Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

School of Computer Science and Statistics

Evidence based Guidance for Open Source Development

Sunit Deshpande
19302026

September 7, 2020

A Dissertation

Presented to the University of Dublin, Trinity College in partial fulfilment of the requirements for the degree of

Master of Science in Computer Science (Data Science)

Supervisor: Prof. Stephen Barrett

Declaration

I hereby declare that this project is entirely my own work and that it has not been submitted as an exercise for a degree at this or any other university.

I have read and I understand the plagiarism provisions in the General Regulations of the University Calendar for the current year, found at `http://www.tcd.ie/calendar`.

I have also completed the Online Tutorial on avoiding plagiarism 'Ready Steady Write', located at `http://tcd-ie.libguides.com/plagiarism/ready-steady-write`.

Signed: _____ Date: _____

Abstract

Some of the world most successful software products are an outcome of open-source software development. Many commercial software's also partially or fully depend on other open-source software. However, open source software development process is very different than traditional software development process. In open-source software development, the contributors are distributed globally over large distances from each other but they still manage to deliver high-quality software. Traditional research has been done to see what qualities of a software development team help them succeed in a corporate environment. But little research has been done to see which qualities help a team or an individual to succeed in an open-source software development community.

In this research five main principles are tackled to see how open-source software development differ in these principles from its commercial counterpart and traditional beliefs. Five principles that were tackled were Teamwork, Retention of contributors in the project, Ownership of the project, Organizational Hierarchy in an open-source project and common issues faced by open-source projects. For each of these principles, analysis has been done to understand how open-source development differs from traditional beliefs. Based on the learning from this analysis, this research tries to suggest a set of guidelines for an open-source project and a newcomer to an open-source project.

Acknowledgements

To begin, I would like first to express my sincere gratitude to my supervisor, Prof. Stephen Barrett for all his constant help, guidance and enthusiasm throughout this dissertation. His unwavering support ensured that I remained focused and motivated towards the completion of the thesis.

Besides, I would also like to thank the lecturers from the School of Computer Science and Statistics from whom I had the opportunity to learn throughout my studies in Trinity College Dublin.

Also, I would like to acknowledge my fellow students and friends whom I have enjoyed spending the past one-year learning along with me.

Finally, I would like to express my gratitude to my family, for their massive support, love and encouragement given throughout my life to achieve my dreams. Without them, none of this would be possible.

Sunit Deshpande

University of Dublin, Trinity College
September 2020

Contents

List of Figures

List of Tables

1 Introduction

In this chapter, the overall motivation for this dissertation is discussed in detail. Motivation is accompanied by the research question and research objective. The chapter concludes with an overview of the discussion, research contribution and the research challenges.

1.1 Motivation

Open Source Software (OSS) is a term used to describe a set of software which are free to use, modify and distribute by its user. The *"Open Source initiative"* was coined in the year of 1996 (1). From the time the open-source initiative has started the growth and success of the open-source software has been very prominent. Some of the software products that were outcome of the open-source software community are the Linux Kernel (2), Python programming language (3), PostgreSQL relational database (4) and many more. Open Source Software has had a major impact on the technology ecosystem than any commercial software ever has managed to achieve. One example which supports this argument is a survey conducted by Netcraft, regarding the web server software used in the year 2020 (5). The survey showed the number of websites on the internet using the open-source software called Nginx is far greater than a similar commercial solution offered by Microsoft. In fact, in the following year, the market share for Nginx on the web has increased but that of Microsoft has decreased.

Open Source Software (OSS) development is driven by a community of self-motivated and self-interested individuals. It is one of the most important strengths of OSS software. Each user brings in a new perspective and a new personality to solve the same problem. That is why attracting and nurturing newcomers and a new contributor to the open-source software is essential for the survival of the project. Research has shown that most of the failure in the open-source world was due to a shortage of people willing to contribute to the project and the important reason behind the success of a large open-source project is access to a larger talent pool (6).

1.1.1 Barrier for new contributors

Despite incoming flow of new contributors been so vital for the continuous improvement for the OSS community as a whole. New contributor usually feels that contributing to an OSS project is not welcoming or too difficult to start with. A plethora of research and literature is available where an effort has been made to understand what are the different types of barriers a newcomer faces in OSS development and also some remedies are suggested.

In his research (7), Ior has categorized the barriers faced by newcomers into Six different categories and provided a list of useful tips for newcomers as well as for the open-source project suggest how to make the onboarding process much easier in OSS. Similar research (8) by Sholler suggests 10 simple rule that a newcomer can follow to become a successful contributor.

Even though having literature available on how to become a good contributor and how to make the OSS project more approachable to a new user. Research like (9) and (10) has shown that there is a lack of new talent in the open-source world. And this is still an active issue in the OSS community. This following article (11) published in *"InfoWorld"* on 19th of August 2020 also describe the same issue. The article describes that there is a shortage of people who can contribute to the OSS projects.

In this dissertation, the author has tried to analyze how traditional software engineering principles and methodology apply to open-source software projects. And are there any set of guidelines that an open-source project and a newcomer to an open-source project can follow to maximize the overall efficiency of the project.

1.2 Research Question

Based on the research presented in the Motivation Section 1.1. This dissertation tries to answer the following research questions.

RQ1: Do traditional software development methodology and principles work for Open Source Software Development?

RQ2: What are some guidelines that a project and a newcomer can follow in the Open Source world?

1.3 Research Objective

To satisfy the above stated Research Question 1.2, this dissertation tries to satisfy the following research objectives.

- To understand the software development methodology for Open Source Software work with respect to newcomers and top contributors.
- Recommend a list of guidelines for open-source projects and newcomers to open-source projects.

1.4 Dissertation Overview

This dissertation is divided into five major chapters namely Teamwork (2), Retention of Contributors (3), Organizational Hierarchy (4), Ownership (5) and OSS Issues Analysis (6). In the first four chapters analysis has been done to see how some traditional software development methodology and techniques apply for open source software development model. And the chapter OSS Issues Analysis (6) provides insists on what are the most common types of issues that someone faces when dealing with open-source software.

Teamwork chapter (2), starts with discussing the state of the art research which says that teamwork is the key to the success of any software project. Then it presents an initial analysis done over a set of open-source projects. The result from the initial analysis lets the author to believe that Open Source Projects are not operated by a huge team of software developers. It is operated by a small group of individuals.

Retention of Contributors chapter (3), tackles the traditional convention about retention where it is originally believed that retention of contributors over a longer during in a project leads to a successful software. An analysis is presented to validate how this convention holds for open-source projects. The result of the analysis shows that the top contributors in open source project change over time and that the top contributor don't stick to one project. There interest changes over time.

In chapter (4), Organizational Hierarchy, literature is discussed which states that there is no organisational structure in open-source project development. But the initial analysis of data from multiple open source projects has suggested that there are two types of contributor in open-source projects. Firs one being the code contributors and the second one been the admins. Code contributors are the one who do the majority of work for the project like adding new features, fixing bugs etc. They are the one who write most of the code. Admins are the contributor who mainly concentrates on helping the community by replying to questions, closing pull requests submitting issues and much more.

Chapter (5), Ownership, discuss the state of the art which shows that in the open-source project there is no allocation of responsibilities. Everyone contributes to modules that they feel like contributing. But the initial analysis has shown that in open-source projects contributors tend to focus on a particular module and not on the entire project as a whole. This has led the author to believe that the top contributors in open-source projects focus on

a particular module of their interest than the overall project.

In the chapter (6), OSS Issues Analysis, analysis of the most common issues in open source projects is done. The primary analysis of issues from multiple open source projects has shown that the most common issue faced in an open-source project is that of setting up a proper workstation for development. The chapter discusses the analysis process and the result is much more detail.

In the last chapter (8), Result, a set of guidelines are suggested for open source projects and newcomer to the open-source project. These guidelines are based on the analysis presented in the earlier chapter.

The dissertation concludes with a Conclusion, Chapter (9) and a discussion about future work concerning this dissertation.

1.5 Claimed Contribution

This dissertation tries to contribute a set of guidelines that an open-source project and a newcomer to an open-source project can follow. These guidelines were suggested based on analysis done of data from multiple open source projects.

Guidelines suggested for any open-source project are as follows.

- Focus more on attracting new talent then retaining old talent.
- Add documentation to the project especially targeted towards helping newcomers to get started.
- Make the process easier for newcomers to suggest/submit big and drastic changes. So as to attract new idea and keep the project innovating.
- Make it easy for newcomers to set up workstation by providing proper documents or by automating the process.

This dissertation suggests that there are two types of contributors to any open-source projects. First, one being the code contributors of the project who contributor the most of the source code to the project. And the second type of contributors is the one who does most of the admin tasks in the projects like closing a pull request, replying to issues etc. And the following are the guidelines suggested by this dissertation based on the type of contributor.

Guideline suggested for contributors who are more focus on maintaining the code base are as follows.

- Choose a project to contribute that motivates you and that help you learn new things.

- Don't to a project for too long.
- Focus on a single module of the project that you find interesting and worth learning.

Guideline suggested for contributors who are more interested in becoming an admin for a particular open-source project are as follows.

- Start with contributing to the code base of the project and then focus on helping the community.
- Focus more on code review, solving issues and admin task like managing pull requests.
- Focus more on introducing new talent to the project.

More detail about the guidelines and the data supporting those guidelines can be found in Result, Chapter 8.

1.6 Research Challenges

The nature of the problem brings some challenges that are needed to be solved before presenting the research. Some of those challenges are discussed in detail in the following section.

1.6.1 Research Scope

This dissertation tries to compare how traditional software development methodologies and practices related to the open-source software development process. There is a plethora of literature available on software development methodologies and practices. But given the scope of this dissertation and the time constraint evaluating all the principles is not a viable option. So the author of this dissertation has done a thorough literature review for articles, research papers, books and journals and has shortlisted four areas of software development methodology. These areas of the software development process are teamwork, retention of developers in a project, organizational hierarchy in open-source world and ownership of the project. These areas were chosen because the author of this dissertation felt that there are the most impactful and the most discussed in the existing literature. There might be a more important factor in the software development methodology that the author might have missed.

This dissertation tries to suggest a set guideline for a newcomer which would help him become a top contributor in an open-source project. But there is a lot of ambiguity of who is considered as a top contributor. The most common definition of a top contributor is one who has written the most number of lines of code. Another definition of a top contributor can be a person who started the project and has helped the project grow in the right

direction. Code is not the only factor that comes into the picture when we talk about the success of a project. Other forms of contribution that can be considered is someone who has actively helped other developers in the open-source community by maintaining the project documentation, helping in admin tasks like closing pull requests, helping other solve issues related to a project and things like that. Thus there can be multiple definitions of what a top contributor is for a particular project. Therefore for this dissertation, the definition of a top contributor is assumed to a person who has contributed to the growth of the project by contributing code to the project and helped in the administrative task by creating and closing issues and pull request for the project code repository.

1.6.2 Mining GitHub Data

Data required for this dissertation was mined using the GitHub API. There are two challenges when using the GitHub API as a data source. GitHub is a privately owned software product which contains millions of open source projects. To access the data for the repositories, GitHub provides access via REST API. But there is a fixed limitation on the size of data that can be mined from GitHub (12). To solve this the author designed a system which would do an incremental data mining on the public repository on GitHub and store the data in a relational database system. Using this approach the author was able to perform analysis for a larger set of data that would have not been possible using GitHub API alone.

Research by Kalliamvakou (13) has also shown that there is a chance that there would be a difference between the actual data and the data that is mined from GitHub. His research has shown that 40% of pull requests that were shown merged in the mined data whereas they were not merged on GitHub. Another important point that the research highlights is that data mined from GitHub also captures non-user related activities. These are the activities which are done by other software or site or scripts called as GitHub bots. But GitHub been the only source where most of the open source projects code base are hosted the author has decided to use GitHub as the data source and considered the above factors when evaluating the result.

2 Teamwork

Teamwork is considered to be one of the most influential factors in deciding the success of a software project. This chapter first presents a state of the art literature review discussing how teamwork is considered to affect a software project. Then chapter concludes with analysis and result showing how the principle that everyone in the team should contribute equally towards the project do not apply to open-source projects.

2.1 Literature Review

It is a very common understanding for any project software or otherwise, that teamwork is the most important factor that decides whether a project would survive and be completed in the given time constrain and budget. There is a plethora of articles and research paper available which examines how the teamwork and team coordination plays a role in success of software projects. Someone of this literature is presented in this section.

