Project

The purpose of the project in this course is to test an open-source application systematically and thoroughly. You will be working in **teams of 3 people** for this task. First you need to choose the *subject* software application that you plan to test. You will then write a short proposal. Next, you desgin, develop and implement your tests, execute the tests, and record the results (including failures/faults found). You will finally present your work to the class during the last 4 sessions of classes.

NOTE

Note that "systematic" testing means you need to deliberately write tests towards a specific goal. For example, if the goal is to achieve full branch coverage then you should not just (randomly) write/add tests and then measure the branch coverage level; rather, you should deliberately write/add tests that increase the existing branch coverage level. In other words, for any test case that you add to your test suite, there must be a good justification behind it i.e., that is, it increases the coverage level of interest or it is a corner/special/boundary case etc.!

DO AS MUCH VALIDATION AS YOU CAN!

Validation is an inseparable part of any testing activity. This means, you should do validation too as much as you can. For example, if you are testing a program that implements the game of chess, you need to validate whether or not all the chess rules and necessary safety checks are included and implemented or not. Invalid boards, different kinds of draw in chess such as draw by repetition are included and implemented or not? Also, does the software do anything that must not be done (recall the second half of the validation battle: the software does something that it is not supposed to do!) For example, imagine there is a chess "in_check" function that is supposed to return true if a player is in check position. The function does that and correctly returns true if a player is in check (and false otherwise) but it (in)advertently modifies the board!

IMPORTANT NOTES

- Read all the instructions below carefully.
- The deadline to submit project proposals is **Wed 4/9 by 11pm (EST)**.
- Presentations will happen during 4/21-4/28 period. A link will be provided later to schedule your presentations.
- Project deliverables are due **Monday 4/28 by 11pm** on Gradescope.

Subject Application

"Subject application" is an existing open-source application that you choose to use for your project work. We refer to it as software under test (SUT) also. There are no strict restrictions on the type, size, utilized technologies etc. of the SUT, but you need to comply with the followings:

1. The chosen SUT does not necessarily need to be large in size, but must not be trivial either (a very rough measure is that the SUT must be more than at least **2k** lines of **logic** source code). Note that this is a software testing class and not a software development class. So, it is way more important to develop high quality and smart tests and conduct as many different types of testing as possible, with the **ultimate goal of revealing as many faults as you can.**

2. The SUT can be in any language and/or use any technology as long as it is possible to apply (all or some of) the coverage criteria discussed in the class.

TIP

If you use or know of some open-source project that you are passionate about, you may use that as your SUT. Make sure though it is testable and it is reasonable in size (neither too huge nor too trivial). If not, github hosts more than multiple million repositories. Search and find a peice of software that you like! Simple web apps, simple games (e.g., board games such as chess, uno, stratego etc), utility programs (e.g., word processors, PDF utility, task management, reminder apps, sleep tracking etc) can be good candidates.

CAUTION

1. **The SUT you choose must NOT already come with or have considerable tests written for it.**

2. **You may not choose a subject application that you have (co)authored!**

Proposal

The proposal submission is due *4/9 11pm EST* on Gradescope. If there are any concerns with your proposal, the instructor will contact you directly within few days of your submission. You should write at least two pages to provide a "roadmap" of the work and how you plan to accomplish the work as a team. At the very least, a proposal must include the following sections/information:

1. Link to the subject software application
2. An explanation about the subject application, what it is used for, on what platforms, by what set of users, etc. Also, include some basic metrics on the application such as the lines of code (LOC), number of classes/methods etc.

CAUTION

Note that LOC only includes the lines of **source** code that contain complex logic in them; as such this excludes trivial lines (e.g., setters/getters code etc.), config, readme, input/output, IDE-related, etc. files.

3. What types of testing you plan to conduct. At the very least, you must do:
    i. blackbox and whitebox unit & integration testings
    ii. one other type of testing as applicable/appropriate to the SUT (e.g. Mutation testing, GUI Testing, Load Testing, Mock Testing, Usability testing etc.) As an example, if the chosen SUT is a web application, you may do REST API testing for the back-end on top of regular unit/integration testing of the back-end and then do front-end testing using tools such as Selenium to do and/or maybe even load testing!
4. Test tools, technologies and frameworks you plan to leverage.

