VU Softwarequalitätssicherung – Beispiel 4

Angabe

Aufgabenstellung – Überblick

Dieses Dokument enthält eine Beschreibung der Aufgabenstellung und dient als Leitfaden zur Lösung der Beispiele. Die zwei Unterbereiche der Aufgabe umfassen, in jeweils von einander unabhängigen Packages, Tool-gestützte Fehlersuche mit FindBugs und Refactoring (vereinfachte Implementierung eines Pizza-Dienstes).

Das Aufgabenpaket (.zip Archiv) besteht aus:

- Angabe (pdf, diese Datei)
- Bericht-Vorlage im „doc“-Ordner des Java Projekts
- Eclipse Java Projekt
  - „junit-src“ Ordner: FindBugs Beispiel, Quellcode des JUnit Frameworks
  - „src“ Ordner: Refactoring Beispiel, Quellcode des Pizza-Dienstes

Allgemeine Hinweise

- Dieses Beispiel ist eine Einzelaufgabe.
- Lösen Sie die Aufgaben anhand der Angabe.
- Laden Sie ihr Abgabearchiv im TUWEL rechtzeitig hoch (Deadline 15.6.2010 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 Ordner bsp4_ss10 sollten folgende Dateien zu finden sein:
  - Ordner doc: enthält Bericht.pdf
  - Ordner junit-src: der zu analysierende Quellcode mit den von Ihnen verbesserten Codeteilen
  - Ordner src: ihre überarbeitete Version des Pizza-Dienstes samt der zugehörigenTestklasse(n)
  - Ordner lib: die von Ihnen benötigten Bibliotheken
  - .classpath und .project im Projektordner
- Sie können das Beispiel auch in einer anderen Entwicklungsumgebung 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).

4a) FindBugs (7 Punkte)

FindBugs wurde in der Vorlesung bereits kurz angesprochen. Es führt eine statische Code Analyse auf den ByteCode von Javaklassen durch und versucht Code-Patterns zu finden, die mit hoher Wahrscheinlichkeit Fehler beinhalten. In dieser Aufgabe werden Sie den Quellcode des Junit-Frameworks analysieren (Ordner „junit-src“). Dokumentieren Sie die gefundenen Fehler in der Vorlage Bericht.odt bzw. Bericht.doc.

Lösen des Beispiels:

- FindBugs Eclipse Plugin installieren (http://findbugs.sourceforge.net/manual/eclipse.html) und auf das Projekt ausführen.
- In den FindBugs-View wechseln und das Projekt analysieren.
- 4 verschiedene Bugs, die zu Recht von FindBugs bemängelt wurden finden und in der Vorlage (Bericht.doc bzw. Bericht.odt) dokumentieren. In eigenen Worten begründen, warum der bemängelte Bug auch wirklich ein Bug ist.
- Die 4 von Ihnen gewählten Bugs im Quellcode ausbessern.
- Sie müssen das Programm nicht starten und testen, sobald das Projekt wieder kompiliert (Eclipse zeigt keine Fehlermeldungen an) und FindBugs den Fehler nicht mehr anzeigt, haben Sie einen Bug erfolgreich gefunden, dokumentiert und ausgebessert.
- Können Sie auch 2 Bugs finden die, ihrer Meinung nach, zu unrecht von FindBugs bemängelt wurden oder werden alle Bugs zu recht bemängelt? Dokumentieren Sie ihr Ergebnis.

Anm.:

- Falls ausgebesserte Fehler weiter als Fehler angezeigt werden, analysieren Sie den Quellcode noch einmal mit FindBugs. Danach sollte die Fehlermeldung nicht mehr aufscheinen.
- Von FindBugs kopierte Erklärungen der Fehler werden nicht gewertet!

4b) Refactoring Patterns (13 Punkte)

Vorbedingung für Qualitätssicherung (etwa systematische Tests) sind verständliche und testbare Anforderungen. In dieser Übungsaufgabe sollen Sie auf einem Beispielprojekt konkrete Refactoring Patterns anwenden, um die Qualität und Testbarkeit zu erhöhen. Außerdem soll aufgezeichnet werden, in welcher Reihenfolge die Patterns angewandt wurden.