Success of a project can be defined in multiple ways. Completion of the project cannot be the only metric that must be taken into consideration when measuring the success of any software project. One of the most important factors that are commonly taken into consideration when estimating a project success is the project budget and the timeline under which the project was delivered. Work by Ralph Katz and Thomas J. Allen (14) has shown the relation between the projects performance and communication between the team. In their work the authors have suggested that the performance of the project increases over time and reaches its high in about 1.5 years for any project and then the project performance decreases over time. Authors have done study over multiple larger projects and has concluded that there is a positive correlation between the communication activity between the team and the performance of the project. Their study shows that communication in the team plays an important factor irrespective of the type of project and the age of the team members in the project.

In a similar perspective, work by Xiang and Wang (15) has investigated the reason for failure in multiple information system projects. In there study the authors invested three prominent information software development firms. The subject of there investigation was to identify

what factor contributes the most on a team performance which eventually leads to a project failure. The author suggests that four main factors that influence a team performance are quality of communication between the team, the ability to efficiently share knowledge between the team members, management support and clarity of the goal that the team is trying to achieve. Their research suggests that out of the four above stated factors, communication and knowledge sharing were the most defining factor which played a major role in dedicating the faith of the project. The authors in their work have also provided suggestions for building a better and outperforming team. This work also emphasizes that having a good team with constant communication is a key factor which influences the success of any software project.

The open-source software development process is a very different process than traditional software development process. In an open-source project, the contributors are distributed across the globe and each contributor to the project almost works autonomously. In such a scenario, efficient coordination between each contributor is a must. In the study by Pinto and J. K. Prescott (16) shed some light on how coordination between team member influence the project outcome. The authors suggest that the important factors that influence a successful team and thus a successful project is the accessibility of team member and formalized rule and procedures for sharing information between the team. In there, studies author has even presented direction for management as to how to build an effective team.

With similar perspective research by Sandra and Robert in their research (17) has stated that proper coordination among the team in a software project can lead to shorter project duration, better product quality and effective utilization of project budget. Their study identifies three main types of co-ordinations necessary for an efficient team performance namely process coordination, technical coordination and temporal coordination. In there study the author has also presented cases studies where lack of coordination has resulted in project failure.

Previous research has even tried to quantify the quality of teamwork in a project. Work by Martin Hoegl and Hans Georg Gemuenden (18) has presented a structural equation model called Teamwork Quality (TWQ) which can be used to measure the quality of teamwork in the project. The Teamwork Quality model uses six different facets of the team to access the quality of teamwork. Two of the important factors taken into consideration are the balance of member contributions and mutual support. The author suggests that the balance of members contributors make the team survive longer and helps in efficient project delivery. Another study by Hoegl and Gemuenden (19) has improved upon the proposed Teamwork Quality model and incorporated factors such as shared values and equal distribution of efforts into the model.

Thus various articles and research suggest that teamwork is a key factor for a project to be successful. Most importantly the equal distribution of labour and team coordination is considered to be the most influencing factor contributing to the success of the project. As discussed in the Motivation section of the Introduction chapter (1.1), the success of the open-source project is something that even commercial software find it difficult to achieve. So in the next section effort has been made to validate how the above-stated principles applies to various open-source projects.

2.2 Analysis

To analyse the teamwork principle for open-source projects, data from the following 5 open-source projects namely Systemd, Django, ReactJs, Deno and Spring Boot was gathered and analysed. And a primary analysis was done on the top 10 Python project from GitHub. More information about why these projects were selected and how data was gathered can be found in the Implementation chapter (7).

2.2.1 Distribution of Efforts

The first analysis that was done was to verify how the effort is distributed in an open-source project. To analysis, the effort of a particular individual in a project below the Effort Estimation algorithm was used, Algorithm (1).

To calculate the coding effort of each individual in the code base the following technique was used. For each commit done to the project, the difference in the code before and after the commit was calculated. Then based on the difference introduced by the commit, the commit was categorised as addition, maintenance, deletion or comments. Addition commits are commits where a new feature or new code was added to the code. In maintenance type commits, the old code was either replaced or modified. In deletion, type commit old code was removed completely and in comments type commit efforts were made to add comments or documentation to the existing code.

Based on the type of commit and efforts score was calculated and assigned to each individual contributor for a particular git repository. Then a percentage was calculated based on the total efforts that was done in a particular project over a fixed time. This allowed the author to rank the contributors based on the efforts that have put in the project.

2.3 Result

Based on the analysis done as explained in the previous section. The following result was observed (2.1).

Algorithm 1 Effort Estimation

```
Require: repo ← git repository
  for contributor ∈ repo.contributors do
    effort_score ← 0
    for commit ∈ contributor.commits do
      diff ← get_diff_of_commit(commit)

      diff_type ← UNKNOWN

      if diff contains new code then
        diff_type ← ADDITION
      else if diff refactors or replaces old code then
        diff_type ← MAINTENANCE
      else if diff only adds comments then
        diff_type ← COMMENTS
      else if diff removes old code then
        diff_type ← DELETION
      end if

      if diff_type is ADDITION then
        effort_score ← effort_score + 1
      else if diff_type ∈ MAINTENANCE, COMMENTS, DELETION then
        effort_score ← effort_score + 2
      end if
    end for
    add_effort_score_to_contributor(contributor, effort_score)
  end for
```

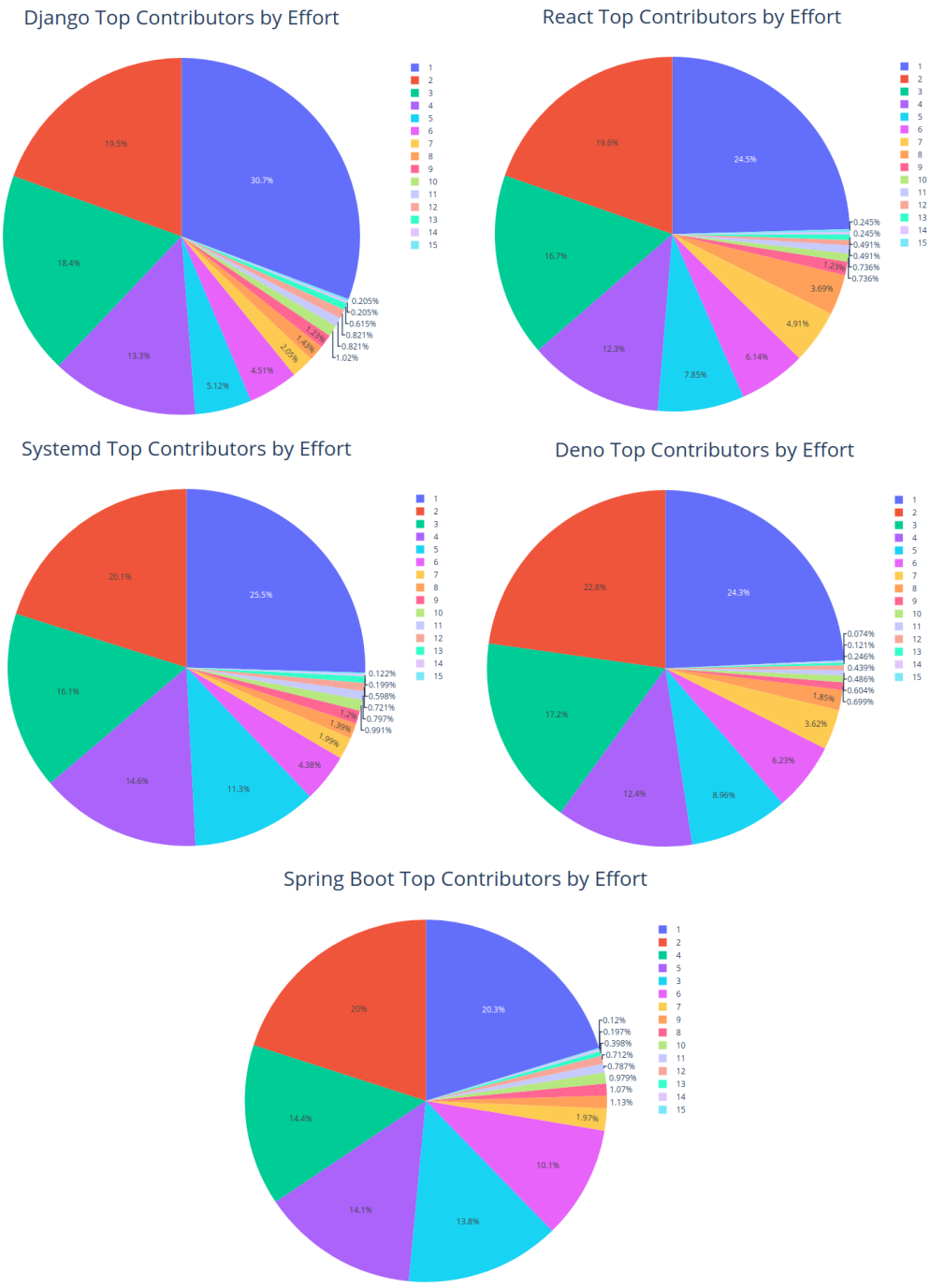


Figure 2.1: Efforts Distribution for project for the year 2019

Figure (2.1) shows the percentage of efforts done by the top 15 contributors in various projects over the year 2019. To plot the graph, each commit was assigned and effort score based on whether the type of commit was the addition of new feature or effort made in making the existing code much faster or effort made in adding comments to the code base. Then a cumulative sum of effort score was given each contributor based on the sum of effort score for each commit done by the respective contributor. This allowed the author to rank the developers based on the efforts that have put in the code base of the project. To make the result more presentable only top 15 contributors per projects were taken and a relative percentage was plotted based on the effort for each individual contributor and the total effort score for the top 15 contributors.

The literature review that was discussed in the previous section suggests that teamwork is the key to the success of a software project. And the distribution of efforts must be done equally to ensure the project success and survivability for a longer period of time. But the initial analysis of open source projects shows a completely different picture. As seen in Figure (2.1) it seems that not all contributors contribute equally to the project. From the initial data analysis, it shows that there are quite a few, almost top 5 to 10 contributors in an open-source project who are doing the majority of work.

To verify if this phenomenon is usually present in all open-source projects and not just happen to present in the few selected projects a very simple analysis was done. The top projects written Python (3) programing language were selected from the GitHub trending projects page. For each project effort score was calculated for each contributor. The total effort score was assigned to each contributor based on there commits in the projects. Then the developers who had effort score greater than or equal to top 30 percentile were filter and a count was taken for each project. The result of this analysis can be seen in the Effort distribution table (2.1). This primary analysis on a larger set of projects also strengthens the belief that in open source projects the majority of the work in done by a few contributors.

The detailed analysis of the 5 open-source projects and an initial analysis for top 10 Python open-source projects seems to validate the hypothesis that the majority of work is done by a small group of individual in the project. Literature that was reviewed in the earlier section suggests that team coordination and equal distribution of effort is key to the success and survival of a software project. But the initial analysis of data from open source projects suggests that the same principle might not apply to open-source projects. The analysis that was done to prove this is very basic and would require to be verified no a larger set of data. But the initial result gives a strong indication that the same phenomena would exist on most of the open-source projects. Thus the learning from this analysis can be summarised as follows, ***"Open Source Projects are not operated by a huge team of software developers. It is operated by a small group of individuals".***

Table 2.1: Effort Distribution Table

Project	No. of Contributor above 70%	Total No. of contributors
Flask	21	614
Photon	4	19
aiohttp	11	494
xonsh	10	215
ansible	102	5,000+
httpie	11	101
keras	33	864
requests	21	591
scrapy	21	383
sentry	15	473

Open Source Projects are not operated by a huge team of software developers. It is operated by a small group of individuals.

3 Retention of Contributors

Employees of an organization are one of the most valuable assets for a company. It is commonly assumed that the retention of an employee is crucial for the success and future of any software project and the same applies to open-source software projects. This chapter starts with a discussion on the literature which focuses on this principle and then analysis on open-source projects is done to validate how much retention is observed in open-source projects.

3.1 Literature Review

A lot of research has been done in understanding the motivation and techniques behind retaining the old contributors to a project for a longer duration. It is traditionally believed that as a contributor gains more experience in the project he becomes more productive on working with the project. One such research work is done by Minghui Zhou and Audris Mockus (20) where they have studied how the productivity of a software developer increases as time passes by. Authors suggest that for a smaller project it takes up to three months and for bigger projects it takes almost 12 months for a developer to attend the highest productivity level. In there, research authors have even presented a way to model the learning curve for a newcomer in the product. Their research suggests that senior members of a software developer team are the most valuable team member in the project and efforts must be made to retain them in the project for as long as possible.

