Project Report 3

Fault Localization using NLP on Bug Reports

with Kostas Kontogiannis

Gurpreet Singh & Paul Bartlett

Software Developers

Contents

Faulty

Project description

The focus will be to design and develop a system that is capable of processing bug reports and extracting useful information about them, and then using that information to provide the developers useful insight into where the bug may be within a large code base. The goal is to reduce the amount of time a developer will need to reach the correct bug after initially reviewing the bug report.

The system will be developed in such a way that it is easily integrated into a continuous delivery pipeline. The project can be divided into three distinct components.

Data Processing

In order for the entire system to work correctly the core essential data processing and analysis has to be effective in detecting the errors. Therefore, the first step is developing a system that can use NLP to process all the bug reports associated with a project to come up with a list of keywords and process the code base to determine a map of relationships between function calls. This code has now been written and is mostly within our project.

User Interface

The next step in delivering the system to a real user, is developing a front end where a user can input a repository for the system to begin processing. Ideal operation of this tool would occur like other DevOps pipeline tools such as travis-ci.org where a user can link a repository they own and the tool can push it's results back into the bug report for developers to see. There will not be too many interactions available on the front end other than viewing the results of the system and picking new repositories. This portion of the project is not yet completed. Refer to deviations section to understand the reasoning for putting the interface as the last priority.

Runtime Processing

Since the front end will be making REST API calls to Github repositories, and we need a way to persist processing while providing consistent feedback to users, there needs to be a backend API service allowing those operations to occur. Another task this portion will be responsible for is

handling the flow of information when a new bug report appears. The backend will be responsible for detecting this, starting a new processing task, and posting a "Fault Report" back into the bug discussion. A substantial amount of work has been done on this portion of the project. The backend is able to communicate with Github and catch issues posted by users.

Revised Roles

Gurpreet Singh

- Lead Architect / Back-End developer
- Documenter

Paul Bartlett

- Lead Front-End developer
- Lead Tester & Quality Controller

Revised System Requirements

Section A : Data Processing

- **Feature 1:** Able to generate entity relationship rsf from codebase
  - FR 1: Pass code through cdif2rsf to generate rsf
  - FR 2: Clean up incorrect entity and relationships
  - FR 3: Store in accessible data storage for next step to use
- **Feature 2:** Able to generate set of keywords from bug description
  - FR 1: Compare each token to codebase to find valid functions
  - FR 2: Expand initial token set by a factor of 3
  - FR 3: Use NLP to determine question context
- **Feature 3:** Able to combine keywords and rsf into ranked outcomes
  - FR 1: Run LSI on each token and generate search space for each

  - FR 2: Expand the search space for each result in FR 1
  - FR 3: Find similarities between the initial token expansion and the final set of tokens
  - FR 4: Apply ranking equation from research paper to come up with final outcome

Section B : Front-End User Interface

- **Feature 4:** User is able to scan and mark a new repository for processing
  - FR 1: Scan user's Github repos using Github's API
  - FR 2: Allow the user to select ones they wish to run processing on
  - FR 3: Remember which ones the user selected by storing on backend
- **Feature 5:** User is able to view the results of a new bug report's processing
  - FR 1: Monitor output from backend endpoints showing new results for user's repos
  - FR 2: When a new bug report is created, and the processing finishes, show the output of that processing on a separate page
  - FR 3: Allow the user to rerun processing on a specific bug report by sending a request to backend
- **Feature 6:** User is able to login using their Github credentials
  - FR 1: On first usage redirect user to Github's App authentication page
  - FR 2: Ask backend to associate bug reports and repositories with this user
  - FR 3: Redirect user to main UI

Section C : Back-End Runtime Processing

- **Feature 7:** Support front-end operations
  - FR 1: Allow registration using Github Auth
  - FR 2: Allow retrieval of processing results for each bug report
  - FR 3: Support re-running processing on a specific bug report
  - FR 4: Create a API where the UI can fetch everything from
- **Feature 8:** Manage the automation of Data Processing (F1, F2, and F3)

  - FR 1: Automate RSF generation when a new repository is linked
  - FR 2: Automate keyword generation when a new repository is linked
  - FR 3: Continuously improve and modify RSF and keywords as code/bugs change

- **Feature 9:** Handle processing and evaluation when a new bug report comes in
  - FR 1: Monitor marked repositories for each registered user and trigger when new issue is filed
  - FR 2: Run through automated ranking algorithm
  - FR 3: Store result for later retrieval
  - FR 4: Be able to connect to a repository and fetch an issue when it is posted

- **Feature 10:** Combine each element of data processing into the backend runtime
  - FR 1: Combine entity generation into token generation
  - FR 2: Combine FR1 with RSF processing all into one unit
  - FR 3: Move all functionality to an exposed part of the API
  - FR 4: Create all endpoints for data processing functionality

Project Actual and Deviations

Refer to the attached spreadsheet to see the current progress on the project. Some deviations we made from the original plan during this iteration include changing the priority of some features and spending more time on parts of the project we thought would be easy. One of the causes for this was the lack of code organization and commenting present in the research project we are working with. This lead to a lot of reformatting which wasn't terrible but still consumed time we would have liked to spend elsewhere. We were also relying on receiving an additional research code base to reference but we were unable to communicate with the people in charge of the research.

We changed the priority of the frontend features and made them last priority. This was done because we decided that having the functional portion of the code working is much more important than focusing on the looks.

List of features that have been implemented

Feature 1

- Requirement 1
- Requirement 2

Feature 2

- Requirement 1
- Requirement 2

Feature 3

- Requirement 1
- Requirement 2
- Requirement 3
- Requirement 4

Feature 7

- Requirement 1
- Requirement 2
- Requirement 4

Feature 9

- Requirement 1
- Requirement 4

Feature 10

- Requirement 4

Future implementation plan to completion

The next step for this project is putting everything together. By the next milestone we want to atleast be able to completely aggregate issues from a Github repo and produce results to the backend. When this process is reliable and consistent we will create a minimal web interface to display the results.

The next major problem we will be running into is combining multiple parts of the data processing section (features 1, 2, 3) into the runtime section (features 7, 8, 9). Currently they are operating separately of each other. We have created a new feature (10) to handle work related to this task.