Systematic study on UI Flaky tests

Project Proposal

Sampath Grandhi SXG180072

Aditya Mathur

AXM180195

Shashidhar Krovvidi SXK180203

Satya Sarvani Baru SXB180153

Course Project Proposal for CS6367.001 Software Testing and Validation

1 Problem

An important assumption of software testing is that test outcomes are deterministic: an unmodified test is expected to either always pass or always fail for the same code under test. However, in practice, some tests often called flaky tests that have non-deterministic outcomes, i.e sometimes the test fails and sometimes it passes for the same code version. Such tests undermine the regression testing as they make it difficult to rely on test results. They are common in large systems and are troublesome to the developers. Due to their non-determinism, Flaky tests are hard to reproduce.

We specifically aim to focus on Flaky tests pertaining to UI. We want to perform an extensive study on UI based flaky tests. This study can help guide future research on mitigating flaky tests.

2 Related Work

Flaky failures frequently block and delay releases. Google spends between 2 and 16% of their compute resources re-running flaky tests.[1] There are a few studies on Flaky tests. Luo et al. empirical analysis on flaky tests classified the most common root causes of flaky tests, identify approaches that could manifest flaky behavior, and describe common strategies that developers use to fix flaky tests[2]. They have studied commit logs and for each of the commits ,the author examines it to answer these questions: Is the commit indeed fixing a flaky test?, What is the root cause of the flakiness for the test?, How can the flakiness be manifested? , How is the test fixed?

Although there are a few more studies on flaky tests in industrial setting [3] and automatically detecting flaky tests [4], there are no studies in UI flaky tests. UI tests are important as they are very difficult to automate. They can sometimes be really slow to execute on CI and older machines. It requires some significant effort to build a robust UI testing ecosystem for mobile clients.

The common approaches to deal with flaky tests are: Ignore the test if it passes only one time (declare it passing), remove the test altogether, ignore the results of this test. Flaky tests consume developer time investigating and lead to delayed project releases and waste compute resources re-running to confirm. Many developers adopted (incorrectly) a policy to eliminate it from the test suite. They might be real bugs and it’s risky to ignore them.

3 Study Plan

In order to find flaky tests, we retrieve samples from GitHub repositories. We search through GitHub issues to find examples of UI tests in web projects exhibiting flaky behavior. We search through issues and pull requests for mentions of flaky test fixes using the keywords “flaky”, “ui”, and “test”. We focus on closed issues as these would provide information on how developers resolved the flakiness observed and filter the languages to Kotlin and Swift to focus on mobile UI projects. In the flaky tests from collected issue reports, we look at the root cause of the flakiness, the symptoms caused by the flakiness, the steps to reproduce, the fixes done by the developers, how long it to took to fix the issue, and how many issues were opened on the same underlying root cause.

From these samples, we aim to find characteristics that are particular to flaky UI tests. A few root causes of flaky behaviour in UI tests are due to flaky test environment, flaky test framework and flaky tests [5]. The most predominant root cause for these flaky UI tests involved improper handling of resource loading or timing. These resources can include network resources as well as elements that have not yet been loaded in the page. This resulted in erratic behavior in the tests such as being attempting to click buttons that hadn’t opened yet. Slow UI automation and improper assumption on UI responsiveness also lead to flakiness.

Many of these issues were resolved by refactoring the code to include delays when handling a potentially flaky call. We found that the root cause of the flaky behavior could present a challenge to find and properly fix, with some issues spanning over months to fix. In addition, the flaky nature led some of these issue reports to be closed and reopened in another report as many as five times. Other root causes included platform-specific behavior, cross-platform behavior, stale data, and random data. Platform-specific behavior produces flaky results for different runs in the same platform. Cross-platform behavior causes flaky results due to inconsistencies across different platforms. Flakiness resulting from stale data is the result of improper cleanup of data after a run of the test. UI tests involving random data generation can fail intermittently because of specific characteristics of the data generated for a particular run.

4 Experimental Evaluation

Most of the work would be done manually, but we plan to use GitHub archive queries to automate collection. Our study would detail some of the root causes for UI flaky tests and will help in categorizing them for further study. We shall manifest the flakiness of flaky tests which would help in developing fixing strategies. Also we would provide some common fixes to address flaky UI tests. We aim to provide actionable information about avoiding, detecting, and fixing flaky tests. For evaluation we can connect our findings with other papers whether they are similar results or contradicting results.

REFERENCES

[1] John Micco. “The State of Continuous Integration Testing @Google”

[2] Luo, Qingzhou, Farah Hariri, Lamyaa Eloussi and Darko Marinov. “An empirical analysis of flaky tests.” FSE 2014 (2014).

[3] ILam, Wing, Patrice Godefroid, Suman Nath, Anirudh Santhiar and Suresh Thummalapenta. “Root causing flaky tests in a large-scale industrial setting.” ISSTA 2019 (2019).

[4] Bell, Jonathan, Owolabi Legunsen, Michael C Hilton, Lamyaa Eloussi, Tifany Yung and Darko Marinov. “DeFlaker: Automatically Detecting Flaky Tests.” 2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE) (2014): 433-444.

[5] Keqiu Hu. “Test Stability - How We Make UI Tests Stable” LinkedIn Engineering (2015)

Conference Name:ACM Woodstock conference

Conference Short Name:WOODSTOCK’18

Conference Location:El Paso, Texas USA

ISBN:978-1-4503-0000-0/18/06

Year:2018

Date:June

Copyright Year:2018

Copyright Statement:rightsretained

DOI:10.1145/1234567890

RRH: F. Surname et al.

Price:$15.00