

Übungsblatt 4

Ausgabe: 06.01.2022

Besprechung: 17.-20.01.2022

16 Punkte

Bitte laden Sie die Übungsbeispiele rechtzeitig am Sonntag (16.01.2022) vor der Übungsstunde bis spätestens 23:55 als Zip Datei (`<Matrikelnummer>_<Name>_SE1_ÜB<Nummer>`) in Moodle hoch und tragen Sie Ihre Kreuze in ZEUS ein. Die Zip Datei muss alle gelösten Beispiele (auch den Source Code) und evtl. deren Präsentationen beinhalten. Da es in ZEUS keine halben Punkte gibt, wurde die Anzahl der Punkte für ZEUS verdoppelt → laut ZEUS können Sie für diese Übung 32 Punkte bekommen. Für die Übung selbst werden aber die am Übungsblatt angegebenen Punkte gezählt → es können maximal 16 Punkte erreicht werden (= Anzahl der Punkte in ZEUS / 2). Sie müssen in der Lage sein, die Lösung und den Lösungsweg Ihren KollegenInnen und der LV-Leitung in der Übungsstunde via Beamer erklären können. Sollten Sie Fragen zur Übung haben oder Unklarheiten auftreten, nutzen Sie bitte das Moodle-Forum.

Anmerkung zum Kreuzen: Da die Beispiele in diesem Übungsblatt grob-granular sind, gibt es auf die Teilaufgaben Punkte, um zu verhindern, dass Sie alles oder nichts kreuzen müssen. Das heißt, Sie kreuzen in der Kreuzerlliste die **Teilpunkte** die Sie gelöst haben und erklären können. Sie finden bei jeder Teilaufgabe, für die es Punkte gibt, am Ende in Klammer stehend wie viele Punkte Sie für diese Aufgabe bekommen und eine "Kreuzerlnummer", um Ihnen das Kreuzen zu erleichtern.

Generelle Anmerkung: Benutzen Sie für diese Übung das im Moodle bereitgestellte Projekt. Vermerken Sie bitte Ihren **Namen** und Ihre **Matrikelnummer** als Kommentar in all Ihren Klassen. Weiters werden in dieser Einheit die Tools JUnit¹ zum automatisierten Testen, JaCoCo² zur Messung der Code Coverage, und PIT³ zur Durchführung der Mutations Analyse verwendet. Es wird empfohlen sich vor Beginn der Übung mit den Tools auseinander zu setzen. Eine Einführung zu den Themen finden Sie unter anderem auf Google. Einige Tutorials sind auch hier angeführt: JUnit<http://www.vogella.com/tutorials/JUnit/article.html> . Die Webseite des Buches "xUnit Test Patterns"⁴ bietet zudem einen Überblick über Best Practises zum Thema Testing (z.B. Wie sehen gute Tests aus?).

Abgabedateien

Die Zip Datei `<Matrikelnummer>_<Name>_SE1-ÜB<Nummer>` sollte folgende Dateien beinhalten:

- 4.1.Analyse.pdf
- 4.1.Schritt_<N>_reports.*
- project.zip

¹<http://junit.org/junit4/>

²<http://www.eclemma.org/jacoco/>

³<http://pitest.org>

⁴<http://xunitpatterns.com>

Testing

4.1 Unit-Testen Praxis

(5 Punkte)

Gegeben ist die Klasse `at.aau.ue4.bsp1.RingBuffer`. Verwenden Sie für dieses Beispiel wieder das auf Moodle zur Verfügung gestellte Projekt. Verwenden Sie für Ihre Tests die bekannten Techniken aus der Vorlesung und den Übungen. Mit `mvn clean test` (oder alternativ via IDE) können Sie die Tests laufen lassen und die Reports generieren. Die Berichte befinden sich anschließend im `target`-Folder Ihres Projekts. Geben Sie abschließend die Berichte mit ab. Jeweils nach den Schritten 2,3 und 5 führen Sie folgende Messungen durch:

- a. Messen Sie mittels JaCoCo die Coverage-Metriken für die Klassen und erstellen Sie den Coverage-Report.
- b. Messen Sie die Qualität Ihrer Tests mittels dem Mutation Testing Tool PIT.⁵

Arbeitsschritte:

