Student Internship Programme (SIP)
Final Project Report

at

NUS-OSS

Reporting Period:
05 2022 to 08 2022

by
Goh Yee Chong, Gabriel

Department of Computer Science
School of Computing
National University of Singapore

2022/2023

Project Title: Software Engineering Intern at NUS-OSS Projects
Project ID: SY212142954
Project Supervisor: Dr. Damith Chatura Rajapakse

Summary

WATcher, a software application to monitor software projects in a visual way, has been developed. The application builds on the existing code base of CATcher, a software application for the anonymous peer testing of software products, and is not mission critical. The application gives developers more freedom to add features while gaining expertise that is relevant for maintaining CATcher at the same time. The application addresses the difficulty of adding user-visible features to CATcher itself due to CATcher's mission-critical nature which hinders new developers from exploring and learning freely. The current application consists of two parts, Issue Dashboard and Activity Dashboard.

Subject Descriptors:

- Software and its engineering~Software creation and management~Collaboration in software development~Open source model
- Software and its engineering~Software organization and properties~Software system structures~Abstraction, modeling and modularity
- Software and its engineering~Software notations and tools~Context specific languages~API languages

Keywords:

Angular, Typescript, Graphql, rxjs, Github REST API

Acknowledgements

Acknowledgements to NUS-OSS and Associate Professor Dr. Damith Chatura Rajapakse.

Acknowledgements to CATcher developers, Low Jun Kai Sean, Ding Yuchen, Lee Chun Wei, Lee Xiong Jie Isaac, and Kang Su Min.

Table of Contents

1. Introduction

1.1 Background

National University of Singapore's Open Source Software (NUS-OSS) is an initiative that helps NUS School of Computing students gain experience in software engineering through working on Open Source Software. These projects are being deployed and used by users. Students get the opportunity to work on these projects when they take modules CS3281/82 Thematic Systems Project, CP3108A/8B Independent Work Modules, CS2103R Software Engineering Research, CP3200 SIP, CP3880 ATAP, CP4101 FYP and CP4106 Computing Project.

1.2 Overview

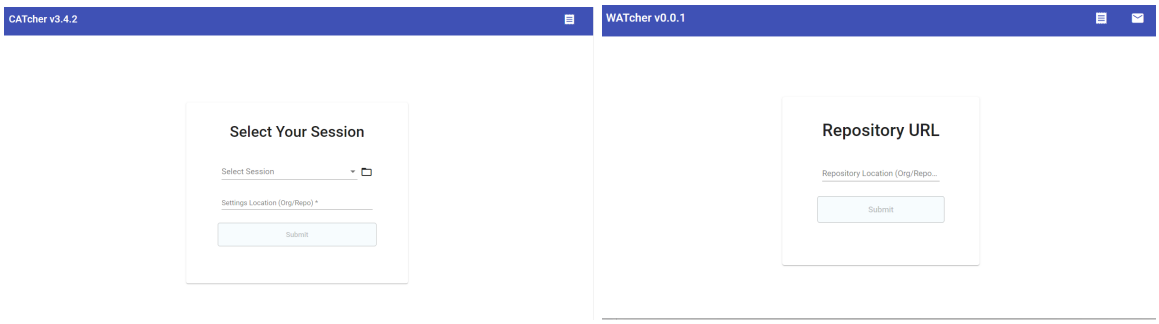
One of the NUS-OSS projects is CATcher, a software application for the anonymous peer testing of software products. However, due to the mission critical nature of CATcher, features must be kept minimal to minimize regression risk and maintenance burden. (Rajapakse, 2022). This nature makes it difficult for new developers to push changes to CATcher. There are thus fewer opportunities for new CATcher devs to add features, and get satisfaction from implementing features visible to the user. To address the above issue, a sister application that is not mission critical but builds on the CATcher code base is developed. WATcher is a software application to monitor projects in a visual way. WATcher also gives new CATcher developers more freedom to add features while gaining expertise that is relevant to maintaining CATcher at the same time. (Rajapakse, 2022).

2. Deliverables

2.1 WATcher Landing Page

Figure 2.1.1

Comparison of CATcher and WATcher landing page



Note. This figure demonstrates the visual changes made to the existing CATcher landing page. Select Your Session with a Multiple Choice Selection has been changed to an open-ended Repository URL input textbox.