Building upon the belief that older more senior contributor who has spent a lot of time working in the project and thus have a great knowledge of the project. A lot of research has been done in finding ways to motivate developers to stay longer in the project. Also, a lot of research has been made for identifying the type of newcomer in the project who has the highest potential of becoming a long time contributor. On such research is work done by Schilling and Laumer (21). Schilling evaluated newcomers in the KDE (22) project at Google Summer of Code (GSoC) (23). The aim of the research was to find a set of newcomers who has the highest potential of becoming a top contributor to the project. Analysis from the research shows that Person-Job Fit and Person-Role Fit are the best way

to model the retention quality in an open-source project. Research also showed that newcomer who spent a lot of time on the project and newcomer who don't feel that their abilities are underused have a high retention rate.

A similar study was done by Zhou and Mockus (24) where they tried to identify the newcomers in the project who would stay in the project for a longer duration. Their research showed that when newcomers were evaluated based on their retention potential, there was a high correlation between the project environment and the individual attitude. If the project environment and the nature of an individual are somewhat similar then there is a high chance that the newcomer would become a long time contributor to the project. Authors have even provided suggestion on how to identify and mentor newcomers so as to make them stay longer in the project. Their suggestions were based on nine factors observed by the authors when studying data from various software projects.

While the research above suggests that newcomer show a trade of long term contributor from the beginning of the project. Other research suggest that becoming a long term contributor is slow and gradual process. One such literature is the Legitimate Peripheral Participation (LPP) theory (25). The legitimate peripheral participation theory suggests that every newcomer to a project has an equal potential to become a top contributor to the project. The factors that define which newcomer will become are top contributor is mostly related to the duration spent by the newcomer in the project. Factors like the social interaction a newcomer has with his team member and the welcoming nature of the project play an important role in helping a newcomer transition into an experience long term developer. Work by Fang and Neufeld (26) has proved that Legitimate Peripheral Participation theory exists in open source projects. Work by Fang and Neufeld also suggests that apart from traditional factors motivation also plays an important role in open-source projects deciding which newcomer would stay longer in the project.

Similar kind of research was done by Qureshi and Fang (27) which shows the effect of socialization on the progress of newcomer to the path of becoming a top contributor. In this research, authors have suggested that the socialization of newcomer with existing top contributor plays a major role in deciding which newcomer would become a top contributor in the project. Authors have suggested there is a correlation between the socialization of newcomers and level the dedication they show in the project.

As seen from there literature reviewed above, there are two types of methodologies of research when it comes to long term retention of newcomers. The first methodology is where it is believed that a newcomer needs to have certain qualities to become a top contributor to the project. And the second methodology says that newcomers gradually gain experience and become more senior and valuable contributors to the team. But both of the methodologies agree that top contributors in a project are valuable assets to the project and

efforts must be made to retain them for a longer period of time. The next section of this chapter discusses how the above-discussed literature applies to open-source projects and their contributors.

3.2 Analysis

To analyse the retention rate in open-source projects data from Systemd, Django, ReactJs, Deno and Spring Boot was gathered and analysed. And a primary analysis was done on the top 10 Python project from GitHub. More information about why these projects were selected and how data was gathered can be found in the Implementation chapter (7).

3.2.1 Retention Rate for OSS projects

In a traditional workspace, the retention rate of the employees is calculated using the following formulae.

$$\text{Retention rate} = \frac{\text{Employees staying entire Period}}{\text{Employees at Period start}} * 100$$

Where the retention rate of an organization is the ratio between the number of original employees currently in the organization verse the number of employees at the start of the counting period. The same formulae can be used when counting the retention rate of contributors in an open-source projects. Below formulae was used to calculate the retention rate of contributors in open-source projects.

$$\text{OSS Retention rate} = \frac{\text{Contributor staying after end of year}}{\text{Contributor count at start of year}} * 100$$

To calculate the retention rate of the contributor over a year following steps were followed. For a repository only commits belonging to year 2019 were considered. All the filtered commits were grouped based on every week in the year in which the commit is made. And for each commit in the group the author was considered contributing for that week of the year if there exists a commit for the author in the given month. Then this aggregated data was used to calculate the retention rate of the contributor over a period of one year for the product.

3.2.2 Top contributors trail

The same analysis was done for top contributors in the project. Where top contributors were defined as the one above 30 percentile in the project when ranked by efforts (2.2.1). To study how top contributors tend to change there focus over time more analysis was done on

Table 3.1: OSS Project Retention rate year 2019

Project	Retention Rate	Top Contributor Retention Rate
Django	18%	58%
Spring Boot	34%	40%
React	16%	50%
Deno	72%	63%
Systemd	29%	72%
Flask	38%	58%
Photon	82%	85%
aiohttp	24%	86%
xonsh	37%	53%
ansible	20%	35%
httpie	47%	83%
keras	33%	66%
requests	20%	36%
scrapy	17%	38%
sentry	32%	43%

individual contributors of particular projects. For each top contributor, all the commit data was capture which spanned multiple projects and distributed over several years. For each commit, by the contributor, the commit was categorised as either addition of new feature or refactoring the code or maintenance commits like adding comments. Based on this data, analysis was made that the contribution pattern of top contributor change over time.

3.3 Result

Based on the analysis shown in the previous section following result was observed (3.1). As seen from the result in table (3.1) open-source software seem to have very less retention rate of developers. From the analysis, it seems that for the project analysed the average retention rate is 34.6%. There are some expectations, the project "Photon" has a very high retention rate but the project is also in its very early stage and there are only 19 contributors in total. For such a small project data available is also small hence doing a deeper analysis is not possible. This result indicates that contributors don't stick with the project for a longer period of time.

It seems that even top contributor of the project who has invested a quite of a lot of time in the project don't stick with the project for a longer duration of time. To validate this hypothesis, commit history for 4 top contributors in the "Django" project were studied and a graph was plotted showing the number of commits done over a particular time. The result is shown in Figure (3.1).

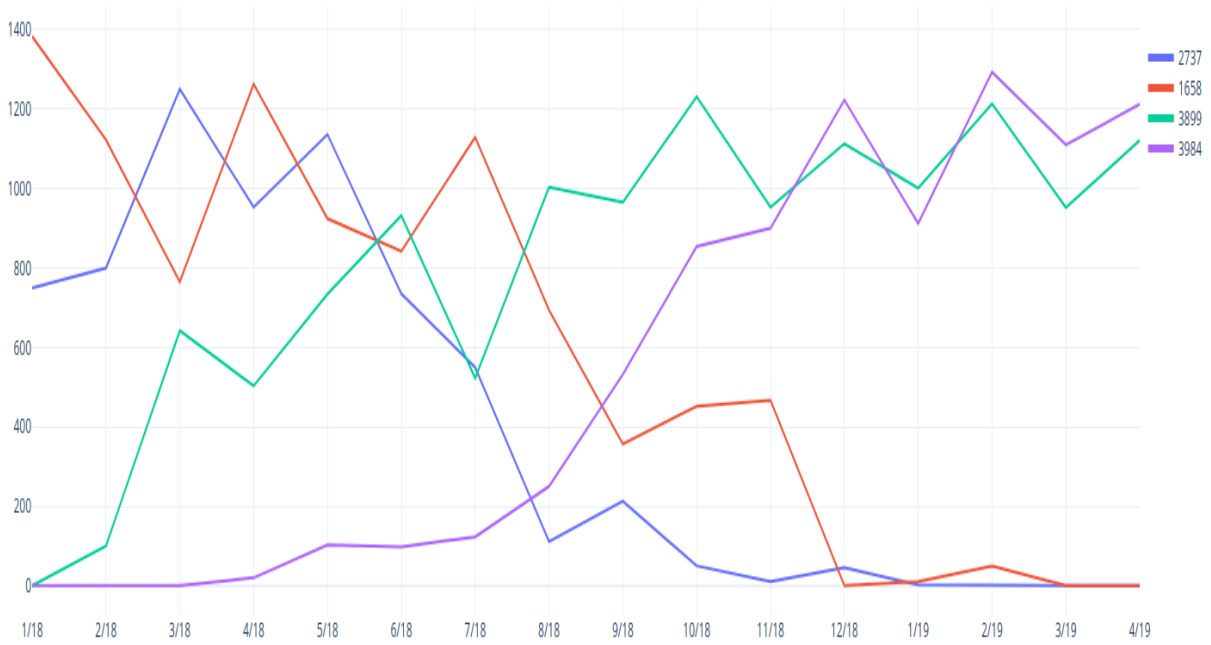


Figure 3.1: Django Project's Top Contributors Commit's Over Time.

Figure (3.1), shows the number of commits done by top 4 contributors in Django project. For anonymity, each user was given a random ID. As seen in the image, users with id 2737 and 1658 were top contributors at the start of the year 2018. And in the same time frame users with ID 3899 and 3984 were not part of the development team for the project. But as time passed by we can see a transition in the contribution pattern of these developers. Top contributors (2737 and 1658) started showing a decline in there contribution and users who were not part of the team (3899 and 3984) gradually progressed towards becoming top contributors in the project. This shows that top contributors to an open-source project usually do not tend to stick a project for a longer duration of time. Top contributors change over time.

Next analysis that was done was to see how the activity of an individual contributor changes overtime. To do that a Top contributor "A" (pseudo name) from Django project was chosen and his trajectory was mapped over a period of time. And Figure (3.2) shows his commit history in different projects for a fixed duration of time.

Figure (3.2) show the different types of commit's done by a contributor "A" from the year 2006 to 2015. The topmost graph shows the commit's done by a contributor "A" in Django, which is a Python-based project and the bottom graph shows the commit's done by contributor "A" in glassfish, which is a Java-based project. As seen from Figure (3.2), that top contributors do not stop contributing altogthere but as time passes but they tend to migrate from project to project. A similar pattern was found for different top contributors in

Figure 3.2: Activity of a Top Contributor overtime

different open-source projects.

Literature review seen in section (3.1) suggests that retention of developers in a software project is a must for the success and progress of the software project. Research has been done even identify which newcomers have the potential to become a long term contributor to the project. Other research has given guidelines to the open-source project as to how to make the project environment more suitable for helping newcomer transition into a long term contributor. But the initial analysis of data from the open-source project has led the author of this dissertation to believe that in open-source projects top contributor or top developers don't tend to stick longer to a single project. The interest of top contributor fades away as time passes by. Initial data analysis has even shown that top contributors switch from project to project becoming a top contributor in a different project. To understand why this pattern is seen in the open-source project more analysis is needed to be done. Thus the learning from the analysis performed in this chapter is as follows ***Top contributors in open source project change over time*** and ***Top contributor don't stick to one project, their interest change over time***.

- ***Top contributors in open source project change over time.***
- ***Top contributor don't stick to one project, their interest change over time.***

4 Organizational Hierarchy

Open-source software development is a process where contributors across the globe contribute to a single project in a distributed fashion. Because for this distributed nature of software development there has been a debate as to whether there is a formal structure to the open-source community. Or does the open-source community is more like a chaotic place where people join the project contribute there part and then leave? In this chapter literature review regarding the same is presented and then the analysis is presented which validated if there is any pattern between the type of contributors for an open-source project.

4.1 Literature Review

Open-source project are constructed in a distributed manner. Where multiple contributors working on the same module of the project might even not be aware of the physical location of other contributors. One of the most famous examples is the anonymous creator of Bitcoin (28), Satoshi Nakamoto. Satoshi Nakamoto started the initiative of creating and submitting code for an open-source cryptocurrency project, Bitcoin. But till date the original identity of Satoshi Nakamoto is unknown (29). Such distributed and anonymous nature of the open-source project has led the researcher to believe that open-source project has no organizational hierarchy as seen in corporate software development teams.

One such research is presented by Raymond in his famous book "The Cathedral and the Bazaar" (30). In his research, the author has presented his finding from studying the Linux (2) and Fetchmail (31) open-source projects. The author suggests that any open-source project generally follow two models the cathedral model or the bazaar model. In the cathedral model, the source code of the final product is available to everyone but only a few exclusive groups of individuals actually contributor to the code base. The second model that the author suggests open-source project follow is a bazaar model. In the bazaar model, the source code of the project is available for anyone to view as well as modify. In this model people usually join the project to provide their contributions and then after submitting there code they leave the project.

Another research that was done by Josh and Parag (32) was to understand what

organizational structure is followed in open-source projects. Authors in the work have suggests that open-source project are not completely open in nature. Many corporate organizations usually benefit from contributing to the open-source project and thus make an effort by allowing their employee to contribute to the open-source project. The authors suggest that in most of the open-source project there are two types of contributors one are the hobbyist contributors who are self-motivated individuals contributing to the project. And the second type of contributors are the corporate contributions who contribute twice as much as the hobbyist contributors.

