Praktikum Software-Engineering 1 SoSe 16

Aufgabenblatt 1

Prof. Dr. Bettina Buth <buth@informatik.haw-hamburg.de>
Raum 7.86b, Tel. 040/42875-8150

Bearbeitungshinweise

- Die Bearbeitung der Aufgaben findet in **festen Vierergruppen** statt.
- Die bearbeiteten Lösungen werden in der Regel während der Praktikumsstunde abgenommen. Dazu werden Sie abwechselnd Ihre Lösungen im Team vorstellen.
- Es gibt **100% Anwesenheitspflicht** beim Praktikum. Beim Fehlen wegen z.B. Krankheit müssen Atteste eingereicht werden und ein Nachholtermin wird vereinbart.

Ziel des Praktikums:

- Refactoring am Beispiel kennenlernen
- Speziell: Code Smells identifizieren auf Basis eines Smell-Katalogs
- Speziell: definierte Refactoring Schritte durchführen und per Test überprüfen
- Glossar mit SE-Begriffen beginnen

Vorbereitung vor dem Praktikum

1) Machen Sie sich mit den Code-Smells im Katalog von Fowler und Keriesky vertraut, zu finden unter http://s3.amazonaws.com/grabbagoftimg/Smells%20to%20Refactorings.pdf
2) Erstellen Sie ein Eclipse-Projekt aus den Dateien des TicTacToe-Spiels. Bitte beachten Sie, dass auch die JUnit3-Bibliotheken eingebunden werden müssen. (Quellen im pub)
3) Sehen Sie sich den bestehenden Code für das TicTacToe-Spiel inklusive der vorhandenen Tests sorgfältig an (ohne ihn zu verändern).

Zusatzinformationen:

- http://de.wikipedia.org/wiki/Refactoring
- http://www.refactoring.com/index.html
- http://www.industriallogic.com/blog/smells-to-refactorings-cheatsheet/
- http://www.tutego.de/java/refactoring/catalog/

Aufgabe 1: TicTacToe – eine Spielefamilie

Nach William Wake.

1.1: Smells identifizieren

Identifizieren Sie anhand des Smell-Katalogs Smells im Code des Spiels (`Game.java`).
Dokumentieren Sie die gefundenen Smells wie folgt:

Datei	Zeile(n)	Smell Name	Kurze Erläuterung

Hinweise:

- Betrachten Sie dazu zunächst die folgenden Aufgaben nicht - der Lernerfolg ist sonst geringer.
- Die Tests dürfen zunächst nicht verändert werden
- Lesen Sie die Aufgaben 1.2 und 1.3 sorgfältig durch, bevor Sie mit dem Refactoring beginnen
- Alle Endstände für die Aufgaben 1.1-1.3. müssen aufbewahrt werden für die Besprechung – dafür bitte ein Git Repository anlegen oder getrennte Dateien verwenden.
- Wenn möglich: lassen Sie die Aufgaben jeweils abnehmen bevor Sie mit der nächsten Aufgabe beginnen

1.2: Einfaches Refactoring

Entfernen Sie die folgenden Probleme eines nach dem anderen mit der Eclipse-Refactoring-Unterstützung.
Prüfen Sie jeweils mit der Durchführung der JUnit-Tests in `GameTest`, dass die Veränderungen keine Änderung der Funktion ergibt.

a) Benennen Sie die Variabel `i` um in `move` (Begründen Sie warum das sinnvoll ist)
b) Der Wert `-1` ist vom Sinn her ein Flag - führen Sie stattdessen eine Konstante `NoMove` ein.
c) Eliminieren Sie die Duplikate in der Funktion `winner()`.
d) Die Prüfung dass der Eintrag in einem Feld „-“ ist hat den Zweck zu prüfen ob das Feld noch frei ist. Extrahieren Sie eine eigene Methode für diese Prüfung und benennen Sie sie passend.

1.3: Schleifen verschmelzen

In der Refactoring-Terminologie bezeichnet der Ausdruck ``Fuse Loops'' die Kombination von zwei Schleifen in eine. Führen Sie diese Kombination von Schleifen am Beispiel der `move`- Methode durch - aber in kleinen Schritten wie unten angegeben. Diese erlauben die Prüfung mit JUnit.

Anmerkung: Fuse Loops ist keine von Fowlers Refactoring Methoden, sondern eine Standardtechnik, die zB. auch bei Compilern verwendet wird.

In der Methode `move` finden sich zwei `for`-Schleifen und drei `return`-Statements - das ist kein guter Programmierstil (Mehrfach-return) und soll geändert werden. Gehen Sie dabei wie folgt vor:

a) Prüfen Sie ob die Rümpfe der beiden Schleifen schon direkt in einer Schleife kombiniert werden können. Welche Gründe könnte es geben, die dagegen sprechen? Führen Sie die Rümpfe hier noch nicht zusammen, sondern erst in Schritt g).
b) Führen Sie für den zweiten Schleifenrumpf eine temporäre Variable `defaultMove` ein, die den Rückgabewert des ursprünglichen `return` beinhalten soll. Passen Sie den Schleifenrumpf unter Verwendung dieser Variablen entsprechend an, ebenso das `return`-Statement.
c) Ist eine initial-Zuweisung an `defaultMove` sinnvoll? Falls ja, welche?
d) Stellen Sie durch die Tests sicher, dass sich am Verhalten nichts geändert hat!
e) Ändern Sie das Verhalten des zweiten Schleifenrumpfs: das explizite Setzen von `defaultMove` ist unnötig, da ja bei der bisherigen Strategie egal ist, welches Feld wir zurückliefern.
f) Im ersten Schleifenrumpf bricht die Methode `move()` ab, wenn der Gewinnzug gefunden ist – auch das ist unnötig, solange am Ende sichergestellt wird, dass ein Gewinnzug auch zurückgeliefert wird – ändern sie den ersten Schleifenrumpf entsprechend.
g) Nun können die beiden Schleifenrümpfe in einer Schleife zusammengefasst werden
h) Prüfen Sie ob die Tests in `GameTest` angepasst werden müssen und ändern Sie sie gegebenenfalls.
i) Hat Ihr Code noch mehrere `return`? Warum?
j) Was wäre vermutlich passiert, wenn Sie die vorigen Änderungen alle auf einmal durchgeführt hätten?

Aufgabe 2: Glossar für SEA

2.1: SE-Begriffe – Erstellen eines Glossars

Erstellen Sie ein Glossar für die folgenden Begriffe und Abkürzungen, die im Zusammenhang mit Software-Engineering relevant sind.

- Glossar
- Agile Software-Entwicklung
- MVC
- Requirement
- Review
- Software Qualität
- Test
- UML
- Zustandsautomat

Geben Sie für jeden Begriff eine knappe Erklärung mit eigenen Worten an sowie mindestens drei Quellen, in denen Sie Informationen zum Begriff oder der Abkürzung gefunden haben.
Finden Sie heraus, wie unterschiedliche Arten von Quellen angegeben werden können
Verwenden Sie folgendes Format für Ihre Glossareinträge:

Begriff	Erläuterung	Quellen
Glossar	Text in eigenen Worten	Webseiten oder Bücherch oder Zeitschriften oder…
…..		

Hinweis: Abgabe der Aufgabe 2 schriftlich, per email an buth@informatik.haw-hamburg.de mit Betreff „[SEP1] Aufgabenblatt 1, Aufgabe 2“ – Kopie an alle Teammitglieder

---

Abgabe der schriftlichen Teile bis

Do, 21.4.2016, 18:00

Viel Spaß!