Software-Qualitätssicherung VU

Übung 4

Aufgabenstellung

1 Testreview (9 Punkte)

Folgendes Beispiel beschäftigt sich mit dem Refactoring von Tests. Gegeben ist eine Applikation mit deren Hilfe der Versand von Büchern an Customer verwaltet wird. In der Klasse TestsToRefactor sind bereits einige Tests vorhanden. Untersuchen Sie die gegebenen Tests auf folgende *Bad Practices*:

- Doppelter Code
- Falsch geworfene Exceptions
- zu viele Asserts
- Schleifen und Switch
- Random-Logic
- unklare Benennung der Tests
- keine klare Abgrenzung der Tests
- usw.

Finden Sie sämtliche Bad Smells, verbessern Sie diese, fügen Sie auch Tests hinzu und/oder teilen Sie Tests, wenn es Ihnen richtig erscheint und dokumentieren Sie dies mit GIT. Eine Commit-Message sollte folgendes beinhalten:

- Name des gefundenen Bad Smells
- Begründung der Auswahl
- Nutzen des Refactorings

2 Code-Refactoring: Refactoring-Patterns (10 Punkte)

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. Verwenden Sie hierfür das mitgelieferte lokale GIT-Repository.

Eine Commit-Message sollte folgendes beinhalten:

- Name des angewendeten Patterns
- Nutzen des Refactorings
- Begründung der Auswahl
- Name der erstellten Testmethode

Im Ordner *"src\"* finden Sie den Videoverleih. Dieser besteht aus einer minimalen Swing-GUI (OrderedWindow), 3 Modellklassen (Customer, Film, LentFilm) und einer Startklasse (Runner). Führen Sie das Projekt aus und lesen Sie den Code, um einen Überblick über die Funktionalität zu bekommen.

Identifizieren Sie zuerst *"Bad Smells"* in den Modellklassen und refaktorisieren Sie diese. Erst danach verbessern Sie die Swing-GUI, indem Sie vom GUI unabhängige Logik in eine eigene Klasse refaktorisieren.

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 fur einige Refactoring-Patterns.

Kurzbeschreibung des Programmes:
Nach dem Anlegen eines Customers soll es möglich sein, zu diesem einen Film und eine bestimmte Anzahl an Tagen zu wählen. Der Output sollte folgendermaßen gestaltet sein: Zuerst kommt eine Überschrift, in der der Name des Customers vorkommen soll. Nach einer Trennlinie werden die Filme wie folgt aufgelistet: zuerst die Anzahl der Leihtage, dann der Name des Films, die Einzelkosten und anschließend die gesammelten Videopoints. Es sollte ersichtlich sein, dass man erst Filme und Tage zu einem Customer hinzufügen kann, wenn der Customer bereits existiert. Anschließend gibt es eine Zusammenfassung der Bestellung, in der die generellen Kosten und Videopoints ausgewiesen werden, sowie die Gesamtkosten als Summe der Einzelkosten der Videos und den generellen Kosten unter "Total". Pro Film, egal welcher Kategorie, gibt es 5 Videopoints. Die Preise der einzelnen Filmkategorien setzen sich wie folgt zusammen:

- Children: Anzahl der ausgeliehenen Tage * 3 Euro
- Old: Anzahl der ausgeliehenen Tage * 3 Euro * 7 / 10
- Horror: Anzahl der ausgeliehenen Tage * 3 Euro * 3 / 2

Die Anzahl der Tage muss positiv sein und der Name des Customers sollte mindestens 5 Buchstaben lang sein. Nach dem Hinzufügen eines neuen Customers sollte das Anzeigefeld geleert werden.

Refactoring Patterns:
Verwenden Sie unter anderem folgende 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 in welcher Weise deren Anwendung sinnvoll ist, um den bestehenden Code zu verbessern. Wenn möglich schreiben Sie einen Test, der die Funktionalität Ihres Refactorings überprüft und wenden Sie danach das jeweilige Pattern an. Dabei soll folgendes verbessert werden:

Modellklassen

- Erhöhung der Datenkapselung
- Im aktuellen Code muss bei jeder neuen Filmkategorie (CLASSIC, OLD, etc) die Customer- Klasse angepasst werden. Dies soll durch Polymorphismus oder Komposition verbessert werden. Beachten Sie auch, dass der Preis abhängig von der Kategorie ist, jedoch zur Zeit in der Startklasse mitgegeben wird.
- Der Preis eines LentFilms wird zur Zeit im Customer berechnet, obwohl keine Daten des Customers verwendet werden. Die Klasse LentFilm soll ihren Preis selbst berechnen können (calculatePrice()).
- Die Customer-Klasse soll Methoden anbieten, um die Gesamtkosten aller geliehenen Filme und die Video-Points abfragen zu können (getTotalPrice(), getVideoPoints()).
- Im Sinne des OOP Prinzips “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 (neuen Customer anlegen, Video hinzufügen) soll komplett von der GUI losgelöst werden, um auch andere UIs verwenden zu können. Kapseln Sie dieses Handling in eine eigene Klasse. Diese Klasse sollte keine Abhängigkeiten zum UI haben.

Vorgehensweise

- Refactoring Pattern identifizieren (z.B. ”Move Method”)
- Tests laufen lassen (es sollten keine Tests fehlschlagen)
- Wenn möglich/nötig einen Test für das Refactoring implementieren
- Refactoring Pattern anwenden
- Tests 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.

