UNIVERSITY OF WESTERN ONTARIO

DEPARTMENT OF COMPUTER SCIENCE

CS4470Y: SOFTWARE MAINTENANCE AND CONFIGURATION

FINAL PROJECT REPORT

Faulty: Fault Localization as a Service

Author
Gurpreet SINGH
Paul BARTLETT

Supervisor
Kostas KONTOGIANNIS

Instructor
Nazim MADHAVJI

April 10, 2018

Abstract

Lorem ipsum sodales, accumsan neque eu, placerat purus. Interdum et malesuada fames ac ante ipsum primis in faucibus. Nulla id varius metus, id vestibulum purus. Nullam malesuada urna purus, quis euismod velit tristique et. Fusce auctor laoreet arcu ac maximus. Duis ultricies malesuada dui id pharetra. Donec tempus semper enim, in interdum ante pharetra sed. Vivamus vel accumsan metus. Vivamus eu enim est. Duis ac dolor a quam lacinia interdum in ut sem. Ut ipsum orci, dignissim vel ante eget, blandit sollicitudin dolor. Sed eu orci dolor sit amet, consectetur adipiscing elit. Duis dapibus nisl vitae tempor placerat. Duis feugiat odio vitae quam pellentesque, ac semper ex sagittis. Nunc id egestas tortor. Morbi nibh tortor, suscipit vel libero quis, placerat molestie nulla. Nullam pellentesque ex ac viverra lobortis. Donec hendrerit nibh nisi, a bibendum urna efficitur ut. Cras venenatis sem magna, vel dignissim augue convallis a. Proin sapien justo, viverra ac enim sit amet, cursus aliquet tellus. Nulla at lacus magna. Nullam sit amet dui convallis, interdum felis eu, viverra ligula. Pellentesque sed mollis nibh, at ultricies nisi.Quisque id velit suscipit ipsum auctor egestas egestas sit amet dui. Curabitur at sem nunc. Nunc non ultrices ex, et egestas odio.

Introduction *(1.5 pages max)*

Concepts, Terms, Definitions, Equations *(1 page max)*

RSF

An RSF is a map of relationships betweens tokens within a codebase, where a token is a keyword in the codebase such as a method. It is generated using preprocessing scripts, and the result allows us to verify tokens inside of bug reports.

Bug Report

A database entry for each bug report used for the analysis. Each bug report contains a String field containing what a user wrote inside their bug report.

Token Expansion

Tokens extracted from the bug report are expanded to find other similar tokens. Token expansion includes tokens that are referenced by the original token set.

Clique

A collection of tokens that are referenced the most within the expanded set of tokens

