Course "Software Processes"
WS 2013/2014
Prof. Dr. Lutz Prechelt, Julia Schenk

Freie Universität Berlin

Practice sheet 1 – due on 2013-10-21

Please prepare your solutions / answers in written form. Make sure to always prepare them in a way that you are able to present them to your class mates and discuss your solution process.
Please remember to always list your references.

Learning aims: The aim of this practice sheet is to **understand JUnit** and to gain first experiences with the **quality assurance method** of **reviewing**. Later on in the course we will use JUnit in the context of »Agile Software Processes« and reviews with respect to »Cleanroom«.

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 be utilizing the most recent version 4.11, however note that many projects still use the less powerful version 3.8.

1. **Get to know JUnit:**
   a. Find initial guidelines on the project page of JUnit: http://junit.org/.
   b. Then read the **JUnit Cookbook** (http://junit.sourceforge.net/doc/cookbook/cookbook.htm) and
   c. Study the **JUnit Documentation** (link on http://www.junit.org) in order to get a good understanding of the framework.
   d. For a better understanding, download JUnit 4.11 and view the sources.
2. **Explain JUnit:** Answer the following questions in a way that you are able to *explain* the answers to your fellow students:
   a. Visualize and explain the life cycle of a single unit test, particularly with regard to the specific methods used and their order.
   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 name what is usually put in 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 good test cases.

1. **Drill down on what characterizes good test cases and the principles to follow for writing good tests.**
   a. Note down any attributes / principles you can find for good test cases.
   b. Explain why each attribute / principle is important and what the respective benefit is.
   c. Illustrate each attribute / principle in an example showing how the attribute - principle should be utilized, and give a counter-example of how it shouldn't. Use the JUnit Notation. (You can use a test case you have written in a real software project.)
2. **What is your 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 refactor it again and again because it was designed somewhat less than perfect? If so, show or explain the case. What were the problems that prompted you to refactor 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 method of manual static analytical quality assurance. In this task, you are to conduct reviews of a manipulated code version of JUnit 4.8.2. This version can be found on the lecture's website as ps_1_junit_4.8.2_src_error.zip.

1. **Familiarise with the static process of reviewing**: An introduction to reviews was given in the course »Softwaretechnik«. Read up in the respective set of slides[1] or some other source found on the web.
2. **Conduct a code review**: Download the file ps_1_junit_reviews.zip from the website and unpack it. Conduct a review of the class `org.junit.runners.BlockJUnit4ClassRunner.java` and list all defects found. Pay attention to the following:
   a. Remember: We are concerned with **static reviews**, thus, the code should not be executed.
   b. Focus on discovering defects resulting from programming mistakes.
   c. 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?
   d. In case you do not understand certain parts of the code, consult the code of other classes or the JUnit JavaDoc (http://kentbeck.github.com/junit/javadoc/latest/)

In the Wiki (Prepared Page Tabelle Code Review Efforts): Note how long the review took for each method (and, if applicable, each attribute). Estimate the total effort, the average effort per code line and the average effort per defect found.

[1] http://www.inf.fu-berlin.de/inst/ag-se/teaching/V-SWT-2013/42_Analytische-QS2.pdf from slide 28