INFO

Ideally, the subject app must come with some useful resources to facilitate blackbox testing e.g. (formal) specs/comments, API doc, user manual, etc.

> **CAUTION**
>
> Note that you need to study and familiarize yourself with the subject application (at some high-level at least) and decide what types of testings are relevant. For example, GUI testing is not relevant if the app does not have a GUI-based front-end and/or there are no tools that you can easily leverage to develop GUI tests for that particular app.

5. A breakdown of tasks and who in the team will accomplish what. Draw a table to list all the (sub)tasks that need to be accomplished, assign them to the team members and ideally list a completion date for each task.

> **CAUTION**
>
> Note that it is not about the length of the proposal and also not merely about the quality of the writing (even though these carry their own weight). Rather, it is more about what SUT you choose and how well you flesh out the details of your plans to accomplish the listed tasks.

Make progress on the project

Based on your proposal, make progress on the project. **Remember that the goal here is to do a systematic testing, not random testing**. This means you need to plan ahead applicable code coverage criteria and then aim to write tests to achieve as much coverage as possible. For instance, if the plan is to acheive 100% mutation score, then try to write as minimal (a) test suite(s) as possible to achieve 100% mutation score. Similarty, for blackbox testing, recruit EP, BA, and EG techniques to write smart test cases.

> **TIP**
>
> Remember the ultimate goal in software testing is to write tests that result in failures that subsequently reveal faults. This means no matter what type of testing you are doing at any

time, your primary aim remains the same: develop tests that are (more) likely to reveal faults.

IMPORTANT NOTE - EXPLANATORY COMMENTS

When implementing the tests, make sure to leave explanatory comments in your source files indicating what each test case does and how it contributes to a testing goal you have set as part of your plan e.g., a specific coverage critetion you are contributing to with this test case.

Presentation

You will make a concise presentation (8-9 minutes) on your project. The most important part of the presentation is to showcase the subject app, the tests you have developed, and run those tests **live** against the app (at least partially!) and present the results to the class. Here is a suggested breakdown:

1. **[1-2 minutes] Intro to the subject app:** For this part, you want to answer these questions: i) what is the SUT app you have chosen and what is it used for? ii) what kind of an app is it? (e.g., web-app, mobile app, desktop app etc.) and on what platform(s) is it run? iii) who are the potential/active users of the app, iv) what technologies and programming language(s) the app is implemented/developed with?

2. **[5-6 minutes] Showcase the app, tests, and test results:** In this part, you give a live demo by running the tests you have developed against the SUT. If the execution of tests take considerable time, you can start the execution but then stop it after a short time and move on to present the test results.

TEST RESULTS

The tests results include tool-generated reports, charts, spreadsheets etc. that show numbers achieved on the relevant metrics e.g., branch coverage, mutation score etc. Also, if you have found any faults, this is where you mention them and explain them.

> **TIP**
>
> Since you do more than one type of testings, it might be sensible for you to break the task here e.g., one team member demos the unit tests and another member demos the load tests.

3. **[2-3 minutes] Cocnlude and Q&A:** Summarize what you have accomplished in this project, the achieved results (notable coverage levels and/or revealed faults etc.), lessons learnt. Leave a minute for questions.

Submission

The final submission is due 4/28 11pm EST. Each group makes only one submission on Gradescope. The package you submit must contain everything you have developed/used when completing the project, but at the very least the following must be included:

1. Testing artifacts/tools if any (developed/used in the testing process)
2. The SUT (as used in the testing process).
3. Test files and/or scripts, configuration files, tools etc. (needed to execute and reproduce the reported results).
4. Presentation slides (plus any additional documents, generated reports etc. that you like to include.)
5. A step-by-step instruction how to run the tests againsts the SUT and reproduce the results presented in the final presentation/live demo.

> **TIP**
>
> To get full marks here, we need to be able to run your tests and reproduce the results you have presented, so make sure to include all the necessary information we need to know to execute your tests.

6. [If applicable] Screenshots of the running tests and the achieved results.

Grading

The grading will be based on the overall quality of the work. Some of the factors that decide the quality of a project are (in no particular order):

1. Project proposal,
2. The chosen SUT,
3. Coverage criteria and metrics used,
4. Developing high quality test cases that are derived systemically,
5. Conducting at least two different types of testing as described above,
6. Finding real faults in the SUT,
7. Making a concise and engaging presentation.

Edit this page