Im Ordner „src“ finden Sie den Pizza-Dienst. Dieser besteht aus einer minimalen Swing-GUI (OrderedWindow), 3 Modelklassen (Customer, Pizza, OrderedPizza) und einer Startklasse (Runner). Führen Sie das Projekt aus und schauen Sie in den Code um ein Verständnis der Funktionalität zu bekommen.

Identifizieren Sie zuerst „Bad Smells“ in den Modelklassen und refactoren Sie diese. Dafür existieren bereits ein paar Tests, die gegen die Swing-GUI programmiert wurden, um die Refactorings abzusichern. Erst danach verbessern Sie die Swing-GUI, indem Sie vom GUI unabhängige Logik in eine eigene Klasse refactoren.

Wenden Sie die in der VO beschriebenen bzw. die in der unten stehenden Tabelle aufgeführten Refactoring Patterns an, um die Qualität und Testbarkeit zu erhöhen. Unter http://sourcemaking.com/refactoring können Sie weitere Details zu den einzelnen Patterns nachlesen. Eclipse selbst bietet bereits Support für einige Refactoring-Patterns. Diese sind im Kontextmenü des Editors unter dem Punkt Refactoring (auch über Shift+Alt+T erreichbar) zu finden.

Refactoring Patterns	
Encapsulate Field	Extract Method
Move Method	Replace Temp with Query
Replace Conditional with Polymorphism	Extract Class
Replace Type Code with State/Strategy	

Analysieren Sie jedes Refactoring Pattern und entscheiden Sie, ob und wie es brauchbar ist um den bestehenden Code zu verbessern. Wenn möglich schreiben Sie zuerst/hinterher einen Test, der die Funktionalität ihres Refactorings überprüft und wenden Sie danach das jeweilige Pattern an. Dabei soll der Code um folgende Kriterien verbessert werden:

Model-Klassen

- Die Datenkapselung in der Pizzaklasse soll verbessert werden.
- Im aktuellen Code muss bei jeder neuen Pizzagröße (SIZE_* Konstanten) die Customer-Klasse angepasst werden. Dies soll im Sinne von Polymorphismus oder Komposition verbessert werden. Beachten Sie auch, dass der Preis abhängig von der Pizzagröße ist, jedoch zurzeit in der Startklasse berechnet wird.
- Der Preis einer OrderedPizza wird zurzeit im Customer berechnet, obwohl keine Daten des Customers verwendet werden. Die Klasse OrderedPizza soll ihren Preis selbst berechnen können (calculatePrice()).
- Die Customer-Klasse soll Methoden bieten um die Gesamtkosten aller georderten Pizzen und die Pizza-Points abfragen zu können (getTotalPrice(), getPizzaPoints()).
- Im Sinne des OOP Prinzip „Single Responsibility“ soll die Erstellung der Order-Summary in eine eigene Klasse ausgelagert werden.

UI-Klassen

Zukünftig soll das Projekt auch durch eine Web-UI bzw. eine Konsolenschnittstelle erweitert werden. In der derzeitigen Implementierung steckt die komplette Logik der Buttonklicks, Eingabevalidierung und des Customer-Zustands in der OrderedWindow-Klasse.

- Die Validierung der Eingaben soll in eine eigene Validierungsklasse ausgelagert werden.
- Das Handling des Customers (neuer Customer anlegen, Order hinzufügen) soll komplett von der GUI losgelöst werden, um auch andere UIs verwenden zu können. Kapseln Sie dieses Handling so gut wie möglich in eine eigene Klasse. Diese Klasse sollte keine Abhängigkeiten zu Swing haben.

Lösen des Beispiels:

- Refactoring Pattern identifizieren (z.B. „Move Method“)
- Test laufen lassen (es sollten keine Tests fehlschlagen)
- Wenn möglich/nötig einen Test für das Refactoring implementieren
- Refactoring Pattern anwenden
- Test laufen lassen (es sollten keine Tests fehlschlagen)
- Dokumentieren mit Begründung der Auswahl

Sollten Sie der Meinung sein, ein Refactoring anwenden zu können, zu dem Sie keinen Pattern Namen kennen, erstellen Sie eine kurze Beschreibung dazu. Geben Sie in der Spalte „Nutzen, Begründung der Auswahl“ an, was genau in der jeweiligen Klasse / Methode schlecht gelöst wurde

und wie Sie dies verbessert haben. Gegebenenfalls müssen Sie die existierenden Tests anpassen, da manche Anwendungen der Patterns die Klassenstruktur abändern.