Vergleich Beschreibung - Produkt:
Nachdem Sie nach dem Refactoring schon einen guten Überblick über das kleine Projekt haben, ist es nun die Aufgabe, Unstimmigkeiten zwischen der Beschreibung und dem Produkt herauszufinden und auszubessern. Dokumentieren Sie bitte jeden Ihrer Schritte mit einer ausführlichen Commit-Message.

3 Code-Refactoring: Codeanalyse (9 Punkte)

Führen Sie eine Codeanalyse auf dem Quellcode von Commons Collections von Apache mit Hilfe von FindBugs und Checkstyle durch. FindBugs wurde in der Vorlesung bereits angesprochen. Es führt eine statische Codeanalyse auf den Bytecode von Javaklassen durch und versucht Muster im Code zu finden, die mit hoher Wahrscheinlichkeit Fehler beinhalten. Mit Hilfe von Checkstyle kann das Einhalten von Code-Konventionen automatisch überprüft werden.

Importieren Sie das Projekt als ersten Schritt das Maven Projekt in Eclipse.

3.1 FindBugs

FindBugs wurde in der Vorlesung bereits angesprochen. Es führt eine statische Codeanalyse auf den Bytecode von Javaklassen durch und versucht Muster im Code zu finden, die mit hoher Wahrscheinlichkeit Fehler beinhalten. Dokumentieren Sie die gefundenen Fehler, nachdem Sie ausgebessert wurden, in der Commit-Message. Diese sollte eine kurze Analyse uber die durch FindBugs gefundenen Fehler enthalten.

Lösen des Beispiels:

- FindBugs Eclipse Plugin installieren (über “Help -> Eclipse Maketplace”) und auf das Projekt anwenden.
- In die FindBugs-View wechseln und das Projekt analysieren.
- 5 verschiedene Bugs, die zu Recht von FindBugs bemängelt wurden, finden.

- Die 5 von Ihnen gewählten Bugs im Quellcode ausbessern.
- In einer Commit-Message den gefundenen Bug dokumentieren. Sie mussen 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.
- Finden Sie zumindest 2 Bugs, die Ihrer Meinung nach, zu unrecht von FindBugs bemängelt wurden oder werden alle Bugs zu recht bemängelt? Dokumentieren Sie ihr Ergebnis.
- Anmerkung: Falls bereits ausgebesserte Fehler weiterhin 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!

3.2 Checkstyle

Der zweite Teil der Aufgabe ist es, Checkstyle auf den oben bereits verwendeten Code anzuwenden.

Installieren Sie Checkstyle über "Help -> Eclipse Maketplace". Aktivieren Sie Checkstyle, indem sie auf das Projekt einen Klick mit der rechten Maustaste machen, auf Checkstyle gehen und 'activate Checkstyle' drücken. Nun lassen Sie das Projekt von Checkstyle überprüfen, indem Sie auf "Checkstyle -> Check Code with Checkstyle" gehen.

Suchen Sie sich 10 Checkstyle Probleme heraus und kommentieren Sie, analog zum FindBugs-Teil, ob Checkstyle hier zurecht ein Problem gemeldet hat und begründen Sie dies. Verwenden Sie dazu die mitgelieferte Vorlage.

3.3 Vergleich

Sie haben nun FindBugs und Checkstyle ausgeführt. Werden die gleichen Fehler gefunden? Werden auch zu Unrecht bemängelte Bugs gefunden? Begründen Sie ihre Antwort.

4 Testen mit Mockito (6 Punkte)

Gegeben ist ein einfaches Login Service, das ein Interface IAccountManager zur Verfügung stellt, um Benutzer im System zu authentifizieren. Im Moment existiert weder eine Implementierung der Account Klasse, noch eine Implementierung des Account Managers.

Testen Sie das Login Service, indem Sie die benötigten Klassen mit Hilfe von Mockito mocken.

Erstellen Sie eine JUnit Testklasse und testen sie folgende Fälle:

- Testen Sie, ob der Benutzer eingeloggt wird, wenn das richtige Passwort übergeben wird.
- Testen Sie, ob der Benutzer nicht eingeloggt wird, wenn das übergebene Passwort falsch ist.
- Testen Sie, ob eine AccountNotFoundException geworfen wird, sollte der Account nicht existieren.

5 Theoriefragen (5 Punkte)

1. Wofür werden Plugins wie Checkstyle und FindBugs verwendet? Welche Vorteile/Nachteile bringen sie?
2. Ab wann im Softwareentwicklungszyklus ist der Einsatz von Codeanalyse sinnvoll? Begründen Sie Ihre Antwort.
3. Welche Vor- und Nachteile bringt Refactoring?
4. Macht es Sinn, auch Tests zu refaktorisieren? Begründen Sie Ihre Antwort.
5. Beschreiben Sie ein Refactoring-Pattern, das Sie in den obigen Beispielen noch nicht angewendet haben. Erklären Sie den Nutzen und die Einsatzgebiete. Geben Sie ein Beispiel an.

6 Abgabe

Folgende Dateien sind abzugeben:

- QSVU_UEbung4_<Matrikelnummer>_<Nachname>_<Vorname>.pdf
- 1 Testreview
- 2 Code-Refactoring
- 3 Code-Refactoring
- 4 Mockito

Die Ordner (1-4) sind inklusive aller Unterordner und Dateien zu verstehen. Wählen Sie treffende Namen für die Tests bei 4. Bitte verwenden Sie die Vorlage und beachten Sie, dass ausschließlich richtig benannte Dateien im angegebenen Format bewertet werden.

Bei der Übung handelt es sich um eine Einzelarbeit. Plagiate werden mit 0 Punkten bewertet.