Software Testing 2019

Term Project

Aim. In the context of software engineering, a developer is the one who writes the code and the tester is the one who tests code. However, we believe that even if these two roles are distinct, they are also complementary. In this project, your group will experience software testing from both perspectives. There will be some constraints, but you will have enough opportunity to use the testing techniques and approaches you heard about during this course.

Method. You will work in groups of 4-5 students and perform the following activities: (1) coder-view testing, (2) tester-view testing and (3) project evaluation and reflection. You will start by developing (and of course testing) your own software product. After three weeks, the roles will change; your product will be black-box tested by another group, and you, in your turn, will get a new product to test. Finally, after another two weeks, you will write a report and present your experiences in class.

Your tasks

Task 1. Coder-view testing

There are three possible project ideas available, that require roughly the same amount of work. All product requirements have been deliberately left vaguely defined, to encourage variation and creativity.

Project idea #1: A Heart Monitor

This program simulates the controller of a heart monitoring device. The system continuously monitors the heart functions of a hospital patient. Your monitor should read the pulse, oxygen level and blood pressure. Its main purpose is to raise alarms if there is something seriously wrong, and the patient might die. Search medical literature to determine the normal and life threatening values. Create differentiated warnings, because not all combinations of readings are equally dangerous. For example, the fact that a patient has high blood pressure is less dangerous on short term than low pulse and low blood pressure. As this is just a simulation, you will have to replace real sensors readings with simulated data.

Project idea #2. A Hangman Game

This program simulates a Hangman game[1]. Basically the user has to guess a secret word by trying one letter at a time. If the letter is correct, then the computer shows the places where this letter occurs in the word; if not, then a new element is added to a hangman drawing and the user gets one step closer to the

[1] https://en.wikipedia.org/wiki/Hangman_(game)

“disaster”. Obviously, the number of tries is limited. The game ends when the user wins before the drawing is complete, or when the user loses and is “hanged”. The program should supply the user with feedback information, such as the letters he tried, the number of trials and if not guessed, also the right word.

Product idea #3. A Telephone Switching System

This project simulates a system responsible for routing telephone calls. The system is not hooked up to real phones, but will instead read input commands from the keyboard or a file. The system continuously displays the status of each phone and call, such as: “phone xx hears dialtone”, or “phone xx and phone yy are talking”. Think here about malfunctions and recovery from malfunctions.

Your group has to choose one of these three project ideas (Heart, Game or Switch) using a *first come-first served* mechanism. The task is to build a bug-free software product that implements the specified idea. Let us know about your choice. Exactly like in a real company, you should adopt a certain development methodology such as (1) Waterfall or (2) Agile Scrum with its variant, (3) Test Driven Development (TDD).

The next step is to produce a software requirements specification (SRS). Be aware that your SRS will be critically inspected by another team in three weeks, so try to carefully formulate your requirements.

The programming language used for implementation can be only Java or Python, because for these two languages we can suggest the right testing tools. **You should write your code from scratch, without any GUI or API.** In this way nobody else will be to blame except you, in case bugs will be found.

Write a test plan, including your test strategy and test cases. You can use all the testing techniques you learned about in class. Only writing test cases is not enough; you should also explain why you used a certain strategy or technique. Because there is a high chance that you will run out of time at the end of this phase, it is wise to prioritize both your requirements and your test cases. Don't forget to justify these decisions. Evaluate your testing by using code coverage and mutation tools. We also recommend to continuously monitor your testing process, for example by using a bug tracking system in order to extract metrics for your testing process. Don't forget to register the time you spent on each activity.

Wrap up your code and requirements and make them ready for exchange by Sunday 5 May 23:59. You should produce an executable for Windows and Linux. We will exchange the products on **Monday 6 May** after you shortly presented your products in the class.

Task 2. Tester-view testing

Now you are going to experience another role, that of a black-box tester. If everything went fine, the projects have been swapped. You received an executable file developed by another group, together with their requirements specifications. Your task is to black-box test this product. The time is again limited. So you have to write a test plan. First, take some time to verify the requirements document and write a report on this. After that, generate adequate test cases, prioritize them and justify these decisions. Make, if needed, assumptions on the requirements in order to generate your test cases. You should prepare a test report, where you specify your strategy, all the test cases you executed and whether they passed or failed. If you encounter a bug that blocks your testing (called show-stopper), meaning that you cannot continue testing because the program crashes, go to the developers and fix it right away. Don't forget to mention these incidents in your test report. Also, register the time you have spent in this phase. In the last week, you will hand in the test report to the developers, so that they can reflect on their own testing.

Task 3. Final report

Now that you practiced testing in both roles, you can prepare a final report that puts things together and reflects back on the experiment. This report should contain the following items:

- For product #1: Your requirements, your white-box test approach (techniques, metrics), the feedback you got from the testers and your reaction.
- For product #2: Requirements verification report, your black-box unit test approach (techniques, metrics) and the test report you sent to the developers.
- A reflection. In this reflection, you should state how you selected the testing techniques, what testing you performed in addition to your original test plan, and why you did it; how realistic was your planning and how you decided to stop testing; how compares your white-box testing to the black-box test report you received, and how do you feel about this; what went fine, what went wrong, and whether you had any surprises; summarizing, what did you learn from this entire experiment.

Finally, you will give a short presentation about all your findings on Monday 20 May. The deadline for the project report is **Sunday 26 May 23:59.**

Schedule

Day	**Activity**
Mon 15 April	Start the project. Choose product#1
Sun 5 May	Wrap up Product #1 (executable + SRS) and make it available
Mon 6 May	Get a new product#2 to test (executable + SRS)
Fri 17 May	Send your product #2 test report to its developers
Monday 20 May	Present in class your testing experience
Sunday 26 May	Finalize your report

Assessment

Your will be graded on the quality of your final report.

Appendix A

Recommended testing tools

	Black-Box testing	Java	Phyton
Requirements verification	NASA ARM		
Bug tracking	Mantis, Jira		
Combinatorial testing	AllPairs, ACTS		
Static analysis		FindBugs, IntelliJ IDEA IDE	Pylint, PyCharm IDE
Code coverage		djUnit, CodeCover, EclEmma	Coverage.py
Mutation testing		Jumble	MutPhy
Source code freezing		not needed	PyInstaller
Model based testing		ModelJUnit	