Cluster

A group of relationships that are closely related to each other

Background and Related Work *(2 pages max)*

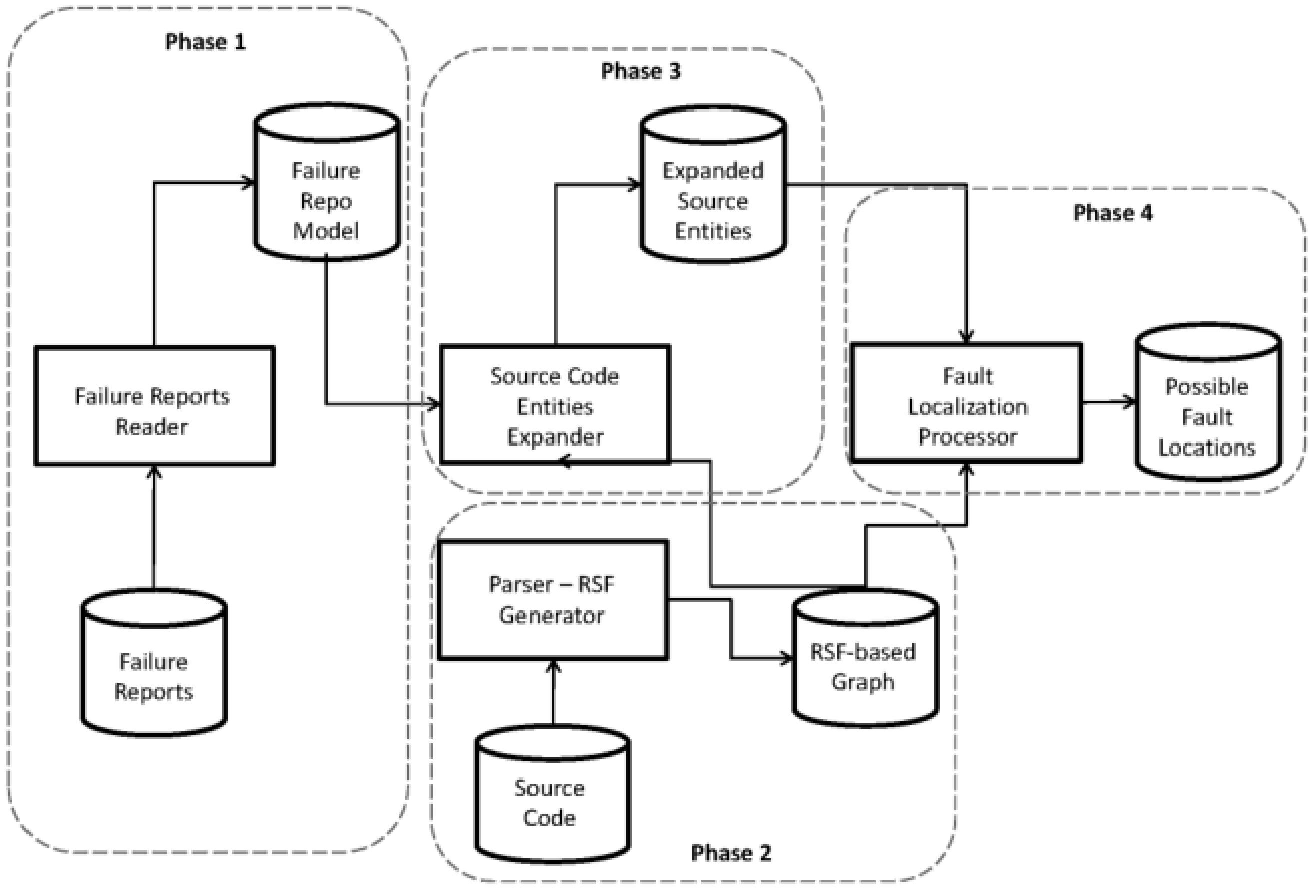
The main purpose of this project is to assist in locating faults so developers can isolate errors quicker. There are three main categories of fault localization: static-analysis/test-case based approach, machine learning based approach, and a type based approach. The method that we are expanding upon falls into the first category with an approach that heavily relies on preforming static analysis on the code and bug texts.

This project was conceived from research that Professor Kontogainnis had completed with past students regarding the task of determining faults in a large codebase using only past bug reports. Past students had experimented with multiple algorithms for processing large amounts of bug reports in order to come up with an index of files that are most likely to have a new bug within them.

The project operated in many phases. Each one containing one very specific task. The first phase involved extracting bug reports from a repository and parsing them through a reader into an object model that allowed easier processing. The second phase handled pulling the source code of the project and generating tokens. Tokens can be entities in the codebase but for our specific use they are method names that are later lifted up to filenames. A map of these tokens is created and then using the results from phase 1 and 2, the next piece of software is ready to make a connection between these two datasets. In phase 3 each of the tokens extracted from the bug report are verified and expanded to generate a set of tokens that include all relatives to each token. A token's relative is defined as a token that makes a call to or from the original token.

