CS 731: Software Testing
Term I 2020-21
Project

This document contains details and problems for a project in CS 731 course. towards experimenting with testing using open source tools and the test case design strategies learnt in the course.

Guidelines

- All submissions for this assignment should be original work. **Plagiarism of any form shall not be tolerated and strict action will be taken against defaulters. In particular, no marks will be awarded for project work and no make-up work will be assigned.**
- You need to work as a team of two members with your classmates, one team of two students per problem. Please discuss amongst yourselves and finalize your team members and the problem you will be working on as a team. A form will be made available to choose your project and upload the names.
- Your submission should be uploaded in the designated folder within LMS. It will be available a few days before the submission date.
- Submissions should be named in the format <roll-number-1-2>.tar.gz
- Deadline for final project submission is Wednesday, 25th November 2020, 5 PM.
- The tarred-gzipped folder should include the files containing the source code used with complete documentation (preferably Java code, **no** Python code), the test case strategy used (from amongst the listed strategies), designed test cases, (open source) testing tools used, executable files, screen shots containing the results of testing done. Please document the details in a small, plain text README file and include the README file in the tar, gzipped file.
- Kindly use your own source code, as appropriate for the chosen project. The source code used should provide at least one complete functionality or feature (should be written in the documentation of the code) and should contain approximately 1K lines of code, excluding documentation. In addition, source code should contain features specific to the chosen project, as detailed below. For e.g., a project using CFG criteria should have source code rich in control flow structure, a project using logic based testing should have decision statements with three or more clauses a project with ISP should have clearly laid out requirements whose inputs can be partitioned based on the requirements etc.

- Evaluation will be done based on a review which includes execution of the submitted code on the test cases and static inspection.
- Half the score for each student will be based on the success of team effort and half for each student's individual contribution. Please clearly document each team member's contribution in the README file.

Problems

- CFG-graphs: Projects that use graph based testing, with only control flow criteria. These projects need to use test cases that are designed for edge coverage and for prime paths coverage. Your code should have decision statements and loops that are nested.
- DFG-graphs: Four projects that use graph based testing, with only data flow criteria. Projects need to use du-paths based testing for designing test cases. Your code should have du-paths that are defined in presence of loops and include all-du-paths coverage.
- Design-integration-graphs: Projects that use graph based testing at the design integration level. The designed test cases need to use criteria based on last-defs and first uses. Your code should have function/method calls and pass and return parameters.
- Logic-based: Projects that use logic based testing. These projects need to use at least one of the active clause coverage criteria to design test cases. Your code should have decision statements with at least three clauses in its predicates.
- Symbolic execution based: Projects that use symbolic execution to test for path coverage in source code. Source code should have rich control structure, including nested decision statements, loops and function calls.
- ISP-based: Projects that use input space partitioning. These projects need to be applied on code that have several inputs and requirements that can be tested by partitioning the input domain. You need to design test cases based on pair-wise or T-wise coverage.
- Mutation-source-code: Projects that use mutation testing, based on mutation operators applied at the level of a statement within a method or a function. The mutated program needs to be strongly killed by the designed test cases. At least three different mutation operators should be used.
- Mutation-design-integration: Projects that use mutation testing at the design integration level, based on mutation operators for design integration. The mutated program needs to be strongly killed by the designed test cases. At least three different mutation operators should be used.

- Web applications testing: Projects involving manual modeling of web applications code, as relevant, using CIM and ATG models and testing them for coverage.

Testing tools

Please use the following testing tools as appropriate. In addition, you can use **any** available tool in the open domain to design your test cases based on the chosen project and provide details of the same in your documentation. Kindly ensure that final execution is done using Selenium or JUnit only.

- The web applications to use various coverage criteria to design test cases as taught in the course. They are available under the section titled **Support software** in the course page `http://cs.gmu.edu/~offutt/softwaretest/`.

  You can also directly generate your test cases by applying the criteria.

- Selenium: Available from `http://www.seleniumhq.org/`.

- JUnit: Available from `http://junit.org/junit5/`.