, und e wird die Implementierung benötigt. Zu Beginn, in Beispiel d, soll die Testabdeckung der Implementierung des Bankomaten, mittels des Code Abdeckung Tools CodeCover, ermittelt werden. Beispiel e befasst sich mit der Vervollständigung des Testberichts um eine Fehlerstatistik.
Am Ende, in Beispiel f, soll eine Implementierung des AccountService Interface mittels TDD erstellt werden.

Das Aufgabenpaket (.zip Archiv) besteht aus:

- Spezifikation (pdf): enthält die Anforderungen und das Interface den Bankomaten betreffend.
- Angabe (pdf, diese Datei).
- Abgabedateien: Testbericht. Leere Tabellen für Black-Box Tests, eine Fehlerstatistik sowie eine leere Tabelle für das TDD Beispiel stehen zur Verfügung. Die Ergebnisse der Aufgaben sind direkt in Ihrer persönlichen Kopie des Testberichts einzufügen und elektronisch abzugeben.
- Eclipse Java Projekt (sqs-bsp3 Ordner)
  - Interfaces CashMachine, AccountService und MachineConnector.
  - Implementierung der CashMachine.
  - Vorbereitete Abgabedateien: SimpleAccountService.java, SimpleAccountServiceTest.java, CashMachineTest.java

Allgemeine Hinweise

- Dieses Beispiel ist eine Einzelaufgabe.
- Lösen Sie die Aufgaben anhand der Angabe.
- Füllen Sie den Testbericht elektronisch aus.
- Laden Sie ihr Abgabearchiv im TUWEL rechtzeitig hoch (Deadline 29.05.2010 23:55 Uhr).
- Bitte beachten Sie, dass später bzw. per Mail eingelangte Abgaben als nicht abgegeben gewertet werden.
- Für die Abgabe exportieren Sie das Java Projekt (File → Export → General → Archive File) als ZIP-Archiv. Das Zip-Archiv muss keinen bestimmten Dateinamen aufweisen. Bitte schließen Sie den Ordner „bin“ vom Export aus. Im Projektordner sollten folgende Dateien zu finden sein:
  - Ordner doc: enthält Testbericht.doc/Testbericht.odt, Zustandsdiagramm (State.png) und exportierte CodeCover Session (Coverage.xml)
  - Ordner lib: die von Ihnen benoetigten Bibliotheken
  - Ordner src: den kompletten Projekt Sourcecode mit den von Ihnen modifizierten Dateien:
    - SimpleAccountService.java
    - SimpleAccountServiceTest.java
    - CashMachineTest.java

        - FakeMachineConnector.java
    - classpath und .project im Projektordner
- Sie können das Beispiel auch in einer anderen Entwicklungsumgebung und/oder mit einem anderen Coverage Tool ausarbeiten. Bitte beachten Sie aber, dass wir hierfür keinen Support anbieten können. Wir empfehlen Ihnen daher das Beispiel mit Eclipse zu erarbeiten.
- Vorgehensweise bei Fragen zum Beispiel:
    - Posten Sie zuerst im TUWEL Forum.
    - Dann wenden Sie sich bitte an einen Tutor.
    - Bei allfälligen schwerwiegenden Problemen, die durch die obigen Schritte nicht gelöst werden konnten, wenden Sie sich an die Übungsleitung (barbara.schuhmacher@qse.ifs.tuwien.ac.at).

3a) Zustandsdiagramm zeichnen (3 Punkte)

Zeichnen Sie von dem vorhandenen System ein Zustandsdiagramm. Das Diagramm bezieht sich auf die **Blackboxebene**, d.h. auf das Dokument Spezifikation.pdf und das Interface (CashMachine.java). Die vorhandene Java Implementierung (CashMachineImpl.java) sollte beim Zeichnen eines Zustandsdiagramms vernachlässigt werden.