Jae Moon (33) in his work also has tried to understand what organizational structure does an open-source project follow and what are the essential things required for an open-source project to be successful. The author suggests that for every open-source project there are two models of leadership. One type of leadership model is where there is one single leader in the project who is responsible for driving the entire project. Linux (2) is an example of this kind of project where the entire project is governed by Linus Torvald. And the other type of leadership model that the author suggests is one where there are multiple contributors who are responsible for each individual module in the project.

Red Hat (34) is one of the most important corporate organization in the open-source community. In the following article published by Red Hat (35) efforts have been made to understand the governance model of open-source software projects. Red Hat suggests that there are following governance model in any open-source projects Do-ocracy, Founder-leader, Self-appointing council or board, Electoral, Corporate-backed and Foundation-backed. Authors suggest that in Do-ocracy model contributor who is working on the individual model is responsible for the decisions related to that module. In Founder-leader model the group of individuals who started the projects are responsible for taking all the decision.
Self-appointing council model is where a group of contributors are elected to be in charge of all the decisions related to the project. In Electoral model, a formal election is conducted to select a set of individuals who are responsible for taking all the decision in the project.
Corporate-backed and Foundation-backed model is where the open-source project is controlled by a corporation or a foundation group. In this article, the author suggests that open-source source are controlled and maintained by a particular form of the governance model.

As seen from the literature review that research interest has always been to identify an organizational structure in the open-source project. Some research suggests that there is no organizational structure in open-source projects. It is more like a bazaar model where people join the project submit there work and then leave. Another literature have tried to categorized contributors based on motivation and experience level. Research by Red Hat has tried to formally model the governance structure of the open-source project. To verify the above ideologies, analysis was performed to identify different type of activities performed by

contributors in the open-source project. And an effort was made to categorize the contributors based on the type of activity they perform so as to understand the organizational model for open-source projects.

4.2 Analysis

To analysis, if there is an organizational structure in the open-source projects following technique was utilized. For each contributor all different activities performed were capture. Activities capture included commits make, type of code that was contributed, the number of issues closed and open and the number of pull requests closed and open. After doing an initial analysis of data a pattern was observed where two categories of contributors were identified. The first type of contributors had a huge code related activities like adding new commits, refactoring the existing code, deleting old code etc. And another type of contributors were contributor with a high number of admin tasks like creating and closing an issue in the project, closing a pull request etc. Section below explains more about the analysis that was done based on the type of contributors. More detail on what data was collected and what process was used to collect the data can be found in the Implementation Chapter (7).

4.2.1 Code Contribution

To analyse activities related to the code base of the project, commit data for every contributor was collected for every repository. For each commit done by the contributor a difference between the original code and newly committed was calculated. Based on the difference calculated by the commit, commits were filtered based on if the commit was actual code related commit like adding or modifying new code or whether the commit was created due to admin related task like merging a pull request. For commit, there was code related commits further analysis was done to categories commits based on the type of changes that was been done to the commit. Every code related commits were categorised into four types of categories commit that add new code to the code base, commits that modify existing code base, commits that deleted code and commits that add comments or documentation to the code base.

Commits which add, replace or delete code to the code base were considered into one category and commits that add comments and commits that were generated due to admin related task like merging pull requests were considered into another type of category. Then for each developer, a count was generated for each month for the two types of commit categories. The result of this was plotted as a line chart show in Figure (4.1a) and (4.2a).

4.2.2 Admin Contribution

To calculate admin related contribution done by a developer three things were taken into consideration. Firstly all the commit which were efforts was made for adding comments and documentation to the code base. Commits like these indicate that the contributor is making an effort to make the overall project accessible and maintainable. The second thing that was taken into consideration was the number of issues that the author has created and closed. Creating an issue indicates that the contributor is involved in introducing new feature and fixing bugs for the to the project. Third, thing that was taken into consideration is the number of pull requests closed by the contributors. Closing a pull request in an open-source project taken efforts of the contributor as the contributor who is closing the pull request must make review the newly submitted code and make sure that it does not break existing code base. All of these three factors were considered to analysis how much effort was given by a contributor for doing the admin related task in the project. For all the contributor in a repository, a count was calculated for each of the three factors bucketed by every month. The result of this analysis is shown in Figure (4.1b) and (4.2b).

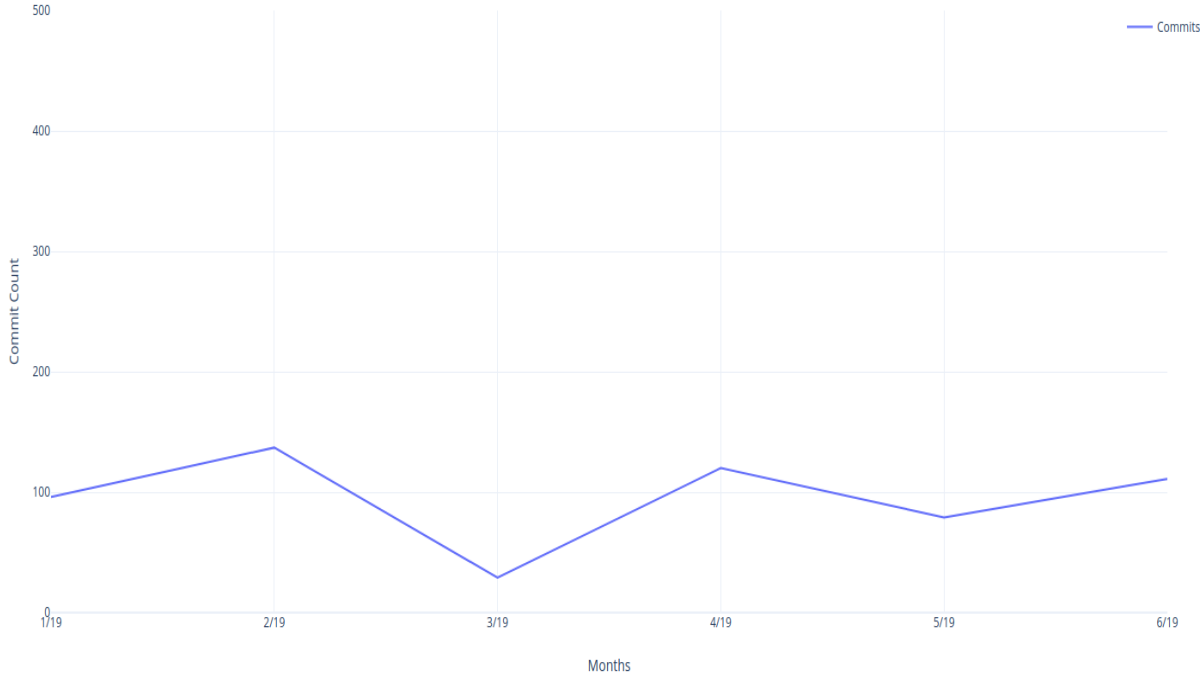
4.3 Result

The result of the analysis is show in Figure (4.1) and (4.2). Figure (4.1) and (4.2) shows the activities performed by two developers **A** and **B** in the Django project for 6 months respectively, starting from 1st January 2019 to 30th June 2019.

As seen from activities performed by developer **A** in Figure (4.1). Developer **A** seemed to be investing more efforts in admin related tasks of the project. The number of commits related to adding documentation and comments to the project is relatively very high. Also, the count for non-code related tasks such as creating and closing issues and closing pull request is high. And at the same time, the number of code commits for the developer **A** is relatively very less than the number of commits done developer **B**. Which suggests that developer **A** invests most of his time and efforts in doing the non-coding related tasks than actually contributing to the code base of the project.

Opposite behaviour is visible when comparing the activities of developer **B** with developer **A**. Developer **B** has a relatively high number of code commits than developer **A**. And developer **B** seem to not be contributing to admin related task of the project. As seen in Figure (4.2), developer **B** has an almost constant number of commits where an effort is made to either add or modify the code base of the project. But at the same time developer **B** has shown very less interest in admin related task like creating and closing issue and pull request. This led us to believe that **B** is more focus on adding code to the code base then doing admin related task.

(a) Code Contribution



(b) Admin Contribution

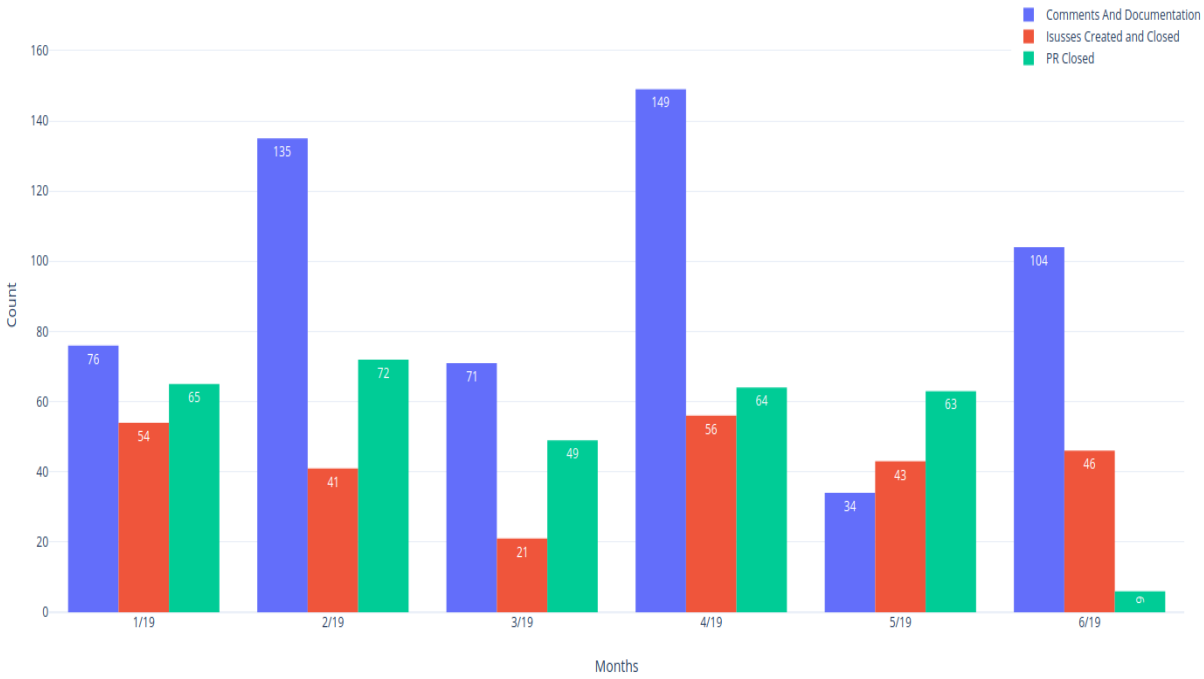
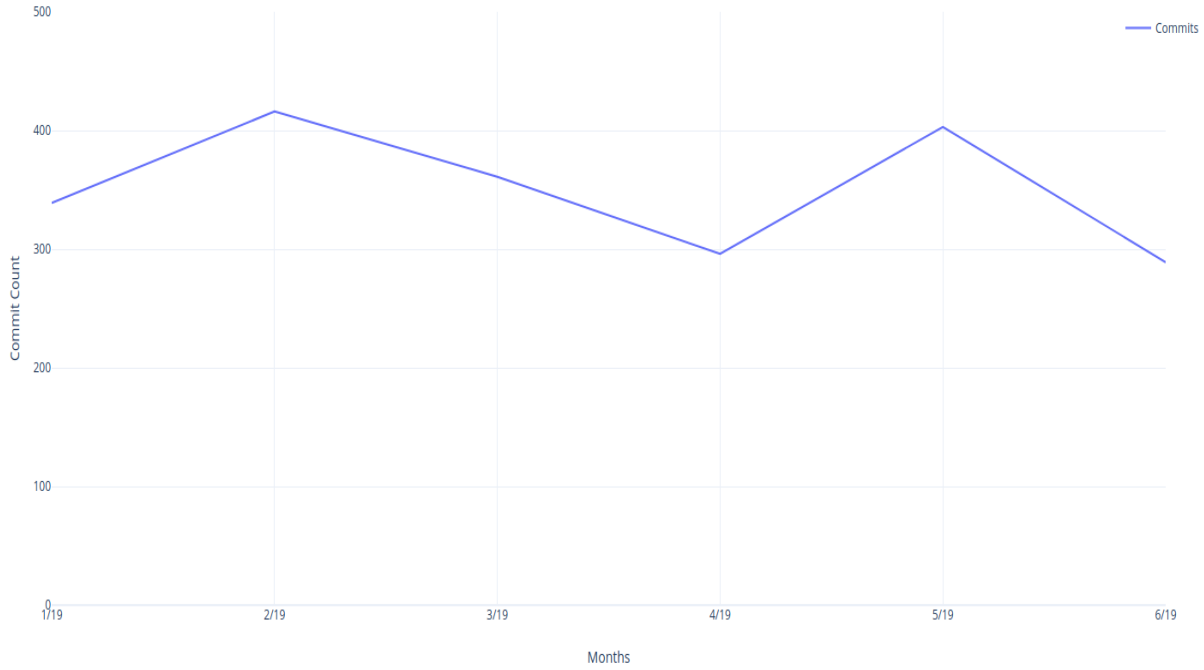


