IIIT Bangalore

CSE 731: Software Testing

Term I 2025 -'26: Project Work

This document contains details and problems for a project in CS731 course in Software Testing. The goal of project work is to self-learn and understand the practical aspects of testing, through use of 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, zero marks will be awarded for project work and no make-up work will be assigned.**
- If using AI tools for any part of the project, please acknowledge the same and provide details of what exactly AI/LLM tool was used for. You will not be penalized for using AI tools, but the details have to be provided.
- You need to work as a team of two members with your classmates, one team of two students per project. 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 the LMS course page.
- Submissions should be named in the format <roll-number-1-2>.tar.gz
- Deadline for final project submission is Tuesday, 25 November 2025, 6:00 PM.
- The compressed file should contain a folder including the files containing a link to the complete code repository that is used for testing, the test case strategy used (from amongst the listed strategies), designed test cases including the complete code for the execution of the test cases, (open source) testing tools used, executable files and details/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, zipped 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 1000 lines of code or more, 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 on web applications testing should have client-side code, server-side code (at least for execution, for client-side testing), along with the required information like user-session data, based on need.

- Evaluation will be done based on a review by one of the TAs and/or the instructor. During evaluation, the code and its functionality should be briefly explained along with the design of test cases and the execution of the test cases on the code should be demonstrated. Routine testing projects will receive the same average marks out of a maximum of 15 marks. To get full/close to full marks for project, your project should stand out in terms of demonstrating the applicability of a particular technique and one or more tools.

- 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

- Data flow graphs: 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 along with all-defs coverage.

- Design integration graphs: Projects that use graph based testing at the design integration level. The designed test cases need to use the call interfaces and criteria based on last-defs and first uses of the coupling variables. Your code should have function/method calls with parameters that can be considered towards coupling data flow.

- 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. Specifically, projects using symbolic execution tools for Javascript and multi-threaded Java will be welcome.

- Mutation testing: Projects that use mutation testing, based on mutation operators applied at the level of a statement within a method or a function and at the integration level. The mutated program needs to be strongly killed by the designed test cases. At least three different mutation operators should be used at the unit level and three different mutation operators at the integration level should be used. Mutation testing tools for JavaScript will be good projects to work on.

- Fuzz testing: A variant of mutation testing, can be fully automated. Any open source tool for fuzzing can be used. Fuzzers like AFL, Peach Fuzzer and OSS-fuzz are good tools to experiment with.
- Client-side web applications testing (user session data based): Projects involving testing of client side code of a web application based on recording and using user session data from logs.
- Client-side web applications testing (bypass testing): Projects that involve testing of client side code of a web application by designing test cases that bypass client-side validation and sending changed/corrupt input to the server. You can use a mutation tool for changing client-side scripts.
- Server-side web applications testing: Projects involving manual modeling of web applications code, as relevant, using CIM and ATG models and testing them for coverage. Alternately, the graph(s) depicting the server-side code that interacts in response to requests from clients need to be modeled and test cases need to be written for end-to-end flow testing.

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.

- 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 `https://www.selenium.dev/`.
- JUnit: Available from `http://junit.org/junit5/`.
- Apache JMeter: Available from `https://jmeter.apache.org/`.
- Mocha: Available from `https://mochajs.org/`.
- Stryker mutator (for JavaScript and C#): `https://stryker-mutator.io/`
- PIT mutation testing for Java: `https://pitest.org/`
- Peach Fuzzer: `https://peachtech.gitlab.io/peach-fuzzer-community/`
- libFuzzer: `https://llvm.org/docs/LibFuzzer.html`
- AFL++: `https://github.com/AFLplusplus/AFLplusplus`

- OSS-Fuzz: `https://github.com/google/oss-fuzz`
- API testing with Postman: `https://www.postman.com/api-platform/api-testing/`