1. Überlegen Sie sich eine Strategie um die Klasse zu testen und notieren Sie sich Ihre Vorgehensweise (Wie gehen Sie vor um gut zu testen?, Welche Strategien wenden Sie an?, Welche Verfahren eignen sich hier gut?,...).
2. Implementieren Sie die Tests gemäß Ihrer Strategie mit Hilfe von JUnit in der Testklasse `at.aau.ue4.bsp1.BaseTest`. (1 Punkt, K1)
3. Erweitern Sie die Tests um volle Branch-Coverage in der Testklasse `at.aau.ue4.bsp1.FullCoverageTest` zu erhalten. (1 Punkt, K2)
4. Erweitern Sie die Tests um 100% Mutation-Score von Jumble zu erreichen. (`at.aau.ue4.bsp1.BaseTest`) (2 Punkte, K3)
5. Wie hängen Code-Coverage Metriken und Mutation Score zusammen? (1 Punkt, K4)

4.2 TDD Feature Implementieren

(6 Punkte)

Gegeben ist die Klasse `at.aau.ue4.bsp2.MyCollection` im Package `at.aau.ue4.bsp2`. Diese Klasse stellt eine stark vereinfachte Implementierung einer Collection dar. In diesem Beispiel sollen Sie zwei Features (`remove()` und `empty()`) mittels Test Driven Development⁶ implementieren. Um die Vorgehensweise zu dokumentieren, verwenden Sie Git⁷. Achten Sie darauf gute Commit-Messages zu verwenden.⁸

1. Erstellen Sie im Projektverzeichnis ein Git Repository (`git init`) und committen Sie das Projekt (`git add .` und `git commit`)
2. Folgen Sie nun der TDD Vorgehensweise und erstellen Sie Tests die die Spezifikation laut JavaDoc des ersten Features testen. (1 Punkt, K5)
3. Lassen Sie die Tests mittels `mvn test` laufen (die Tests müssen fehlschlagen) und committen Sie die Tests (`git commit`). (1 Punkt, K6)

⁵<http://pitest.org>

⁶<https://www.it-agile.de/wissen/agiles-engineering/testgetriebene-entwicklung-tdd/>

⁷<https://git-scm.com/docs/gittutorial>

⁸<https://chris.beams.io/posts/git-commit/>

4. Implementieren Sie die Funktionalität des Features. Die Tests müssen anschließend erfolgreich sein. **(1 Punkt, K7)**
5. Wiederholen Sie Schritte 2. bis 4. für das zweite Feature. **(2 Punkte, K8)**
6. Lassen Sie Coverage und Mutation Analyse laufen, verbessern Ihre Tests falls notwendig und committen Sie am Ende erneut. **(1 Punkt, K9)**

4.3 Basispfade nach McCabe

(2 Punkte, K10)

Gegeben ist Listing 1.

Listing 1: Codebeispiel

```
1 public static int ggt(int zahl1, int zahl2) {  
2     while (zahl2 != 0) {  
3         if (zahl1 > zahl2) {  
4             zahl1 = zahl1 - zahl2;  
5         } else {  
6             zahl2 = zahl2 - zahl1;  
7         }  
8     }  
9     return zahl1;  
10 }
```

1. Bestimmen Sie die Basis-Pfade nach McCabe.
2. Implementieren Sie konkrete Testfälle in der Klasse `at.aau.ue4.bsp3.McCabeTest`, so dass alle Basis-Pfade zumindest 1x durchlaufen werden und erstellen Sie einen Coverage Report und eine Mutationsanalyse. Was können Sie beobachten?

Metriken

4.4 Size oriented Metrics

(1 Punkt, K11)

Ermitteln Sie die LOC, Non-blank LOC, SLOC und LLOC für die Methode aus Listing 2. Welche Vor- und Nachteile haben die Ermittlung von LOC bzw. SLOC und LLOC?

4.5 Metriken nach McCabe

(1 Punkt, K12)

Ermitteln Sie die zyklomatische Komplexität für die Methode aus Listing 2.

4.6 Interpretation von Code Metriken

(1 Punkt, K13)

Interpretieren Sie nun die von Ihnen gesammelten Metriken zur Methode aus Listing 2. Wie hängen die verschiedenen Metriken zusammen?

Listing 2: Codebeispiel

```

1  private static int kgV(int a, int b) {
2
3      int max;
4      int min;
5
6      // set max and min according to a and b
7      if(a<b) {
8          max = b;
9          min = a;
10     } else{
11         max = a;
12         min = b;
13     }
14
15     int i=1;
16     int kgV = -1;
17
18     do{
19         kgV = (i * max);
20         i++;
21     } while (kgV % min != 0);
22
23     return kgV;
24 }

```