CS 621
Idea Proposal Assignment

Due: **Wednesday, February 21, 2011, 9:00 AM EST** via Moodle.
This assignment is to be submitted via the **Idea Proposal Assignment** on Moodle.

Research idea write-up and presentation

The assignment consists of:

1. Coming up with a creative new research idea.
2. An up to 1-page write-up describing your research idea.
3. A 5-minute presentation, given in class on Thursday, February 22.

Overview

Your primary job in this assignment is twofold:

1. To describe your proposed research goal so that people understand what it is and why it is valuable. This must include a *research question* you will try to answer.
2. To describe how you will accomplish your research goal and how you will evaluate it so that it is clear how a team of up to four students can answer the research question in approximately 10 weeks.

You will present your idea to the class. Everyone will then have the opportunity to review the presentations and form groups of up to four students to actually explore the research idea!

One of the purposes of identifying the research idea is to find an area of software engineering research that is interesting to you. The idea will evolve over time, especially as you read the related work. While this initial idea may differ significantly from the final research question you tackle, the initial idea will serve an important role in focusing you on a particular area of software engineering.

Guidelines and examples

The **research idea** written description **and** the presentation must **each** contain:

- Research question. This must be in the form of a question and describe what you will know after this project is finished that the world does not know today. Examples of reasonable research questions include: "RQ1: Can mutations used in mutation testing generate the kinds of bugs observed in real, open-source development?" and "RQ2: Can the k-tail algorithm for model inference be augmented with information about pre- and post-conditions (inferred by dynamic analysis) to improve the inferred models' precision and recall?"
- The key idea behind the new technique you will develop. For example, for RQ2, the idea may be "Preventing k-tail from merging states if the pre- and post-conditions do not match."
- A concrete evaluation plan that you will use to determine when answering your research question is a success. For example, for RQ1, the plan may be "We will survey the mutation testing literature to enumerate at least 10 top mutation operators. We will then find, on `github.com`, at least 6 open-source programs. Each program will have at least 10K lines of code, 100 actively maintained tests, and at least 1000 commits in its history. We will analyze the commit history to find regression bugs (times

when a test that previously was passing but then started failing), isolate the changes that resulted in each of these bugs, and determine whether these changes could be been generated automatically using the mutation operators we identified."

Deliverables

Using the **Paper Selection Assignment** on Moodle, submit:

- An up to 1-page description of the research idea, in a .pdf. Don't forget to put your name on it.
- A digital presentation (keynote, powerpoint, or pdf). Don't forget to put your name on it.
- An in-class presentation. The delivery should take a maximum of 5 minutes. You will be cut off after 5 minutes! Prepare and rehearse your presentation.

You may use any resource you wish in this assignment but you must list your collaborators and cite all your sources. Failure to do so will result in a grade of 0.

Sample project ideas

This section summarizes project ideas discussed in class on February 1, 2018. These ideas are meant as starting points. Students are encouraged to generate their own ideas, or to start from these high-level topics and generate a more concrete idea as a course project.

Generating tests from natural language descriptions.

Software projects are more than just code. There are tests, requirements documents, bug reports, etc. Many of these documents are written in natural language. At the same time, projects' test suites are often incomplete and fail to capture some notions described in these natural-language documents. This project focuses on the problem of generating executable tests from natural-language descriptions of code.

Armed with a way to create tests, one could evaluate if they improve the quality of the existing test suite via coverage or mutation testing, or you could use the generated tests in a downstream process, e.g., automated program repair.

Universal model inference.

There are many model inference tools out there that use logs to produce a finite state machine model (or something close), e.g., Synoptic, InvariMint, kTails, BEAR, Perfume, etc. Different models are useful for different tasks and come with different levels of precision, recall, etc. But each has its own interface and it is hard to apply different tools to the same log, to export the models in a consistent format, and to compare those models or compute their derivatives (e.g., the intersection of multiple models).

This project involved building a universal front end that interfaces with multiple back-end model inference tools and allows the user to display models inferred by the tools and to manipulate the models by intersecting, differencing, unioning, etc. them using standard finiate state machine manipulation algorithms.

To get an idea of what the front end will look like, take a look at Perfume's front end: `http://perfume.cs.umass.edu/`. You can watch a video on how to use it here: `http://perfume.cs.umass.edu/demo/`. We have begun writing code for the universal front end, which you would build on. The back ends are written in Java and front ends in JavaScript.

Verification-guided sound refactoring.

Suppose you have a program and you want to refactor it to have it do the same thing but be simpler, more maintainable, etc. Create lots of mutants of the program. Identify ones that improve some useful metrics (filter out the rest). Run either an existing or a generated test suite to filter out mutants that do not behave the same way as the original program. Take the mutants that are left and (maybe automatically) use symbolic execution to verify that they are equivalent to the original program (e.g., use the Z3 theorem prover). Suggest the equivalent mutants that improve code metrics as sound refactorings that improve the code.

Reducing log size for model inference.

Use techniques in using reduced sample sizes to make high-certainty assertions to reduce log sizes needed for model inference. Read about reducing sample sizes here:
`http://cs.brown.edu/~pvaliant/unseenVV-stoc.pdf`
or watch a talk here:
`http://cs.brown.edu/~pvaliant/stoc_11A_4.mp4`

Misdirected exception handling.

When an exceptions propagates to the top and crashes a program, instead, consider handling it using one of the handlers for that type of exception elsewhere in the codebase. Maybe do some smart thing to figure out which handler to apply. Maybe get exception handles from open-source code. Figure out where to handle the exception. In some domains, it's better to get into some imperfect state and keep going than to crash altogether. For those domains, we can just ignore the exception, but that's extreme. This provides a middle ground. Project would first focus on studying real programs to see how often this could get the desired behavior.

Fairness testing.

This is a large potential area of research that is a priority for my lab. Take a look at our recent paper `http://people.cs.umass.edu/~brun/pubs/pubs/Galhotra17fse.pdf`