Figure 4.1: Contribution by Developer **A** in Django Project

(a) Code Contribution



(b) Admin Contribution

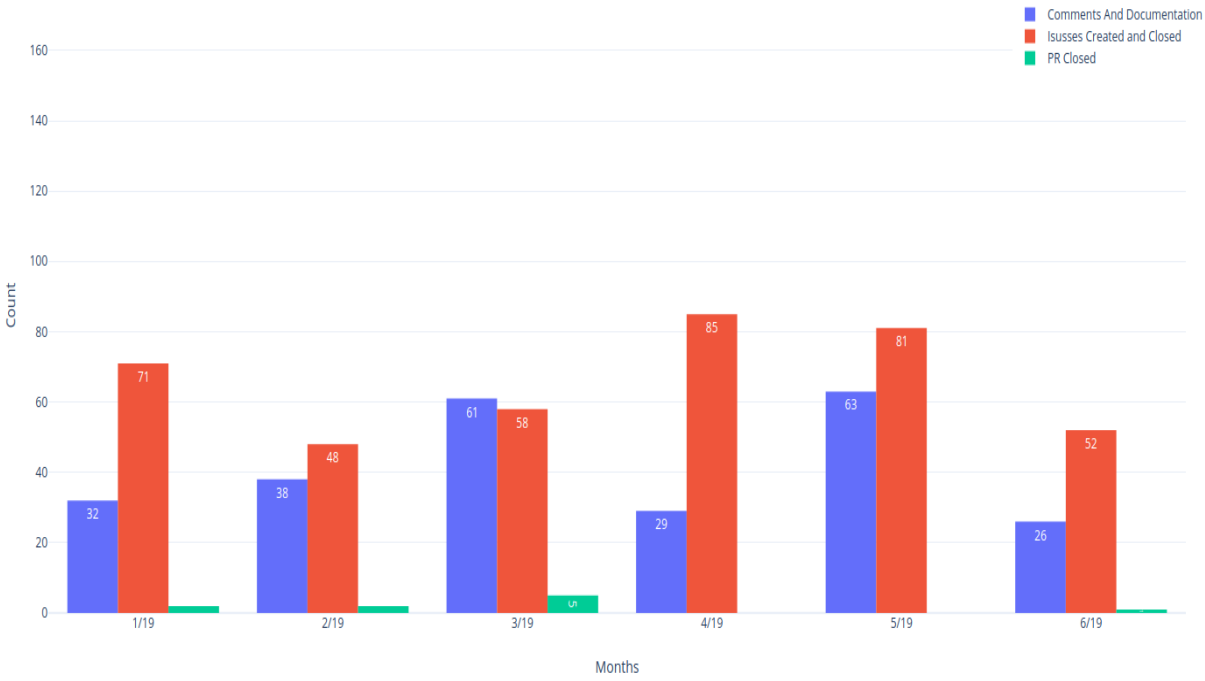


Figure 4.2: Contribution by Developer **B** in Django Project

A similar pattern was observed for multiple contributors in the same repository and other repositories like ReactJs, Spring Boot, Deno and Systemd. This initial analysis has led the author of this dissertation to believe that there are two types of contributors in any given open-source project based on what type of contributor the tend to do. First being the contributors who are more focused on adding new code and modifying the codebase of the project. And the second type of contributors are the admins, who spend most of there efforts in doing the admin related task of the project like creating and closing issues and pull request. And also adding comments in the code and document to the project. More analysis is needed to be done to verify if the same pattern is visible in the majority of open-source projects. Thus the learning from this analysis is as follows.

- ***Analysis show that there are two types of contributor in Open Source Projects. First one being Contributors and second one been the Admins.***
- ***Contributors are the one who do the major work for the project. Like adding new features, fixing bugs etc. They are the one who write the majority of code.***
- ***Admins are the contributor who mainly concentrates on helping the community by replying to questions, closing pull requests.***

5 Ownership

One of the most distinguishing features of the open-source software methodology is its openness and collaborative nature. Due to this collaborative nature, it is usually assumed that there would be no common pattern between users contributing to the project. In this chapter, a literature review is presented which discusses the same question in-depth. Then the chapter concluded with the analysis done on the open-source projects to identify any pattern between the contributions done by the contributors.

5.1 Literature Review

In the famous work by Raymond, "The Cathedral and the Bazaar" (30). The author suggests that the open-source projects operate in two ways the cathedral model and the bazaar model. The author suggests that in the cathedral model a exclusive group of individuals have the absolute ownership over the project and they decide which contributor contributes to which module of the project. And the work suggests that most of the open-source project follow the Bazaar model, where there is no restriction on work done by contributors. The author suggests that there is is not structure involved in the contributions done by the contributors. Any contributor new or old can join the project and contribute to any desired part of the project.

In a similar study by Padhye and Mani (36), the authors have studied the types of pull requests that are accepted in the various open-source software projects. Their study suggests that there is a high chance of a pull request getting merged in the project if the pull request is related to a bug fix then if the project is related to adding a new feature. Authors in their study also suggest that most of the contribution that is done in the open-source projects are from an external source. More code is added to the codebase by contributors who are new to a project than the contributor in the core team of the project. Their research also suggests that for a project written in a mainstream language such as Python (3) and Javascript (37) have a high rate pull request merge rate than a project written in a not so common language like Scala (38).

Open source software development is a distributed efforts of geographically seperated

contributors. This can have effect of the way the contributors contribute to the project. In his works Giorgio (39), suggest that one of the drawback of open-source software development when compared with commerical software development is its distributed nature. The author suggests that having a distributed team means having trouble in cordination between the developers. The author suggests that due to lack of a common leader or an organization governing the overall project, many project dont reach their complition phase. The study suggests that there seems to be no common pattern that can be identified in the way the contributors contribute to an open-source project.

A similar study was done by Xiaohui and Everett (40). In there study there authors have tried to model swarm model to open-source project contribution. Swarm model is study of how collective behaviour occur in a group of social animals and insets. Authors in there study have suggests that the swarm model is not limited to solving optimization problems in computational field but can be used to model open-source contributions. In their work authors have tried to model the open-source contributor using the swarm model. Result of their analysis has shown that in open-source project most contributors contribute to various models of the project based on there need and motivation. Authors suggest that open-source software development is a gradually growing process, contributor join the project contributor to modules of their liking and leave there is no form of governance over the process. And this way the software gradually gets build in its final form.

The literature reviewed above shows that the open-source development process is a more chaotic process than an organized effort. Most of the research suggests that there seems to be no pattern in the contributions made by contributors in the open-source project. To verify this clam analysis described in the section below was performed. For contributors to an open-source project, an analysis was made to identify if there is a pattern that can be identified based on the contributions made.

5.2 Analysis

To analyse, if there is any pattern in the contributions made by the contributors in the open-source project it was required to visualize if there is any specific part of the project that an individual contributor is more focused on. To perform the said analysis commits from individual projects were grouped by the contributors and the code was parsed to visualize which modules contributor has contributed code to.

5.2.1 Module Based Contribution

The analysis was performed on following repositories Django, ReactJs, Deno, Spring Boot and Systemd. Given the scope of the research, only the contributions of top contributors

were taken into consideration for performing analysis. Contributors were ranked based on decreasing order of there effort score in each repository (2.2.1). Then top 10 contributors were selected on whom more detail analysis was performed.

A combination of automation and manual process was used to analyze which modules a contributor has contributed to. To calculate which module a contributor has contributed to, a list of code changes done by the contributor was captured based on the commits done by the contributor in the repository. Then from each code change done by the contributor a list of the module changed was extracted. The list of module changed was generated using the current file path that the code has been written to as well as the module that the current code depends upon.

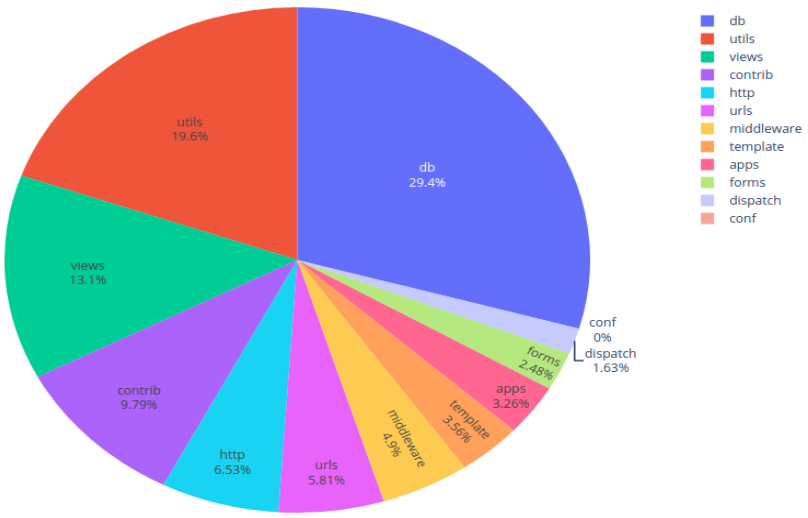
Then from the list of module path, there were generated each path was manually tagged to a module in the code base specific to the repository. For each change made by the contribute in an individual module a single point was assigned to the contributor for the module. This led to each contributor having a score for each module that they contributed to. This score was represented a pie chart showing the percentage of code contributed to a particular module with respect to the overall contribution made by the contributor. Result of this analysis can be seen in Figure (5.1). More detail explanation about what data was collected and what technique was used to collect the data can be found in the Implementation Chapter (7).

5.3 Result

Result of the analysis is shown in Figure (5.1). Figure (5.1) shows the efforts for two contributors **A** and **B** in an open-source project named Django. Where **A** and **B** are among the top contributors in the project. For each contributor, the effort score was calculated for each module that they contributed to. And then the percentage contribution for each module was calculated based on the total effort for each contributor. The result is shown in the form of a pie chart.

We can see in Figure (5.1), that there seems to be a difference in the way contributors contribute to any project. Contributor **A** seems to be showing more efforts in contributing to the database module whereas contributor **B** seems to be showing more efforts in contributing to the Http module. A similar pattern was seen in the contributions made for multiple projects by multiple contributors to the project. The same analysis was performed on other open-source projects like ReactJs, Systemd, Deno and Spring Boot. Analysis of these projects also relieved the same result. This led the author of this dissertation to believe that the given an open-source project all contributors contribute to a specific module of there choose based on liking or motivation of the individual contributor.

(a) Modules Contributed By Developer A



(b) Modules Contributed By Developer B

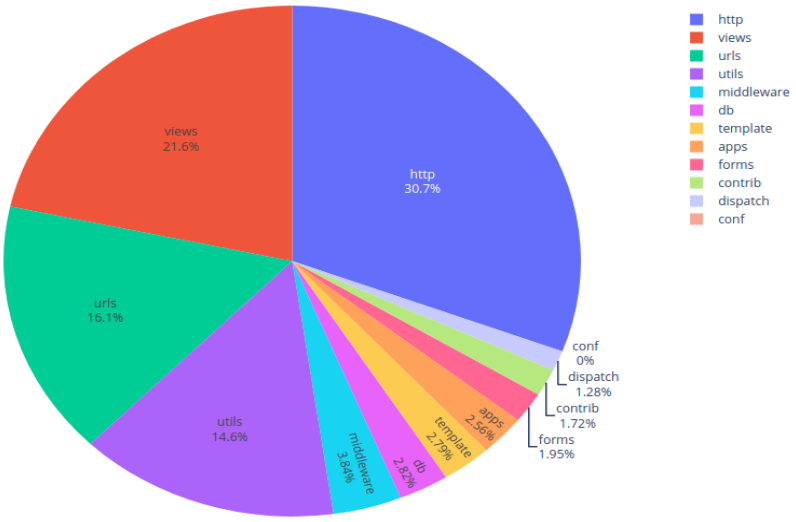


Figure 5.1: Contribution by Developers in Django Project

The literature reviewed in the earlier section suggests that open-source project follow more kind of bazaar model where there is no common pattern between the way the contributors contributed to the project. But the initial analysis done on a few open-source projects suggests that there seems to a pattern in the contributions done by the contributor. Top contributors tend to be more focused on contributing to a specific module as opposed to the standard believe that there exists no pattern in the contribution done in the open-source project. Thus the learning from this analysis is that, ***"Top Contributors in Open Source Project focus on a particular module of their interest than the overall project"***.