CATcher uses Github repository as storage for session settings. An example of this is CATcher/public_data repository. The repository stores user information and open phases. However, WATcher did not require different levels of user permissions and open phases, since WATcher is essentially different in purpose. With the idea that WATcher should be able to visualize repositories without external storage, I modified the authorization process to accept only the URL of the repository to be viewed. WATcher also uses local storage to keep track of the repository that is entered in, and can easily be expanded to use multiple repositories for future features.

CATcher stores only the repository information in local storage, reading the settings file in an external Github repository to know which phases are open. In CATcher, a phase referred to one of the phases of peer testing in Peer Evaluation. For instance in CS2103, there is Phase Bug Reporting, Phase Dev Response and Phase Tutor Response. These phases are opened in different periods of the semester in sequence to facilitate the Peer Evaluation process.

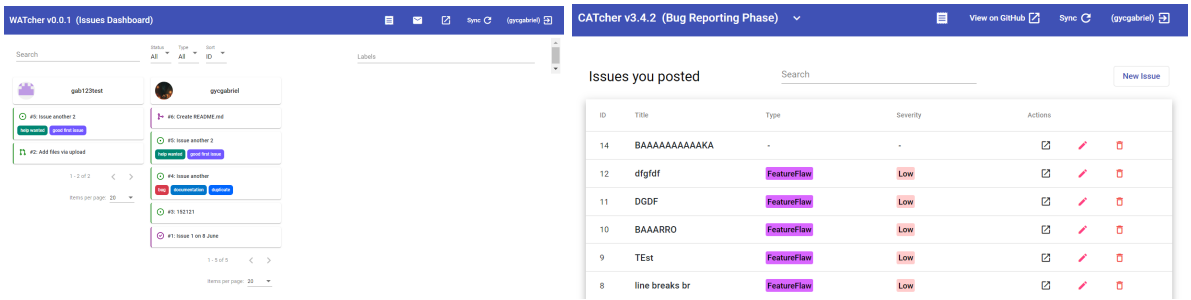
In WATcher, there is no concept of phases, since WATcher is not used in the Peer Evaluation process. Instead, I have adapted the concept of Phase, into a Feature of WATcher. Each Phase represents a different feature, or visualization dashboard of WATcher to view information about a repository.

Additionally, in WATcher, I have changed the structure of SessionData, in which a SessionData comprises a SessionRepo, a Session repository type given the phase and an array of repositories relevant to it. This new type comprising a phase and an array of repositories allows future features of WATcher to include more than a single repository, for instance for features that compare across various repositories. One instance of SessionData comprises an array of SessionRepo, that is, multiple of these phases and their repositories. This type allows new phases of WATcher to be added with ease. By not hard-coding the phases, it becomes possible to specify the phases in a single file, for A Single Source of Truth, and populate SessionData by iterating through the phases, This change in architecture is possible in WATcher because unlike CATcher, WATcher does not have a certain set number of phases that must occur. Instead, there are infinitely many new dashboards that can be added in the future by new developers.

2.2 WATcher Github Issue Dashboard

Figure 2.2.1

Comparison of CATcher Bug Reporting Phase and WATcher Issue Dashboard



Note. This figure demonstrates the modifications made to the Bug Reporting Phase to adapt it to the WATcher Issue Dashboard. The table of issues has been modified into a more visual-friendly list of cards, separated into columns by users.

CATcher uses Github Issues in storing bug reports that peer testers make in the Bug Reporting Phase. It thus needs to show a list of bug reports that the peer tester has already made, by fetching the issues from a Github bug reporting repository on the user's account.

This feature is similar to the basic purpose of what WATcher is intended to do. WATcher's purpose is to monitor the progress of software engineering projects. Part of that is monitoring the Github Issues assigned to each user and the Github Pull requests created by each user. By knowing which issues and

how many Github Issues each user is assigned to, and how many Github Pull requests have been created and merged by each user, project mentors or teaching staff will be able to tell how much work is being done, and by whom. Thus, there is a need for a WATcher Issue Dashboard.

