Course "Software Processes" WS 2015/2016

Freie Universität Berlin, Software Engineering Research Group
Prof. Dr. Lutz Prechelt, Franz Zieris

Practice sheet 1 **JUnit, Code Reviews** due on 19.10.2015

Please make sure to always prepare your answers and solutions in a way that you are able to present them to your classmates and discuss your solution process effectively. Remember to always list any used material and references.

Learning aims

The aim of this practice sheet is to understand JUnit and to gain some experience with the quality assurance method of *code reviews*. Later in the course, we will use JUnit in the context of *Agile Methods*, and we will conduct reviews in the context of *Cleanroom Software Engineering*.

Task 1-1: JUnit

The aim of this task is to familiarize with *JUnit*, a Java framework to write and execute automated unit tests. We will utilize the most recent version 4.12. Note, however, that many projects still use the less powerful version 3.8.

1. Get to know JUnit:
   - **a)** Take a look around the JUnit website: http://junit.org
   - **b)** Read the JUnit Cookbook: http://junit.org/cookbook.html
   - **c)** Study the JUnit documentation, in particular the "Usage and Idioms" section of the JUnit website, to get a basic understanding of the framework's concepts.
   - **d)** Download JUnit 4.12 and take a look at the source code.

2. Explain JUnit:
   - **a)** Visualize and explain the life cycle of a single unit test, particularly with regard to the specific methods used.
   - **b)** Which processes are offered by JUnit to run a group of test cases? Explain the meaning of the necessary annotations.
   - **c)** What is the difference between *errors* and *failures* in JUnit?
   - **d)** Explain what a *fixture* is in the context of unit testing and what is usually put into a fixture.

Task 1-2: How to write good tests

The aim of this task is to get an understanding of what makes up good test cases and to know the *dos and don'ts* for writing them.

1. Research on what characterizes good test cases and the principles to follow for writing good tests. Write down these attributes and principles, and for each of them:
   - Explain why it is important and what its benefit is.
   - Give an example to illustrate how it should be utilized, and give a counter-example of how it should not be used.
     Use the JUnit Notation. You may use a test case you have written in a real software project.

2. What is your personal experience with writing tests?
   - **a)** Can you give an example of a case where you had something that was very hard to test? If so, show or explain the case. How long did it take to write the test in relation to the time needed for writing the production code?
   - **b)** Do you have an example where you wrote a test and had to rework it over and over again, because its design was suboptimal? If so, show or explain the case. What were the problems that prompted you to reworked your test?
   - **c)** Do you have an example for a good test you wrote? If so, show or explain the case. What, in your opinion, made the difference: What made the test *good*?

Task 1-3: Reviews

Reviews are a *manual* and *static* method for *analytical quality assurance*. In this task, you are going to conduct a code review of a manipulated version of JUnit. Read all subtasks before you start.

1. Familiarize with the process of reviewing: An introduction to reviews was given in the course "Softwaretechnik". Read up in the original slides[1] or use some other source found on the web.
2. Conduct a code review: Download the file **`01_junit_src.zip`** from the lecture's resources (KVV) and unpack it. Conduct a review of the class **`BlockJUnit4ClassRunner`** (directory **`src/main/java`**, package **`org.junit.runners`**) and list all defects found.

   Pay attention to the following aspects:
   - We are concerned with *static* reviews. Thus, the code should not be executed.
   - Focus on discovering defects resulting from programming mistakes.
   - Rather than looking for all types of defects at the same time, it is suggested to concentrate on a certain perspective such as *exception handling* or *concurrency*. Which perspective would suit this particular class?

   In case you do not understand certain parts of the code, consult the code of the surrounding classes (referenced or referencing) or read the JavaDoc in the **`javadoc/`** directory.
3. Write down how long your review took for each method (and, where applicable, each attribute). Calculate the total effort, the average effort per line of code and the average effort per found defect.

[1] http://www.inf.fu-berlin.de/inst/ag-se/teaching/V-SWT-2015/42_Analytische-QS2.pdf#page=28