> ***Top Contributors in Open Source Project focus on a particular module of their interest than the overall project.***

6 OSS Issues Analysis

Survivability of an open-source project depends upon active contributions form newcomers in the project. But research has shown that there is a shortage of newcomers joining the open-source project. This chapter starts with a literature review discussing the problem of shortage of newcomers in open-source project. Then an analysis on issues of various open-source projects is done and the result of the analysis is presented.

6.1 State of the Art

Open-source software development processes is where contributors across the globe contribute towards a single project. One of the major defining factors of open-source development is that there is no single body who is controlling the entire project and the combined effort of individual contributors is what keeps the project alive. Israr Qureshi and Yulin Fang (27) in their research studied the contribution in 40 OSS projects. Based on there study the authors have suggested that accepting innovative ideas from newcomers to the project is the reason that some of the open-source projects have been able to dominate the software industry.

Research has even shown that one of the major reason for the failure of some of the open-source project is the shortage of newcomers. In there literature Crowston and Annabi (6) have suggested that one of the major reason why some of the open-source projects have failed to delivered is due to a shortage of new contributors. The authors have also suggested that one of the main reason why some of the open-source projects have succeeded is the availability of a pool of new contributors. Their research suggests that the reason why this happens has is that having a stronger community in the project attract more contributors to the project and thus making the community more attractive.

Previous research has shown that a constant influx of newcomers is crucial for the survivability of any open-source project. But other researchers have also shown that most of the open-source project faces a shortage of newcomers. Steinmacher and Wiese (9) in there research have gathered data related to multiple open-source projects from multiple sources like from the publically available codebase, version control system used by the project and

issue trackers like Jira. In their research authors have suggested that most of the open-source project face a shortage of new contributors willing to contribute in the project and most of the newcomers in the project abandon the project a very early stage. Authors have suggested that there can be multiple reasons why newcomers abandon the project. Few of the reason the author suggests are delay in timely reply to the question asked by newcomers, getting an unpolite answer for the questions asked by newcomers and many more.

Research has also been done to understand why newcomers abandon the project and what kind of barriers are faced by newcomer when joining an open-source project. Another literature review by Scacchi (41) suggests that one of the most prominant reasons that newcomers leave an open-source project is because they are expected to understand the project on their own. The research focuses on how requirement engineering is done in open-source projects and how does that knowledge is transferred to a newcomer who is willing to contribute to the project. The author suggests that due to lack of a clear knowledge sharing process or path, newcomer generally have to understand the project requirements on their own. And if the project is of huge size there is a higher chance that newcomers would find it difficult to understand the project without any mentorship.

Similar research was done by Steinmacher (7). In his research, the author has tried to identify the barrier faced by newcomer while contributing to open-source projects. The author has collected data from publicly available code base as well as from interviews with contributors. In his research author has categorised the barriers into three main categories, motivation to contribute, attractiveness to the project and the ease of joining process.

Research is available which suggests that a constant inflow of newcomers to an open-source project is crucial for the survivability of the project. But there seems to have a decrease in the rate of newcomers joining the open-source community due to the multiple barriers they face. To understand the type of barriers that newcomers face for a project an initial analysis is done on the type of issues that are submitted to the project. Detail about the analysis process and the result of the analysis is presented below.

6.2 Analysis

To analysis what is the most common barrier faced by a newcomer joining an open-source project analysis of issues submitted in the open-source project was done. Open-source projects usually provide a way to submit a bug, feature request and any question regarding the project via a publicly accessible issue tracking system. For this analysis data from issues from the GitHub issue tracking system was used. Given the limited scope of this research, the analysis was done only on Django, ReactJs, Deno, Systemd and Spring boot open-source project. More information about what data was collected and the process to do so can be

found in the Implementation chapter (7).

To understand what is the most common type of issues faced by open-source projects data from issues submitted to the open-source project was gather and analysis was made. For analysis data were obtained from the issues submitted using the GitHub issue tracking system. Github issue tracking system was chosen as the main source of analysis because it is the first point of contact a newcomer would ask a question or submit an issue related to the project.

6.2.1 Issue Categorization

Each GitHub issue has multiple data points that can be used while analyzing the type of issue. One of the most useful data point to analyze the type of issue is the tag that the contributors have assigned to a particular issue. Each issue can have multiple tags assigned to it ranging from a bug, feature request, maintenance, documentation, question etc. Having the issues precategorized by the contributor's help in a faster and better categorization of the issues.

To analyze the uncategorised issues or to further analyze the ambiguous issues submitted by the contributor a combination of a manual and automated process was used. For each uncategorised or ambiguous issue, the title and message body were extracted from the issue and initial analysis was made manually to find out the most common type of issues that are present in the data. Then based on the initial analysis of the data a list of commonly used words were taken out. For example, issues having words like "setup", "not working", "installation" and "running" in the issue title or body have a high chance that the issue is where the contributor is having difficulties in setting up his workstation for the development of the project. Then from the uncategorized issues, all the issues having the same set of words were filtered out and assigned to a particular group, in this case, "Workstation Setup". Then a manual inspection was done to check if there are some issues that should not be belonging to the said group. This process was repeated on the remaining uncategorized issues several times until a proper set of groups were found.

Using the above approach issues from multiple projects were categorized into groups and a relative percentage was calculated for the number of issues belonging to each group. More information on the actual process and the detail on what data was used and how can found in the Implementation chapter (7).

6.3 Result

Result of the above analysis can be seen in the table (6.1). Table (6.1), shows the percentage of issues belonging to a particular category in different open-source projects. As

Table 6.1: OSS Issues Categorization

Issue Type	Django	ReactJs	Deno	SystemD	Spring Boot
Bug	68.23%	73.28%	66.94%	74.12%	73.66%
Feature Request	22.17%	11.84%	27.32%	20.27%	17.85%
Workstation Setup	5.31%	6.13%	2.78%	1.38%	3.47%
Help / Documentation	2.67%	5.21%	0.77%	2.91%	1.9%
Others	1.62%	3.58%	2.19%	1.32%	3.12%

seen from the table, issues of type bugs and feature request are the most common type of issues submitted to the project. And it is expected behaviour because open-source project work in the same fashion people globally submits bugs and feature request related to the project. And then an individual contributor submits the code changes that are related to the issue.

Table (6.1), also shows that the next common type of issues expect bugs and feature in the open-source project are issues related to setting up the workstation for development of the project. Analysis has shown that most of the issues that an open-source project receive excluding the issues reporting a bug or demanding a feature are related to problems facing while setting up their workstation for development purpose. Such issues are mostly answered by other contributors but due to the open-source nature of the project there can be a delay in the response time for the issue and the author believes that this may lead to newcomers losing motivation. Thus learning from this analysis is that, *"The top most issue faced by the contributor in open source projects is setting up workstation"*.

> ***The top most issue faced by the contributor in open source projects is setting up workstation.***

7 Implementation

This chapter gives a detailed overview of the process that was used to collect and analyse the data of open-source projects. This chapter starts with a detail discussion on the challenges that were faced when gathering the data and discussion on techniques used to overcome those challenges. The chapter concludes with a detail explanation of the system developed to gather and analyze the data and a detail explanation of each of the components in the system.

7.1 Challenges

This research required analysis to be done for multiple open-source projects. GitHub was used as the main source for gathering data about the project. When gathering data from GitHub two main challenges were faced. The first challenge was dealing with the rate limit on the GitHub API, which prevented from querying a large amount of data at a single instance. And the second challenge was dealing with ambiguity in the data provided by GitHub API. Next section provides more detail explanation on the challenges and provides a solution that was developed to mitigate the same.

7.1.1 GitHub API rate limit

GitHub platform was used as the data source to gather information about the open-source project. GitHub provides API using which data related to open-source projects can be requested but GitHub has an API rate limit of 5000 requests per minute. Having such a small rate limit on the API prevents doing analysis one a large dataset. Given the high frequency of commits done in open-source projects (42), the rate limit of 5000 requests per minutes seems too small to perform any kind of analysis on the data.

To solve this problem a system was designed with a pipeline to process data and save the data in an offline database. More information about the data processing pipeline is given in the Data Analysis Pipeline (7.2) section. The data pipeline was designed in such a way that it would start collecting data by requesting the GitHub API and if the rate limit of the API is crossed it will suspend the execution for one hour and then try calling the GitHub API again

after one hour. And all the data collected in the process was stored in an offline database making it available for future analysis.

7.1.2 Data Ambiguity

Research by Kalliamvakou (13) has also shown that there is a chance that there would be a difference between the actual data and the data that is mined from GitHub. There is a lot of data ambiguity in data collected from the GitHub API. One example of such data ambiguity is when the a same users uses two different accounts and email id to commit in the repository thus increasing the count of unique users contributing to the repository. Another data ambiguity is where a commit is created in the repository whenever a pull request is merged in the project. GitHub also assigns the author of the commit to the user who is merging the pull request and not the user who has submitted to code.

To solve this data ambiguity problem a combination of a manual and automated process was used. Each type of data ambiguity was treated differently based on the type of data that it represented. More information on each type of data of ambiguity can be found on the section Data Processors (7.4).

7.2 Data Analysis Pipeline

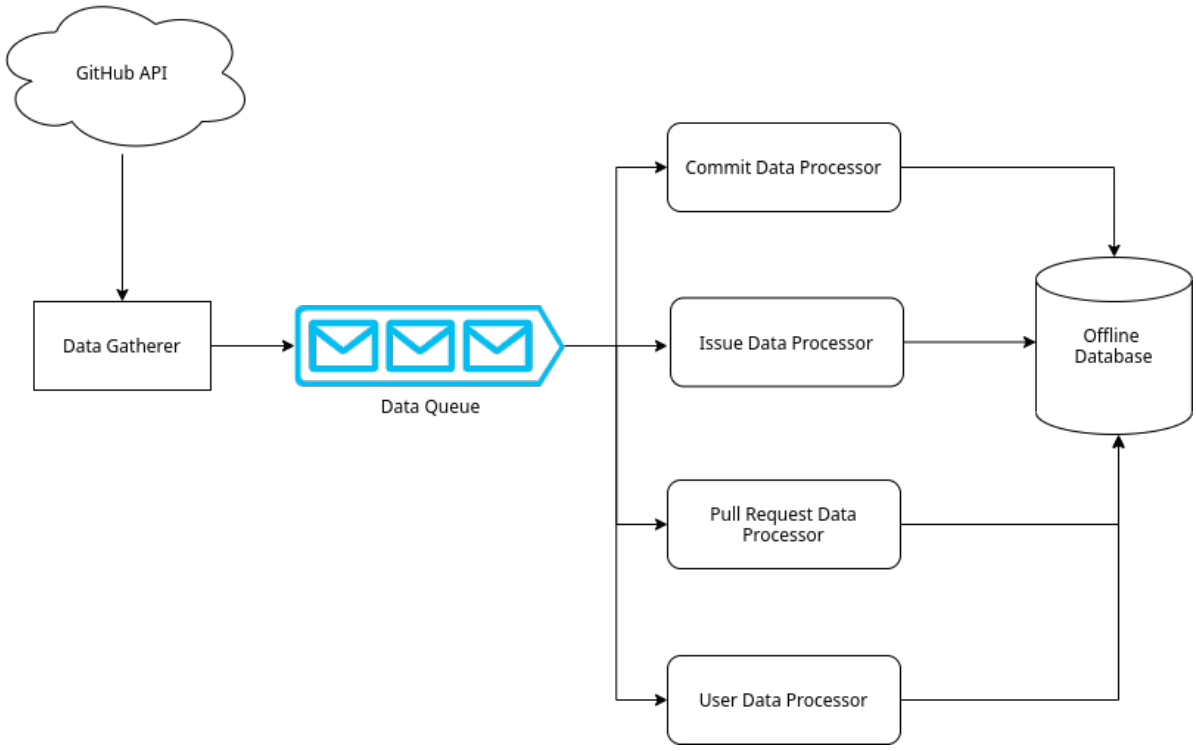


Figure 7.1: Data Analysis Pipeline

Figure (7.1) show the data pipeline that was created to analyze the data for open-source

projects using the GitHub API. The main objective of designing a data pipeline was to capture data from the GitHub API and store it offline for future data analysis. The second objective that the data pipeline satisfied was to allow adding pluggable modules for processing data based on the type of individual dataset.