I adapted relevant CATcher code used in fetching and displaying Github issues, to be used in the WATcher Issue Dashboard, porting over useful features already existing in CATcher. For instance, the search bar implemented in CATcher, to search through issues, the GithubService and existing code infrastructure to fetch the required information from Github through Github APIs. To adapt these codes to my needs, required an in-depth understanding of existing code, and only through that could I understand what I needed to remove, change or leave as-is.

2.2.1 Separate Columns by User

In contrast to CATcher, WATcher needed to differentiate clearly which issues and pull requests are done by whom. To create separate columns, I needed to duplicate the number of tables by the number of members to the Github project. WATcher does this through fetching a list of possible Assignees to the Github repository, that is, members that can make changes to project code.

WATcher fetches all issues to the repository and stores it in Issue Service, before each table filters the issues to display to their corresponding user. An alternative I considered was to send multiple Github API calls to fetch issues related to the corresponding user, but it would be less efficient due to sending multiple calls.

2.2.2 Fetch Pull Requests

Since CATcher bug reports are stored in the Github Issues of a Github repository, there was no need to fetch the Github Pull requests of the repository. For WATcher, however, Github Pull requests are essential and indicative of the actual work done by each member of the project. Through the Github Graphql query, the Github Service in WATcher has been modified to fetch Github Pull requests in addition to Issues. However, due to the complexities of Graphql cursor pagination, I have opted to fetch issues and pull requests separately. The results of both of the API calls are merged into a single data stream of Github Issues, which comprises both Issues and Pull requests. This typing is synonymous to Github's own type definition, that Github Pull requests are also Github Issues.

2.2.3 Fetch Milestone data

Github Milestones are used in Github to mark important versions of a project and comprises Github Pull requests and Github Issues that reach that Milestone. Utilizing Github REST API, WATcher fetches a list of Milestones to provide as checkboxes in Filter by Milestone. I have also modified the Graphql query in fetching Github Issues and Pull requests to include fetching the assigned Milestone, if any.

2.2.4 Filter by Status, Type, Milestone, Labels

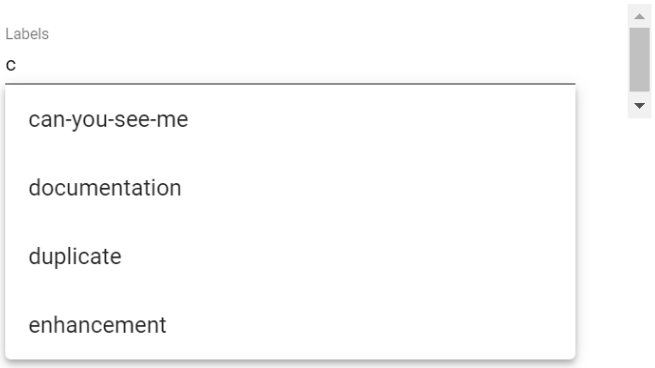
The typical software engineering project in CS2103 has more than a hundred issues and pull requests each. To filter and display the relevant ones that the WATcher user is interested in viewing, I implemented a filter feature. Currently supported filters are Status (Open, Closed or All), Type (Issue, Pull requests or All), Filter by Milestone and Filter by Labels.

The former two filters are comparatively easy to implement, by checking each issue or pull request for a particular value.

The latter two filters required fetching more data. Specifically, for Milestone, I had to create Milestone Service and related code infrastructure to fetch and store as variables the milestones from Github. The list of milestones of a project is needed to provide a checkbox dropdown, for WATcher users to select which milestones they wish to filter by.

Figure 2.2.2

Filter by Label dropdown



For Labels, I needed to adapt CATcher's Label Service to fetch Labels instead of setting a set of labels. CATcher had a set of labels hard-coded and syncs with the Github Bug Reporting repository to populate these hard-coded labels. WATcher does away with these labels, and simply fetches the existing labels from Github instead. The labels fetched are displayed as dropdown suggestions as the user types.

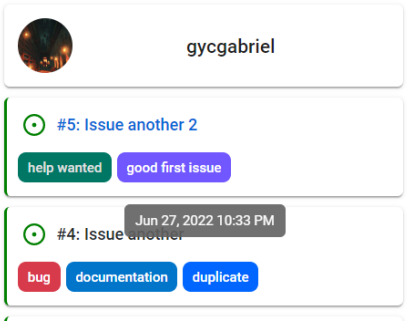
2.2.5 Sort by ID, Title, Date Updated

WATcher provides the option for sorting of Github Issues. The implementation involved the Sort Header of Angular Material. Ordinarily used for sorting columns of a single table, I have adapted a single instance of MatSort to be used to sort all user columns at once.

