Group Project Report (20%)

Com S/SE 417 Spring 2024

Handed out March 19th, 2024

** 2 points out of 100 ***will be assigned individually*** for showing up to the final and participating in the class presentation**

Due at midnight, Tuesday, April 30th, as a pdf uploaded to Canvas

Team formation and topic selection will start after break

Homework Policy

Homework Policy: This homework ***assignment is a group assignment***. You should form teams of 4-5 students to complete this project. You are encouraged to use online and published sources and existing tools for this homework, but you will need to reference those in your report. We will allow some time during class on March 19th to help students find teams and March 21st will be dedicated time for students to talk to their team members about the project. Please use the discussion board and/or speak with the Instructor or TAs to help find teams if you are stuck. While students are encouraged to find their own teams, there may need to be some additions/reshuffling to make sure that all students in the class have a team. Students will be able to sign up for teams themselves in Canvas.

Late policy: 10% penalty per day for late homework. Assignments will not be accepted after May 5th unless otherwise arranged/discussed with me.

Note: the final exam class period will be used for short student presentations on their projects. More details on that to come.

Project Report Details

The goal of this project is to learn about ***a state-of-the-art technique*** in software testing (and to 'try it out'). You should pick something that you have not had a chance to study in depth in this class. Some of these topics have already been presented at the surface level in tools used in class, but this project gives you a chance to learn some of the fundamentals behind those techniques.

I provide example topics at the end of this report, but you can also study additional approaches as long as you they are approved.

The report should be in 12-point font (single column). Your report does not have a page limit, but a typical report will 5 pages long. You will hand in a .pdf document for this report.

As a team, first select a topic that is of interest. Then spend some time reading about this and try to find a tool/implementation that you can try out. You can combine topics below or suggest your own.

Your report should have the following parts (see grading criteria at end):

1. **Introduction/Motivation:**
   What testing problem is the technique attempting to help with? You will need to explain this to the audience (reader). You can assume general software engineering/testing knowledge, but do not assume that the reader has this knowledge.
2. **Previous/Existing Approaches:**
   What was the state of the art prior to this technique (i.e. how was testing done prior to this technique?)
3. **Technical Description**:
   Describe the technical approach taken by this advanced technique/tool. How does it work? Explain, with examples, what it does, clearly and in your own words. Use figures where helpful. It is always good to give a small example and walk the reader through that. Feel free to use examples from class (such as the famous triangle program) to explain what this tool does.

4. **Evaluation:**
   Find an implementation of this technique and try it out. Report on what you learn from this experiment. What are the advantages and disadvantages of this technique? Are there some cases where this would be useful (or not). Please describe these. Note: while we are not asking for you to submit the artifacts (test cases etc.) specifically from the running tools, BUT your report should demonstrate your efforts and provide evidence of your success running these tools.

5. **Summary and Recommendation:**
   Provide a summary of what you have learned and a recommendation for when/if you think you should use this technique.

6. **References:**
   Include the paper(s) that you read describing the advanced testing technique you're reporting on and/or the websites where the advanced tool is described. You will probably need to read some other papers, or look at some alternative tools, to understand the approach enough to explain it clearly and to see what others have done. Use the IEEE style guide to format the references:
   https://ieee-dataport.org/sites/default/files/analysis/27/IEEE%20Citation%20Guidelines.pdf

   Guidelines for grading: We will use the following weighted criteria for grading this report:

   1. Advanced testing technique & problem statement (15)
   2. Previous/alternative approaches (10)
   3. Technical approach description along with example (25)
   4. Evaluation of an existing tool (25)
   5. Summary and Recommendations (10)
   5. References (10)
   6. Report is well-written and clear (5)

Possible Topics of Study:

Note: I have tried to provide at least one tool link for each topic, but this is not complete. They are meant as a starting point. Please feel free to meet with me to discuss and/or find alternatives for these topics.

1. **Automated test generation using evolutionary algorithms**
   This approach to test generation uses evolutionary algorithms to target the lines/branches/data flow in a program. It also provides a test oracle at the end of the generation for each test case (embodied as JUnit tests). We have already used Evosutie in this class, however we have only touched the surface. If you want to explore this tool further, you can learn more about the actual algorithms that are being used, experiment with the different types of coverage and tool parameters and apply it to larger classes.

   Paper: http://www.evosuite.org/wp-content/papercite-data/pdf/tse12_evosuite.pdf
   EvoSuite tool for Java: http://www.evosuite.org/

2. **Automated web testing using Selenium (or Cypress or Playright)**

   In this class we were shown an example of Selenium but did not try it out for any assignments, etc. This project would dive more deeply into using Selenium. You can use the webdriver version or browser IDE. You would be expected to explore this further than we did in class by trying to build a more significant test suite and learning the strengths/limitations. You can also use the webdriver and learn how to code selenium on the command line.

   https://www.sciencedirect.com/science/article/pii/S1877050915005396

   https://www.selenium.dev/

   https://docs.cypress.io/guides/overview/why-cypress

   https://github.com/microsoft/playwright

3. **Combinatorial testing**
   We have seen combinatorial testing in Assignment 3 (pairwise testing). I provide a general paper below and also a link to a paper/tool that handles dependencies and/or constraints between parameters (unlike the tool we used in class). For this project you will explore constrained combinatorial testing and apply it to several models that you build on real software systems.