As seen in Figure (7.1), GitHub API was the main data source that was used for analysing data. As discussed in the previous section GitHub API has an hourly limit of 5000 requests (12). To overcome this limit a "Data Gatherer" module was designed. Data gatherer module is responsible to continuously request data using the GitHub API. If the API returns a valid dataset then the data was added a data processing queue. Else if the GitHub API returns an error indicating that rate limit has reached then the data gatherer module will suspend itself for an hour and then retry the same request after one hour. All the message send in the data queue are consumed by different data processor modules based on the type of data currently processing. Each processor will then do a data cleaning and data transformation task-specific to the type of data in the queue. All the process data is then stored in a PostgreSQL (4) database. Data from the PostgreSQL database then used to analyze the data and visualize result from the analysis. Detail discussion on each of the modules and how it operates is discussed in more detail in the below sections.

7.3 Data Gatherer

Data gatherer module is the first module in the pipeline and it is responsible for requesting data from the GitHub API. This module takes care of the hourly 5000 limits on the GitHub API. Data gatherer module also maintains a local state saving the last successful data was that requested from the GitHub API.

Data gathering module always run in the background. It takes a list of GitHub repositories as an input. Given the list of repositories, the module tries to request data about the repository using the GitHub API. Data that is requested using the API is the commits data, pull request data, issues data and data related to the contributors of the repository. Then the data is added to the data processing queue which is then consumed by other data processors.

If during the process of requesting data from the GitHub API if the API returns a 403 HTTP status code the module would know it the rate limit for the hours has reached. In such a case, the module suspends its execution for an hour and then retries after an hour and keep pulling data until the rate limit reaches again. This helps to overcome the GitHub rate limit and provides a way to perform analysis on a wider dataset.

7.4 Data Processors

Data gatherer module after getting data from the GitHub API pushes the data in the data processing queue. Then the data from the queue is consumed by different types of data processor based on the data that is currently in the queue. Each data processor then does data cleansing on the data, perform aggregation operation is required and the stored the final aggregated data in the PostgreSQL database. This section describes each individual data processor in detail and explains what type of data each processor consumes and what data cleaning operation is performed on each of the datasets.

7.4.1 Commits Data Processor

Commit data processor works on any commit related data pushed in the data processing queue. It does different types of preprocessing with the data like filtering the commits, associating a type for the commit, calculating the effort score for the commit and finding out which module the commit was related to. The detail on each of these tasks is given in the section below.

Type of Commits

Commit data was used to analyse what kind of work has a contributor done in the open-source project. To do that commits were categorized into four main types as follows.

- **Addition:** Commits that add new code to the codebase.
- **Maintenance:** Commits that modify existing code in the codebase.
- **Deletion:** Commits that remove existing code.
- **Comments:** Commit that adds comments or documentation to the codebase.

Before categorizing commits the commit processor module filters out the commits which are related to any code change and commits that are generated due to admin related actives. Whenever a pull request is merged in the codebase a commit is generated with empty content and the author of the commit is set to the contributor who is merging the pull request. Counting such commit could not be considered as a contribution to the codebase of the project and it would lead to counting the work done twice once in commit and one while counting the pull requests. Hence the first task done by the commit processor is to filter out commits which have zero content in it.

Another task that is carried out by the commit processor is to categorize commits into addition, maintenance, deletion or comments commit. Categorizing comments into

comments commit is easy, it involves checking if the file modified in the commit is a documentation file or the code added in the commit is a comment in the code then the commit type is comments commit. The file extension was used to identify if the changed/added file is a documentation file and regular expression matching was used to identify if newly added code is a comment. To identify if a commit is of type addition, maintenance or deletion, python library GitPython (43) was used. GitPython library uses *git diff* command to find which code was changed in the commit and parses the output to return a list of changes that were done. Then for each modification analysis was made if the modification added a new line, deletes an existing line or replaces an existing line of code. Commit was assigned the type based on the maximum from the types of modifications. Thus categorizing commit into addition, modification or deletion. All the commit information with the type of commit and the code different was saved in the PostgreSQL database for further analysis.

Effort Estimation

Commit processor is also responsible for assigning an effort score to every commit in the repository. The effort score was used to calculate the distribution of contribution done by contributors in an open-source project (2). Algorithm (1) was used to calculate the effort score for each commit done in the repository. More points were given for any commit type that is of modification, deletion and comments type. The idea behind doing this was that someone who is modifying an existing codebase would require more efforts in terms of first understanding the codebase and then modifying it. While adding a new independent feature to the codebase would not require a relatively low understanding of how the project works. Commit with its effort score was save in the PostgreSQL database for further analysis.

Module Contribution

Commit data was also used to analyze which module the contributor has contributed to (5). This data was used to understand the distribution of efforts by contributors. A combination of a manual and automated process was used to calculate which modules a contributor has contributed to. First, the difference between the code that the commit changed was calculated using GitPython. Then the list of filenames that were modified in the commit was generated and associated with the author of the commit. For the files that were modified in the commit and list of other modules that the current module depends upon was generated using a combination of regex match and by parsing the code using the abstract syntax tree. All the file path for the modules that the current modified code depends upon as also associated with contributions made by the commit author. Then after doing this for the entire repository and unique list of file path was generated and each file path was associated with a module manually. Thus associating contribution to a module for each contributor

based on his commits in the codebase. All the final processed result was stored in the PostgreSQL database for further analysis.

7.4.2 Issue Data Processor

The second type of data processor is the issue data processor. The main task issue data processor is to categories the issue type based on the content of the issue and to capture the metadata of the issue. Data from this analysis were used in identifying the most common type of problem faced by open-source projects in the chapter 6.

Issue Type

One of the analyses on the issues of open-source projects was to categories the issues based on the type of content they were describing. GitHub issue system has a way to add tags to each issue submitted to the project. For most of the open-source project issues that described a bug or issues there were described as a feature request were tagged accordingly. Hence categorizing issues that were of type bugs and feature requested was simply done by using the already associated tags.

But for other non-tagged issues, a combination of a manual and automated process was used to group the issues in one category. For each non-tagged issues, a manual an initial manual analysis was done to see if there is a common pattern in the issue by analysing its title and message content. For example, a manual analysis of issue showed that issues having words like "setup", "workstation", "complie" etc are issues which are most likely related to issues having to deal with difficulty in setting up the workstation for development of the project. Then an automated script was executed which filtered out issues containing the same set of words in the title and message of other issues and grouped them into one category. After that, a manual evaluation was again done to verify that issues are grouped properly into a single category and there is no issue in the category which was not supposed to be there. This process was repeated again multiple times until there were very fewer issues left which were not categorized or there seemed to be no common pattern in the issues. All the aggregated data was stored in the PostgreSQL database for future analysis.

Issue Metadata

Multiple metadata was collected for each issue by the issue processor. Example of some of the data that was collected was issue title, message, information of author submitting the issue, time the issue was submitted and closed, the current status of the issue etc. Information of the contributor who is commenting on the issue or closing the issue was also captured which helped in the analysis which contributors are more focused on adding code to the codebase of the project and which contributors are more focused on doing the admin

related tasks (4).

7.4.3 Pull Requests Data Processor

Pull request data processor was responsible for processing data related to the pull requests submitted for the project. Various type of analysis was done on the like categorizing the pull request based on the type of changes it is introducing in the codebase also capturing the efforts done by the contributor who is submitting the pull request and contributor who is closing the pull request.

Pull Request Type

Pull request data that was extracted using the GitHub API consisted of a lot of noise in there. There were pull requests that were never merged in the codebase or pull requests that were closed without accepting in the project. Such pull requests were filtered out before doing analysis. Only the pull requests that added or modified the code or the documentation of the project were considered for the analysis.

Pull requests were also categorized as addition, modification, deletion or comments type based on the type of changes they introduced in the codebase. To categorize the pull request in each of the types the commits associated with the pull request were analysed. As described in the section (), each commit associated with the pull request was analysed and categorized into one of the other types. Then each pull request was associated with a particular type based on the count per type of commits associated with the pull request.

Pull Request Metadata

Multiple different types of metadata were extracted for a pull request belonging to the project. Some example of metadata that was extracted was the title, message content, commits associated with the pull requests, type of pull request, the time pull request was open and closed etc. One data cleaning that was done was changing the author associated with the pull request. Whenever a pull request is merged on GitHub a new commit is generated in the project and the author of the commit is associated with the contributor who merged the pull request and not the author who submitted the new code that was merged in the codebase. To mitigate this while processing the pull request data the author for the pull request and the commit added by the pull request was changed to the author who originally submitted the pull request. All this analysis was done and saved in the PostgreSQL database for further analysis.

7.4.4 User Data Processor

Contributors data was also collected for different open-source projects to study how different contributor contribute differently to the project. Detail on the data that was collected for the contributor is given in the section below.

User Data Cleaning

User data processor was mainly responsible for processing data related to individual contributors of the project. One of the most important task performed by the user data processor was to maintain the anoministy of the contributors by removed any personal information like email or username from the system. The user data processor added a random unique ID to each contributor in the system and the converts the username and email a hash value which is later used for solving data ambiguity in the data.

User data processor also solved the ambiguity in the data where the same user uses two different account with two different email to contribute to the same repository. This usually happens when the contributor was using his personal email address to contribute to the project but later he used his companies email address to contribute to the same project. To solve this data ambiguity a combination of manual and automated process was used. Add the user with similar name or similar email id were filtered automatically via script and then manual evaluation was done if any two users are the same user with different accounts. Based on the result of manual evaluation the account of multiple users was merged into a single account giving an accurate analysis of the data. All the result of the analysis was saved in the PostgreSQL database for further analysis.

7.5 Data Visualization

The above described analysis was ran on five main projects namely Django, Deno, ReactJs, Spring Boot and Systemd. The reason for choosing these projects was to choose projects developed in different programming languages and with a different domain of operation. Django is developed in Python, Spring Boot is developed in Java, Deno and RectJs are developed in Typescript and Systemd is developed using the C programming language. Also, the domain of each project is different. Django and Spring Boot are used for developing backend applications. Wheres Deno and ReactJs are mostly used to develop frontend applications. And SystemD is a Linux utility. Choosing these projects provided diversity in the dataset.

For visualization the result of the analysis two main tools where used. A web based dashboard was created using the Django project and plotly (44) chart API. The web dashboard would generate data by using SQL to extract data from PostgreSQL database

and then performed some automated analysis on it if required. Then this aggregated data was passed to the plotly library and different visualizations were created based on the type of data. Secondly standalone python script were also written to create a visualization for the result from the data. These scripts similar to the web dashboard used SQL to query from the PostgreSQL and they perform aggregation operation on the database and the generated plots using the plotly graphics library.

8 Result

This chapter summarizes the learning from all the previous chapters and then tries to suggest a set of guidelines that open-source software can follow to attract more newcomers to the project and thus to increase the chance of survivability of the project. This chapter also suggests a set of guidelines that a newcomer can follow to be a top contributor in an open-source project.

8.1 Summary

Previous chapters have provided literature review and analysis on open-source projects as with respect to teamwork, retention, organization hierarchy, ownership and the most common issue based by newcomers. Based on the result obtained from the analysis following are the set of key learning's.

- Open Source Projects are not operated by a huge team of software developers. It is operated by a small group of individuals.
- Top contributors in open source project change over time.
- The top contributor doesn't stick to one project. Their interest changes over time.
- There are two types of contributor in Open source projects. First, one being Contributors and the second one been the Admins.
  - Contributors are the one who does the major work for the project. Like adding new features, fixing bugs etc. They are the one who writes the majority of code.
  - Admins are the contributor who mainly concentrates on helping the community by replying to questions, closing pull requests.
- Top Contributors in Open Source Project focus on a particular module of their interest than the overall project.
- The topmost issue faced by the contributor in open source projects is setting up a workstation.

These learning were based on primary analysis done over few open-source project. Based on these learning a set of guidelines can be suggested to the open-source project as well as newcomers so to operate efficiently in the open-source community.

8.2 Guideline for OSS

Based on the learning stated above this research tries to suggest a set of guidelines principles that can be followed by an open-source project as well as newcomers joining an open-source project.