2.2.6 Cards View instead of Table

Figure 2.2.3

Card View of Issue Dashboard



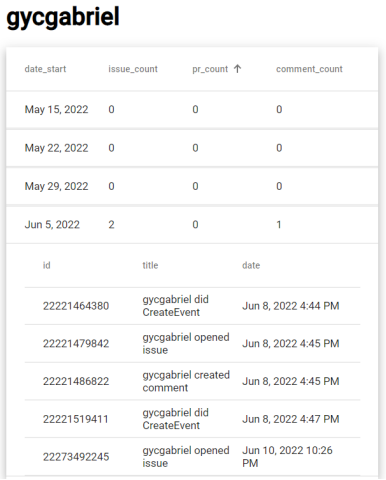
In the early stages of prototyping, the Issue Dashboard had split multiple tables side by side to display the issue for each user. This table view was not visually pleasing as the tables were overlaid on top of the other by the next. There was insufficient horizontal space to fit more information aside from the title of the issue.

To remedy the problem of scarce horizontal space, WATcher displays the issues in a list of cards, with a column for each user. The cards have also been customized with Github-relevant icons to show if the issue is open or closed, or if the pull request is open or merged. The use of cards and icons provide a visual appeal to WATcher.

2.3 WATcher Github Activity Dashboard

Figure 2.3.1

A prototype of WATcher's Github Activity Dashboard



A feature of WATcher that I have implemented but not released, is the Github Activity Dashboard. WATcher's Github Activity Dashboard displays Github events of the repository between a selected start and end date. Github events comprise every user's interaction with the repository, including issue creation, pull request creation, and comments on issues and pull requests. This dashboard supports the purpose of WATcher by monitoring the activity of repositories,

2.3.1 Data Aggregation for number of Issues, Pull requests and Comments

In order to display the number of issues, pull requests and comments week by week, I had to manipulate GithubEvents with a single date, into an EventWeek, a collection of GithubEvents that occured in a week, and aggregate the number of Issues, Pull requests and Comments respectively. The data stream of EventWeek is then displayed on the Activity Dashboard.

While it currently displays tables side by side as in the initial prototype of Issue Dashboard, it is to be changed to display as cards, similar to the Issue Dashboard.

3. Conclusion

3.1 Problems and Challenges

3.1.1 Difficult to understand existing code

It was difficult to understand code written by other NUS-OSS developers that had been contributed a long time ago. I overcame this by going through the code systematically and learning through Angular tutorials.

3.1.2 Modify to my needs

To modify an existing application to my needs, I needed to trace the code to find out which part of the code does what, and which part of the code I needed to change. It was more difficult to trace in a modular project such as Angular.

3.1.3 Learn API calls and query syntax

Without prior experience in Github REST API and Graphql, it took time and effort to learn and apply them to existing API calls. I needed to know what information I could fetch through API calls first. For e.g., there was no easy way to fetch Github events through Graphql, which I only found out after extensive searching and googling. I had to use the Github REST API instead.

3.2 Reflection

This experience gives me a clearer picture of what it may be like to work as a software engineer. It has exposed me to the struggles and rewards of software engineering. It is something for me to consider in my future career. In the process of asking fellow developers for help, it may have helped to build a bit of a network in the form of fellow students.

4. References

Angular Documentation. (n.d.). Retrieved July 28, 2022, from https://angular.io/docs

CATcher-org. (n.d.). *A sister app for catcher · discussion #938 · catcher-org/catcher*. GitHub. Retrieved July 28, 2022, from https://github.com/CATcher-org/CATcher/discussions/938

GitHub Documentation. GitHub Docs. (n.d.). Retrieved July 28, 2022, from https://docs.github.com/

Introduction to graphql. GraphQL. (n.d.). Retrieved July 28, 2022, from https://graphql.org/learn/

OSS projects at NUS SOC. NUS. (n.d.). Retrieved July 28, 2022, from https://nus-oss.github.io/

RxJS Documentation. (n.d.). Retrieved July 28, 2022, from https://rxjs.dev/guide/