Milestone 1 Report

Ayaan M. Kazerouni
Virginia Tech
Blacksburg, Virginia
ayaan@vt.edu

CCS CONCEPTS

• **Software and its engineering → Software testing and debugging**; *Process validation*; *Software evolution*; Software notations and tools;

KEYWORDS

ACM proceedings, LaTeX, text tagging

ACM Reference Format:
Ayaan M. Kazerouni. 2017. Milestone 1 Report. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, Article 4, 2 pages. https://doi.org/10.475/123_4

1 SINCE THE PREVIOUS REPORT

The project has changed slightly from the original proposal. The **goal** of the project remains the same: find a way to empirically differentiate effective and ineffective software testing behaviors. The following has changed:

- **Data source:** Instead of using the Enriched Event Streams data as provided by the MSR conference organizers, I plan to use similar data generated by over 300 students from a post-CS2 junior level Data Structures & Algorithms course. The students work on 4 projects with lifespans of 3 to 4 weeks over the course of the semester, and we collect Git snapshots and IDE-events as they program.
- **Analysis:** Instead of analyzing IDE-events, I plan to do static analysis on detailed Git histories. The hope is that this will provide much-needed context into the activities of students as they program.

To assess testing behaviors, I am doing the following for *each method* in *each project*:

- Find the commit in which it was first *declared*
- Find the commit in which it was first *invoked in a test*
- In some way, calculate 'effort' expended between these two events

The measure of effort between the two events will tell us how much work the student performed between the creation of a method (its declaration) and the testing of a method (its invocation). This can further inform us about how disciplined a student is about testing units of code.

The work done so far can be found at the private Bitbucket repository, shared with the instructor [1]. I use the open source library RepoDriller [2] to help mine student code repositories. I have also begun contributing to the library. To identify commits of interest, I `checkout` each revision and parse the project into an Abstract Syntax Tree (AST) using the Eclipse JDT API [3]. As I visit each commit, I maintain a persistent data structure of methods, allowing me to store and synchronize the commits of interest.

The effort measure: As a naive first pass, I calculated the time between the two commits of interest described above. This is only a first pass because time is not a particularly good measure of work done between two events, since the student could be working on multiple things in parallel, or could have not been working on anything at all. Therefore, a better measure of the work done between the two commits would take something more certainly 'on-task' into account, like the actual code changes.

A natural candidate for this is the `diff` between the two commits of interest. To do this, I `checkout` the method declaration and invocation commits, and calculate Levenshtein's distance between the two. In the prototyping stage, I have been working with a subset of 10 student projects from the Fall 2016 semester. This calculation is an expensive one, and this small subset took $\tilde{2}0$ minutes to process. Therefore, I will switch to simply using the size of Git diffs between the commits, which are much faster to obtain, and (according to my intuition) will be linearly correlated with Levenshtein's distance.

2 FUTURE WORK AND CHALLENGES

By the next report, I hope to have done the following:

- Chosen and implemented an 'effort measure'
- Cloned at least 100 repositories and begun preliminary statistical analysis against project outcome data (available from Web-CAT)

I foresee the following challenges:

(1) Cloning projects from the Web-CAT servers can be a slow process.
(2) The source histories and project outcomes come from disparate sources. It will be a bit of work to resolve the two data sources.
(3) Processing a project takes a few minutes, and will take even more time as I mine more information out of them. All the Git `checkouts` and `resets` add to the compute-intensiveness of this process.

[1] https://bitbucket.org/ayaankazerouni/sensordata-repo-mining
[2] https://github.com/mauricioaniche/repodriller
[3] https://help.eclipse.org/oxygen/index.jsp?topic=%2Forg.eclipse.jdt.doc.isv%2Freference%2Fapi%2Foverview-summary.html

Items (1) and (2) above can be addressed by automation. Item (3) can be addressed by running overnight, or by offloading computation to the ARC supercomputer available to graduate students.