8.2.1 Guideline for Newcomers

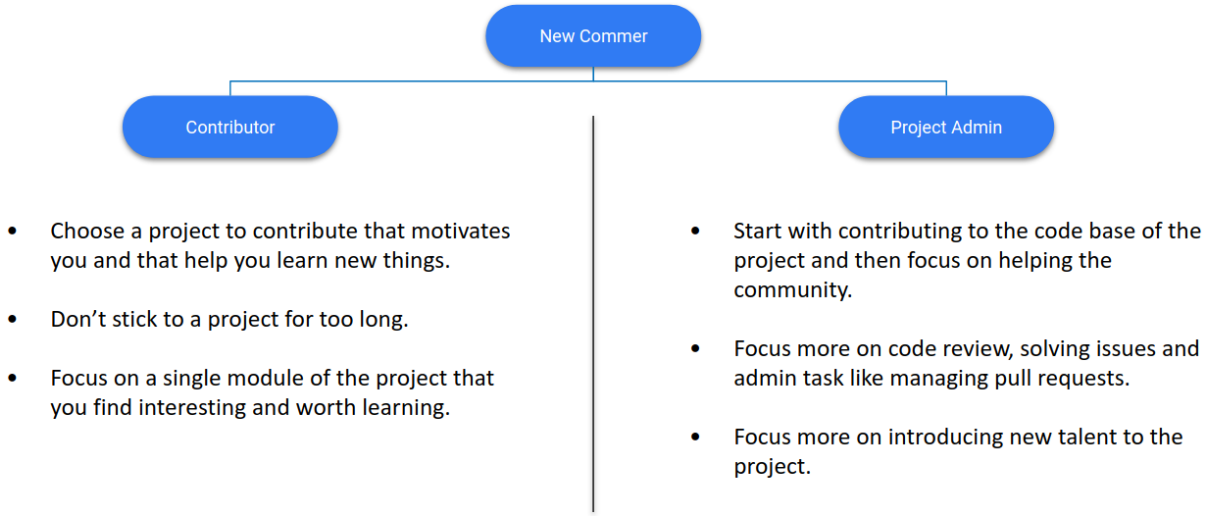


Figure 8.1: Guidelines for newcomers in Open Source Project

As stated in the chapter Organizational Hierarchy (4), there are two types of contributors to open-source projects. First been the code contributors who put most of there efforts in adding and modifying code base of the project. And the second type of contributors is the admins who are mostly concerned about doing admin related tasks such as closing issues and pull request. Thus a different set of guidelines are suggested based on the type of contributor a newcomer aspires to be.

Guideline for Code Contributor

Here are a set of guidelines that a newcomer should follow to become a top contributor who is more focus on contributing to the code base of the project.

- Choose a project to contribute that motivates you and that help you learn new things.
- Don't stick to a project for too long.
- Focus on a single module of the project that you find interesting and worth learning.

Guideline for Project Admin

Following are a set of guidelines that a newcomer should follow if he or she intends to become a top contributor in a project by focusing more on the community then on the code base of the project.

- Start with contributing to the code base of the project and then focus on helping the community.
- Focus more on code review, solving issues and admin task like managing pull requests.
- Focus more on introducing new talent to the project.

8.2.2 Guideline for OSS projects

Result form the analysis can also be used to model a set of guidelines that can be followed by the open-source project to make the project more attractive and thus increase its chance of survivability.

- Focus more on attracting new talent then retaining old talent.
- Add documentation to the project especially targeted towards helping newcomers to get started.
- Make the process easier for newcomers to suggest/submit big and drastic changes. So as to attract new idea and keep the project innovating.
- Make it easy for newcomers to set up workstation by providing proper documents or by automating the process.

9 Conclusion

This chapter concludes the dissertation with an assessment of the research objectives described in the Introduction Chapter (1). Research limitation and future work are also discussed in this chapter.

9.1 Objective Assessment

9.1.1 OSS methodology

Research Objective: To understand the software development methodology for Open Source Software work with respect to newcomers and top contributors.

One of the main objectives of this dissertation was to understand the software development methodology that is followed in open-source projects. To understand how open-source software development operated analysis of open-source software was done concerning five different principles namely teamwork, retention of contributors, organizational hierarchy, ownership and common issues in the open-source project. Analysis of the project for these principles led to the revelation of interesting learnings which led to defining guideline for contributors in the open-source project.

9.1.2 Guideline for OSS

Research Objective: Recommend a list of guidelines for open-source projects and newcomers to open-source projects.

Another objective that this dissertation tried to satisfy was to define a set of guidelines for the open-source projects and newcomer joining an open-source project. Analysis that was done on the five principles led to a methodological understanding of how open-source projects operate. And this led to defining a set of learning's from the open-source project. These learning formed the basis of the guidelines which were defined in this dissertation. Two different types of guidelines were defined, one for open-source projects and other for new contributors in the open-source projects.

9.2 Research Limitation

This dissertation focuses on five principles like teamwork, retention of contributors, organizational hierarchy, ownership and common issues in the open-source projects. These principles were chosen based on the literature review done by the author of the dissertation. There are many more factors that can affect how the open-source project operates. Some of the factors that were not analysed in this dissertation are motivation, socialization in a project and many more. Considering all the factors in the limited scope of this dissertation was not possible hence effort was made to focus on five important factors related and the open-source project.

In this dissertation, the analysis was also done one a limited number of open-source projects. Given the limited scope and time, performing a similar analysis on multiple projects was not possible. But the initial analysis on a few open-source project indicated strongly that similar pattern should exist in other open-source projects as well.

9.3 Future Work

Following are a list of ideas and suggestion for future work related to this dissertation.

- Perform a similar analysis on a larger dataset and validate if the same principle discussed in the dissertation applies to a wider set of projects.
- Perform analysis to see how other factors like motivation, socialization and contributors skillset plays a role in the overall development of an open-source project.
- Do a more granular level analysis for kind of efforts done by different contributors in the project.
- A real-time system can be developed which would categorize contributors based on their activities and then provide guidance in real-time.

Bibliography

[1] Open Source. The Open Source Definition | Open Source Initiative. URL https://opensource.org/osd.

[2] Linux. Linux.org. URL https://www.linux.org/.

[3] Python. Python.org. URL https://www.python.org/.

[4] PostgrSQL. PostgreSQL: The world's most advanced open source database. URL https://www.postgresql.org/.

[5] Netcraft. Netcraft news: July 2020 web server survey. URL https://news.netcraft.com/archives/2020/07/27/july-2020-web-server-survey.html.

[6] Crowston, Kevin, and Howison. Defining open source software project success. *Proceedings of the International Conference on Information Systems*, 2003.

[7] Igor Steinmacher. Supporting newcomers to overcome the barriers to contribute to open source software projects. 2015.

[8] Dan Sholler, Igor Steinmacher, Denae Ford, Mara Averick, Mike Hoye, and Greg Wilson. Ten simple rules for helping newcomers become contributors to open projects. *PLoS Computational Biology*, 15(9):1–10, 2019. ISSN 1553734X. URL https://elib.tcd.ie/login?url=http://search.ebscohost.com/login.aspx?direct=true&db=a9h&AN=138581835.

[9] I. Steinmacher, I. Wiese, A. P. Chaves, and M. A. Gerosa. Why do newcomers abandon open source software projects? pages 25–32, 2013.

[10] I. Steinmacher, I. S. Wiese, A. P. Chaves, and M. A. Gerosa. Newcomers withdrawal in open source software projects: Analysis of hadoop common project. pages 65–74, 2012.

[11] Matt Asay. Open source has a people problem, August 2020. URL https://www.infoworld.com/article/3570483/open-source-has-a-people-problem.html.

[12] GitHub. Github rate Limit. URL https://developer.github.com/v3/rate%5Flimit/.

[13] Eirini Kalliamvakou, Georgios Gousios, Kelly Blincoe, Leif Singer, Daniel German, and Daniela Damian. An in-depth study of the promises and perils of mining github. *Empirical Software Engineering*, 21(5):2035, 2016. ISSN 13823256. URL https://elib.tcd.ie/login?url=http://search.ebscohost.com/login.aspx?direct=true&db=edb&AN=117576131.

[14] R. Katz and T. Allen. Investigating the not invented here (NIH) syndrome : A look at the performance, tenure, and communication patterns of 50 research and development project groups : Research and development management. 1982.

[15] Yaobin Lu, Chunjie Xiang, Bin Wang, and Xiaopeng Wang. What affects information systems development team performance? an exploratory study from the perspective of combined socio-technical theory and coordination theory. *Computers in Human Behavior*, 27(2):811–822, 2011. ISSN 0747-5632. URL https://elib.tcd.ie/login?url=http://search.ebscohost.com/login.aspx?direct=true&db=edselp&AN=S0747563210003493.

[16] Mary Beth Pinto, Jeffrey Pinto, and John Prescott. Antecedents and consequences of project team cross-functional cooperation. *Management Science*, 39:1281–1297, 10 1993. doi: 10.1287/mnsc.39.10.1281.

[17] J. Espinosa, Sandra Slaughter, Robert Kraut, and James Herbsleb. Team knowledge and coordination in geographically distributed software development. *J. of Management Information Systems*, 24:135–169, 07 2007. doi: 10.2753/MIS0742-1222240104.

[18] Martin Hoegl, bullet Hans, and Hans Gemuenden. Teamwork quality and the success of innovative projects: A theoretical concept and empirical evidence. *INFORMS*, 12: 435–449, 08 2001. doi: 10.1287/orsc.12.4.435.10635.

[19] Emily Weimar, Ariadi Nugroho, Joost Visser, Aske Plaat, Martijn Goudbeek, and Alexander Schouten. The influence of teamwork quality on software team performance. 01 2017.

[20] Minghui Zhou and A. Mockus. Developer fluency: achieving true mastery in software projects. 2010.

[21] A. Schilling, S. Laumer, and T. Weitzel. Who will remain? an evaluation of actual person-job and person-team fit to predict developer retention in floss projects. pages 3446–3455, 2012.

[22] KDE Community Home, . URL https://kde.org/.

[23] Google Summer of Code, . URL https://summerofcode.withgoogle.com.

[24] M. Zhou and A. Mockus. Who will stay in the floss community? modeling participant's initial behavior. *IEEE Transactions on Software Engineering*, 41(1):82–99, 2015.

[25] Mellony Graven and Stephen Lerman. Wenger, e. (1998). communities of practice: Learning, meaning and identity. *Journal of Mathematics Teacher Education*, 6:185–194, 06 2003. doi: 10.1023/A:1023947624004.

[26] Eugene Matusov, Nancy Bell, and Barbara Rogoff. Situated learning: Legitimate peripheral participation . jean lave, etienne wenger. *American Ethnologist - AMER ETHNOLOGIST*, 21:918–919, 11 1994. doi: 10.1525/ae.1994.21.4.02a00340.

[27] I. Qureshi and Y. Fang. Socialization in open source software projects: A growth mixture modeling approach. *Organizational Research Methods*, 14:208–238, 2011.

[28] Bitcoin - Open source P2P money, . URL https://bitcoin.org/en/.

[29] Who Is Satoshi Nakamoto, Inventor of Bitcoin? It Doesn't Matter., . URL https://fortune.com/2015/12/09/bitcoin-satoshi-identity/.

[30] Eric S. Raymond and Tim O'Reilly. *The Cathedral and the Bazaar*. O'Reilly amp; Associates, Inc., USA, 1st edition, 1999. ISBN 1565927249.

[31] Fetchmail, . URL https://www.fetchmail.info/.

[32] Josh Lerner, Parag A. Pathak, and Jean Tirole. The dynamics of open-source contributors. *American Economic Review*, 96(2):114–118, May 2006. doi: 10.1257/000282806777211874. URL https://www.aeaweb.org/articles?id=10.1257/000282806777211874.

[33] Jae Yun Moon and Lee Sproull. Essence of distributed work: The case of the linux kernel. *First Monday*, 5, 04 2000. doi: 10.5210/fm.v5i11.801.

[34] The world's open source leader, . URL https://www.redhat.com/en.

[35] Understanding open source governance models, . URL https://www.redhat.com/en/blog/understanding-open-source-governance-models.

[36] Rohan Padhye, Senthil Mani, and Vibha Singhal Sinha. A study of external community contribution to open-source projects on github. page 332–335, 2014. doi: 10.1145/2597073.2597113. URL https://doi.org/10.1145/2597073.2597113.

[37] Javascript, . URL https://www.javascript.com/.

[38] The Scala Programming Language, . URL https://www.scala-lang.org/.

[39] Anthony Giorgio. An analysis and comparison of open-source and proprietary software development methods. 06 2001. doi: 10.13140/RG.2.2.36449.40808.

[40] X. Cui, J. Beaver, E. Stiles, L. Pullum, B. Klump, J. Treadwell, and T. Potok. The swarm model in open source software developer communities. pages 656–660, 2010.

[41] W. Scacchi. Understanding the requirements for developing open source software systems. *IEE Proceedings - Software*, 149(1):24–39, 2002.

[42] Carsten Kolassa, Dirk Riehle, and Michel A. Salim. The empirical commit frequency distribution of open source projects. 2013. doi: 10.1145/2491055.2491073. URL https://doi.org/10.1145/2491055.2491073.

[43] gitpython-developers/GitPython, September 2020. URL https://github.com/gitpython-developers/GitPython. original-date: 2010-11-30T17:34:03Z.

[44] Plotly: The front-end for ML and data science models, . URL /.