Case study on how the role and module-ownership of a developer changes over time in a software project?

Nandan Parikh
Department of Computer Science
University of California - Los Angeles
Los Angeles, CA 90025
`nandanparikh@cs.ucla.edu`

Prateek Malhotra
Department of Computer Science
University of California - Los Angeles
Los Angeles, CA 90024
`prateekmalhotra@cs.ucla.edu`

Tanmay Chinchore
Department of Computer Science
University of California, Los Angeles
Los Angeles, CA 90024
`tanmayrc@cs.ucla.edu`

Abstract

Code ownership is more than just assigning subsystems to individuals or individuals to sub-teams. It varies over the lifetime of a system depending on the software's developing needs [7]. Within this context, there have been multiple recent studies on code ownership in open-source large-scale software systems [3] and relations to the development cycle in big OSS (Open source software) systems like the Apache server [6]. Furthermore, this idea was extended[2] to understand the correlation between software quality and code ownership (taking into account the varying degree of contributions made by different developers) in Windows 7 and Windows Vista. In this work, we build upon existing work [2] to study how the dependency and role of a developer changes in the software development cycle by using commit-data from multiple OSS github repositories. We believe that the module-developer dependencies that we identify are useful in the industry to assess the role and impact of a a particular developer across time.

1 Problem Statement

There have been several studies on developer productivity and analyzing the commits of software developers to rank them and the quality of their code[8]. There are also studies on the developer to be selected for a specific bug assignment [1]. There are studies on the types of commits each developer makes and hence finding the ownership of project modules [2]. We propose to work on the following questions in our project. How does the role of a developer change while working on a project? Does the dependency of an individual increase over time. If an individual starts with a specific module in a project, does his/her reach increase in the code-base? Or does the developer only work on the same module over time. After studying this across databases, we try to check what similarities and patterns are found in various projects. We also try to study and reason the bugs with the kind of ownership in the modules. We try to show a visualization with a time frame across developers and across projects to get a clear idea of how the trends are. Studying these patterns can be useful in the industry as there are a number of developers who leave and join the team. If there is strong ownership for modules, these metrics can help managers to check the dependency on modules with developers.

2 Motivation

Developers change their role as the software ages and new features are added [7]. Due to this reason, it becomes increasingly important to assess the dependency of a module on a particular developer. The recent work of [2] studied Windows 7 and Windows Vista and how they have evolved over time - specifically, they identified a strong correlation between pre and post release bugs in the system and laid down different policies that a company can use to mitigate this problem. A major one being that the managers should be making sure that all changes are reviewed by a major contributor before they are accepted.

In this project, we try and identify the change in the span of a developer over time - are they contributing to more modules over time or are they sticking to the same set of functionalities? Another goal here is to try and assess the "dependence" of a module on a particular developer. This is a measure of the impact that a particular developer will have if they decide to leave the team. Project managers can use this information to make sure that modules are not completely dependent on particular individuals as their absence can potentially halt progress. This can also be used to assign responsibilities wisely to reduce future complications arising from the addition of new features and bug-fixes.

3 Related Work

Don't Touch My Code! Examining the Effects of Ownership on Software Quality: This paper [2] explores the relation between different code ownership measures and software failures in industrial software projects. The authors ask if high ownership is associated with lesser defects, if having a bunch of low ownership developers on a software entity affects it negatively and if code ownership affects the overall development process followed. The authors find a relation between measures of ownership and the proportion of ownership for the primary owner with pre-release faults and post-release failures. This paper is useful for our work as it provides us with the necessary metrics for measuring ownership of a developer throughout a software project.

Code Ownership in Open-Source Software: This paper [3] discusses metrics that measure how workload of software modules is shared among developers. The workload is indicative of software quality. The authors perform this experiment on an Open Source System(OSS) instead of an industrial software project. The authors explore the relation between ownership metrics and fault-proneness of multiple open source projects. The authors conclude that the lack of correlation between ownership metrics and module faults is due to the distributions of contributions among developers and the presence of “heroes” in open source projects. This paper relates to our work directly as we are also exploring open source systems to understand how the role of a developer changes over time.

Managing Code Ownership: This paper [7] discusses the definition of Code Ownership and how it affects the overall development of the software suite. The authors discusses different models of code ownership and even the concept of non-ownership. The different models of code ownership include product specialist, subsystem ownership, chief architect and collective ownership. The paper discusses when a particular model should be used and if it is possible to scale code ownership during the software development life cycle if required. This paper is helpful as we are looking into how a developer can change his model of code ownership mid-project.

Who should fix this bug? [1] Their approach applies a machine learning algorithm to the open bug repository to learn the kinds of reports each developer re-solves. This depends on the types of modules a person has worked upon. When a new report arrives, the classifier produced by the machine learning technique suggests a small number of developers suitable to resolve the report. This is the area we try to focus, we feel that in most cases the developers are stuck on specific parts of a module they already created.

4 Dataset

For our project, we are looking at experimenting on the following datasets and repositories:

AndroidTimeMachine: AndroidTimeMachine [4] is a graph-based dataset of commit history of real-world Android applications. It provides a dataset of 8,431 real world open source Android applications. It offers both the source code and commit history information, including several bug fixes.

Defects4J: Defects4J [5] is a collection of bugs and the commits which fix each of them. This is a smaller dataset. We can analyze the source code of each of the (open-source) projects corresponding to bugs in the dataset, and check who solved the bugs. This will give us a better idea about code owenership and its progression with time.

5 Approach

We will work on data from open-source projects and their commit history from github. We propose to model a context flow graph of the code-base and check the modules that the developer works on. We measure this after specific time intervals for each developer to get an idea of what the developer is working on. From this we try to infer the dependency of each module with respect to a developer. Using these metrics, we will then visualize the trends across time and attempt to answer the question of developer reach and whether there are similar results across various projects.

6 Github Link

We will use the following repository for collaboration https://github.com/nandanparikh/CS230

References

[1] John Anvik, Lyndon Hiew, and Gail C Murphy. Who should fix this bug? In *Proceedings of the 28th international conference on Software engineering*, pages 361–370. ACM, 2006.

[2] Christian Bird, Nachiappan Nagappan, Brendan Murphy, Harald Gall, and Premkumar Devanbu. Don't touch my code!: examining the effects of ownership on software quality. In *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering*, pages 4–14. ACM, 2011.

[3] Matthieu Foucault, Jean-Rémy Falleri, and Xavier Blanc. Code ownership in open-source software. In *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering*, page 39. ACM, 2014.

[4] Franz-Xaver Geiger, Ivano Malavolta, Luca Pascarella, Fabio Palomba, Dario Di Nucci, and Alberto Bacchelli. A graph-based dataset of commit history of real-world android apps. In *Proceedings of the 15th International Conference on Mining Software Repositories*, pages 30–33. ACM, 2018.

[5] René Just, Darioush Jalali, and Michael D Ernst. Defects4j: A database of existing faults to enable controlled testing studies for java programs. In *Proceedings of the 2014 International Symposium on Software Testing and Analysis*, pages 437–440. ACM, 2014.

[6] Audris Mockus, Roy T Fielding, and James Herbsleb. A case study of open source software development: the apache server. In *Proceedings of the 22nd international conference on Software engineering*, pages 263–272. Acm, 2000.

[7] Martin E Nordberg. Managing code ownership. *IEEE software*, 20(2):26–33, 2003.

[8] Minghui Zhou and Audris Mockus. Developer fluency: Achieving true mastery in software projects. In *Proceedings of the eighteenth ACM SIGSOFT international symposium on Foundations of software engineering*, pages 137–146. ACM, 2010.