The first three phases mostly involved preprocessing different datasets into a form that would be ideal for making a conclusion. At this point we have a large set of entities that we suspect are vaguely related to the bug report we want to process. Phase 4 is where we begin making conclusions on the data. The software chooses between two algorithms for determining the score that will indicate the faulty files. If the amount of files associated with the bug are less than 15% of the total system files, we choose the simpler algorithm and find the score by determining which files have the most outward connections to the codebase. If the bug effects more than 15% of total files, we generate clusters of related entities using our relationship map produced in phase 2 and then for each cluster we check how connected it is to the most important files related to our bug. The cluster that is most connected has the highest chance of containing the bug.

The original system consisted of a collection of scripts written in Java, Python and Shell. The scripts were fairly distributed and had very little documentation. Most of the work was done in intermediate steps and did not fit well together as a system because too much user intervention was required between steps.



Development Objectives *(0.75 page max)*

Develop a complete system (O1)

Since they are so many individual parts to the system required to run the system, we wanted to combine scripts and Java code into an easy to access complete system. Considering there were still some Python scripts we did not get access to by the end of the project, it would be very beneficial to have a complete system that is able to handle new codebases and improve with new bug reports.

Be able to integrate with a GitHub based workflow (O2)

We saw a great opportunity to integrate this system into GitHub's system. Initially, the system was set up to read reports from BugZilla, but we wanted to add the ability to integrate the system with GitHib's issue tracker. This would entail fetching new issues as they are created in a repo and processing them, pushing results back to the issue page so the developers can get a head start, and allowing users to authenticate using GitHub.

Operate as a standalone service with a UI (O3)

The system initially was just run through a terminal, and because there aren't too many options needed from the user, it would be beneficial to create a system that can work independantly with an easy to use interface. This would Work in a similar fashion to other CI tools, such as Travis. We would also plan to give users the control to hook into any of their repositories

System Requirements *(2 pages max)*

id 1

In order to complete O1, we needed to have access to all parts of the system which we were unfortunately unable to receive before the end of the project. The implications of creating a complete system would mean that the computer running the system would have to be powerful enough to process all parts of the analysis within a reasonable amount of time in order to be of use to the programmer. Due to the heavy amount of processing required to complete the fault analysis against a large amount of other reports, the average user's computer that runs the system would have to be able to handle repeatedly running the analysis.

id 2

Completing O2 required us to have enough parts of the system to be able to connect to GitHub using their API and get issues from their tracker. Since the biggest change to the existing system is the report data used, the requirements of running the system are similar to id 1. The biggest difference being that the resolution of the fault analysis isn't required as immediately as in id 1 because of the low frequency of new issues that are created in most projects.

id 3

Development Strategy *(2 pages max)*

Since this project had a lot of the main functionality already implemented, we did not have many choices over what development tools and languages to use. The majority of the project was written in Java, with some separate Python scripts used for data manipulation at some of the stages in the system. Including the development tools previously mentioned, the following were also used in the system:

Technologies

- Kotlin
- Python
- Java
- Javalin
- MongoDB
- Github Webhooks

Tools

- Intellij Idea
- NeoVim
- Robo3T
- ngrok

Datasets

- BugZilla reports

Results *(10 pages max)*

BugLocalization Project

- Restructured and cleaned up codebase
- Reduced very large codebase to 10 files
- Documented and made more readable for easier integration

API

- Able to monitor a GitHub repo for events
- Detect new issues and add into a Mongo database
- Able to start and end deployments on Pull Requests

Discussion *(1.5 pages max)*

Conclusions *(1 page max)*

Late Start

- Second iteration of project
- Had to re-purpose project after second milestone

Communication problems

- Lack of communication from the supervisor for long periods of time

Future Work and Lessons Learnt *(1 page max)*

Future work

- Develop software to process issues incrementally into the BugLocalization
- Integrate the BugLocalization project as part of the API and utilize its full power
- Make a web UI similar to Travis to display results

Lessons learnt

- Develop consistent communication plan with stakeholders