   Papers:
   https://pdfs.semanticscholar.org/4345/c4e91e193a0f446cdabc5d24a6d947ed14d4.pdf

https://link.springer.com/article/10.1007/s10664-010-9135-7

Tool: I can provide you with the source code if requested (the current online link is broken) or you can use ACTS (need to request from NIST)
https://csrc.nist.rip/groups/SNS/acts/documents/comparison-report.html#acts

4. **Mutation testing**
   We will talk a bit about mutation testing in class. But this will allow you to go into more detail. The idea of mutation testing is to make small changes to a program (i.e. to mimic faults) and then locate those with a testing technique. The idea is to use this to build a stronger test suite and/or to evaluate existing techniques when you don't have real faults.

   Papers: https://cs.gmu.edu/~offutt/rsrch/papers/mujava.pdf
   Tools: http://mutation-testing.org/
   https://cs.gmu.edu/~offutt/mujava/
   http://pitest.org/ (we use this tool later on in this class)

5. **Continuous Integration testing**
   As we have learned, many software development teams are using continuous integration testing (to incorporate the testing into builds). Since many of you have already used continuous integration, this project should study some advanced topics related to CI, such as test flakiness in CI or incorporating with regression test selection (deciding which tests to re-run).

   Paper: https://www.sciencedirect.com/science/article/pii/S0164121213002276
   Tools: https://jenkins.io/

   https://travis-ci.org/

   Regression test selection: http://sites.utexas.edu/august/files/2020/08/ISSRE2019.pdf
   https://github.com/august782/gitflow-incremental-builder

   Ekstazi: Lightweight Test Selection http://www.ekstazi.org/support.html

6. **GUI Testing using Vision**
   We have seen a GUI testing tool (GUITAR) in class. GUITAR uses properties of the GUI to identify widgets etc. This approach uses computer vision to identify parts of the interface and generate/run tests. You can also combine/compare these approaches.

   Paper: https://link.springer.com/article/10.1007/s10664-016-9497-6

   Sikuli Tool
   http://doc.sikuli.org

7. **Symbolic execution:**
   Symbolic execution abstracts variables into symbols (rather than using concrete input variables). This allows for 'exhaustive' program testing up to some "depth" of control flow.

   Tutorial: https://www.cs.umd.edu/~mwh/se-tutorial/symbolic-exec.pdf
   Tool: https://klee.github.io/

8. **Mobile app testing:**

   Testing Android apps requires special test harnesses (like GUIs). There are a lot of research tools. Feel free to explore. Below are a couple of papers and tools, but there are many more.

   Papers:

   https://link.springer.com/chapter/10.1007/978-3-319-99241-9_1

   https://ieeexplore.ieee.org/document/7372031

   Tools:

   https://developer.android.com/training/testing/other-components/ui-automator

   https://appium.io/

   https://developer.android.com/studio/intro

   Stoat – a model-based, search tool https://tingsu.github.io/files/stoat.html

9. **Test flakiness:**

   There have been some approaches to help with flaky tests. This work provides a tool and technique to remove/detect some types of flakiness

   Papers: https://dl.acm.org/citation.cfm?id=3330568

   http://www.deflaker.org/icsecomp/

   Tool: https://github.com/gmu-swe/deflaker
   https://github.com/idflakies/iDFlakies

   Dataset: https://github.com/TestingResearchIllinois/idoft

10. Fuzzing:

This is often used in security testing. A large number of "random" inputs are sent to a system to try to crash it and find security holes. Today fuzzers use some intelligence in their algorithms to guide them (i.e. they are not completely random). We have seen one tool in class. This project would explore a fuzzer in more detail.

Paper: https://www.usenix.org/system/files/login/articles/login_summer16_03_gutmann.pdf

Tools:

https://github.com/google/AFL

https://github.com/AFLplusplus/AFLplusplus

Java Tool:

https://github.com/CodeIntelligenceTesting/jazzer

11. Automated Program Repair:

Finding faults is only part of testing. You also need to fix the fault once found. Automated program repair aims to automatically fix program faults using intelligent search algorithms. The repairs are driven by a set of test cases which must pass in the modified/corrected program.

Paper: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6227211

Tools: https://program-repair.org/

https://github.com/coinse/pyggi

12. Using Large Language Models LLMs for Test Generation:

The use of large language models (e.g. ChatGPT, co-pilot, etc.) are changing the way we program. They leverage machine learning models which have been trained on very large datasets of code-bases. Prompting (or searching) can be used to produce code for given task. LLMs are also now being used for test generation (see paper below). This is a brand new direction for automated test generation.

https://ieeexplore.ieee.org/document/10329992

Tool: test pilot

https://github.com/githubnext/testpilot