- Spezifikation vollständig durchlesen
- Zustandsdiagramm auf **Blackboxebene** erstellen

3b) Äquivalenzklassen bestimmen (2 Punkte)

Finden Sie für die Methode withdraw() alle gültigen und ungültigen Äquivalenzklassen und dokumentieren Sie diese im Testbericht.

- Interface (CashMachine.java) bzw. Spezifikation vollständig durchlesen
- Äquivalenzklassen bestimmen (Äquivalenzklassenzerlegung)

3c) Black-Box Tests implementieren (8 Punkte)

Implementieren Sie 10 Black-Box Tests. Dazu ist das Eclipse Projekt mit der vorhandenen JUnit Testklasse (CashMachineTest.java) vorgesehen. Für die Aufzeichnung dieser Testfälle stehen Ihnen in dem Dokument „Testbericht“ leere Tabellen zur Verfügung. Darin sollen die erwarteten und tatsächlichen Ergebnisse der Tests dokumentiert werden, da die Implementierung nicht hundertprozentig der Spezifikation entspricht.

Geben Sie den Testmethoden aussagekräftige Methodennamen. Schreiben Sie keine Tests, die nur die get*FillLevel() Methoden testen.

Schreiben Sie 7 eigene Tests, die bestimmte Teile der Spezifikation bzw. des Interfaces testen. Des Weiteren, schreiben Sie 3 Tests um die weiter unten vorgegebenen Testfälle zu testen.

Falls nicht anders angegeben, verwenden sie Mocking um Aufrufe des AccountService oder des MachineConnector zu verifizieren. Im Eclipse Projekt wurde bereits Mockito als Mocking Framework eingebunden. Mehr Informationen über Mockito finden Sie unter http://mockito.org bzw. im Javadoc http://mockito.googlecode.com/svn/branches/1.8.0/javadoc/org/mockito/Mockito.html

- Testen Sie, dass der Bankomat das Geldausgabefach öffnet (CashMachine ruft MachineConnector.openMoneyTray() auf), wenn bei einem Abhebevorgang (dispenseAmount()) genug Banknoten zur Verfügung stehen.
- Testen Sie, dass der Bankomat für einen Betrag von 660 die richtigen Banknoten nach den in der Spezifikation definierten Regeln ausgibt. Da nicht spezifiziert ist ob ein Methodenaufruf auf MachineConnector.dispenseBill() mit einem bestimmten Banknotentyp nur 1x pro Abhebevorgang passiert, scheint Ihnen Mocking nicht die richtige Variante zum Testen zu sein. Schreiben Sie daher für Ihren Test eine Fake-Implementierung des MachineConnector Interfaces (FakeMachineConnector.java) um die Anzahl der Banknoten überprüfen zu können.
- Testen Sie, dass nach genau 3 Versuchen einen falschen Pin einzugeben, AccountService.banAccount() mit der richtigen Kontonummer aufgerufen wird. Weiters testen Sie, dass in zeitlich richtiger Reihenfolge zuvor zuerst AccountService.isValidAccountId() und 3x AccountService.validatePin() aufgerufen wird.

3d) Code Abdeckung der Black-Box Tests ermitteln (3 Punkte)

Ermitteln Sie die Abdeckung, die die Implementierung durch die Black-Box Tests hat. Verwenden Sie dazu das Coverage Tool CodeCover (http://codecover.org).

- CodeCover Eclipse Plugin installieren. Für Eclipse 3.3 beschreibt http://codecover.org/documentation/install.html den Installationsvorgang, für Eclipse 3.4+ weicht dieser leicht ab.
- Abdeckungsanalyse durchführen. Ein HOWTO findet man unter http://codecover.org/documentation/tutorials/how_to_complete.html. Dabei soll die Abdeckung der Klasse CashMachineImpl.java überprüft werden.
- Das Abdeckungs-Ergebnis exportieren (File → Export, CodeCover → Coverage Result Export). Verwenden Sie als Exporttyp *CodeCover Test Session Container*.

Bekommen Sie nach der Durchführung der Abdeckungsanalyse folgende Fehlermeldung, „*no coverage log file found after termination of project bsp3_ws09, either no files are selected for instrumenation or no code was covered at all*“, dann haben Sie vergessen die Abdeckungsanalyse für die Klasse CashMachineImpl.java zu aktivieren.

Sie können auch ein anderes Coverage Tool verwenden. In diesem Fall muss Ihre Abgabe folgende Informationen durch Screenshots und/oder Text-Ausgabe beinhalten:

- Toolname
- Prozentuelle Abdeckung der Methoden in CashMachineImpl.java
- Code Highlighting der Abdeckung in CashMachineImpl.java (z.B. als HTML oder Grafik)

3e) Testbericht vervollständigen (4 Punkte)

Der Testbericht sollte bereits die Ergebnisse aus 3c und 3e dokumentieren. Falls dies nicht der Fall ist, vervollständigen Sie diese zwei Tabellen bevor Sie die Fehlerstatistik eintragen.

Am Ende des Testberichts befindet sich eine Tabelle für die Fehlerstatistik. Hier soll die Anzahl der Tests, unterteilt in Normal-, Sonder- und Fehlerfall, aufgelistet werden um einen Überblick zu verschaffen. Nach der Erstellung der Tabelle sollen die gesammelten Ergebnisse kurz textuell kommentiert werden um das Gesamtsystem zu beschreiben.

- Vervollständigen Sie die Fehlerstatistik
- Kommentieren Sie die gesammelten Ergebnisse:

- Begutachten Sie die Abdeckung der Methoden setAccountId(), setPin() und withdraw() und wählen Sie die Methode mit der geringsten Testabdeckung. Beantworten Sie zu dieser Methode folgende Fragen:
  - Wieviele Verzweigungen (Branches) werden noch nicht abgedeckt?
  - Wieviele Tests würden Sie benötigen um eine 100% Statement bzw. Branch Abdeckung zu erreichen?
  - Währen diese zusaetzlichen Tests um 100% Abdeckung zu erzielen Black-Box oder White-Box Tests? Warum?
  - Ist es sinnvoll eine 100% Code Abdeckung anzustreben?
- Nach welchen Kriterien haben Sie Ihre Testfälle gewählt? Z.B. Requirement-based testing, Risk-based testing, ...
- Sind alle Ihre implementieren Tests Unit-Tests? Was für Kriterien haben Unit-Tests?

3f) TDD (10 Punkte)

Das Interface AccountService wurde in Beispiel 3c gemockt verwendet, um Testfälle für die CashMachine zu schreiben. Mittels Test-Driven Development soll nun eine Referenz-Implementierung (SimpleAccountService.java) dieses Interfaces entstehen.

Schreiben Sie anhand der Interface-Dokumentation mindestens 15 Tests. Es ist nicht zwingend notwendig zuerst alle Tests zu implementieren, jedoch empfehlen wir Ihnen mit mindestens 2 Tests anzufangen und abwechselnd Test und Implementierung zu erstellen. Es soll sowohl „normales“ Verhalten, als auch das Fehlerverhalten getestet werden. In der Testklasse (SimpleAccountServiceTest.java) wurde bereits ein einfaches Test-Fixture mit 2 Accounts vorbereitet. Speichern Sie die Accounts intern in einer beliebigen Datenstruktur ab. Beachten Sie, dass ein Abhebevorgang nach aussen hin sichtbar sein sollte, d.h. eine Abhebung (withdraw) sollte bei einem späteren Abruf des Accounts (validatePin) in einem reduzierten Kontostand resultieren.

Tragen Sie die Reihenfolge der Umsetzung im Testbericht ein. Erwähnen Sie Probleme oder erstellte Hilfsmethoden. Sollte sich während der Implementierung herausstellen, dass Tests fehlen, fügen Sie diese hinzu und dokumentieren Sie diese.