A Study on Sustainability of Open Source Software Projects

March 2017

Kazuhiro Yamashita

Abstract

Open Source Software (OSS) is computer software with source code that anyone can study, change, and enhance. Since the quality of OSS systems is sufficient to use in enterprises, more and more enterprises have introduced OSS systems into their businesses. As OSS systems are becoming a type of infrastructure for enterprises, stoppage of the development and maintenance of OSS projects has a large impact on enterprise users. Therefore, the sustainability of OSS projects is important.

In this dissertation, we consider sustainability from two perspectives; evolvability and stability. Evolvability is the ability to sustainably grow the project and stability is the ability to maintain the current project. We also consider the relationships between OSS projects because the resources that developers can devote to OSS projects are limited and OSS projects compete for developers and their contributions. We introduce magnet and sticky metrics from social sciences and demonstrate their applicability for understanding OSS sustainability. Magnet is a measure of attracting new developers and corresponds to evolvability. Sticky is a measure of retaining existing developers and corresponds to stability. By comparing these metrics between projects, we classify the projects into four categories (attractive, fluctuating, stagnant, and terminal) and show the transitions of categories with time.

Furthermore, we focus on core developers who play important roles in OSS projects. To investigate the impact of core developers on sustainability of OSS projects, we revisit the findings of prior studies.

The contributions of this dissertation are as follows.

(1) We introduce the magnet and sticky metrics for understanding OSS sus-

tainability. We apply the magnet and sticky metrics that were originally proposed in social sciences to the OSS context and show the applicability of these metrics in understanding sustainability of OSS projects.

(2) We empirically evaluate the availability of the magnet and sticky metrics using a large number of GitHub projects. To show the availability of the magnet and sticky metrics based approach, we perform an empirical study with a large number of GitHub projects. We find that 53% of terminal projects (low magnet/low sticky) eventually decay into a state of less than ten developers and 55% of attractive projects (high magnet/high sticky) maintain their popularity.

(3) We revisit prior findings about core developers by performing large empirical study. There are claims in literature about proportions of core developers in successful OSS projects; however, the claims are based on only a few case studies (at most 9 projects). We empirically evaluate the claims by analyzing a large number of GitHub projects.

This dissertation is organized as follows. Chapter 1 gives the background of the work and an overview of this dissertation. Chapter 2 presents a survey of literature related to sustainability of OSS projects. In Chapter 3, we introduce the magnet and sticky metrics to the OSS environment and show the availability of these metrics in a small sample of GitHub projects. Chapter 4 shows the result of a large empirical study on sustainability of OSS projects using the magnet and sticky metrics. Chapter 5 presents an empirical study of the impact of the ratio of core developers on sustainability of OSS projects. In Chapter 6, we discuss results of our study and show additional results for future direction. Finally, Chapter 7 presents a summary of this dissertation.

Acknowledgement

During this work, I have been fortunate to have received assistance from many individuals. To the following people, I owe an enormous debt of gratitude. Without these people this dissertation would never have been possible.

First and foremost, I would like to express my sincere gratitude to my supervisor, Professor Naoyasu Ubayashi for his supervision and guidance for my research. Without his help, I could not successfully accomplish the doctoral degree.

I would like to express my sincere appreciation to Professor Katsuro Inoue, in Osaka University, for his valuable comments and helpful suggestions on this dissertation.

I would like to express my sincere gratitude to Professor Akira Fukuda for his helpful comments, valuable questions, and kind advice for my work.

I also would like to express sincere appreciation to Professor Jianjun Zhao for his valuable questions and discussions for this work.

I would like to express sincere gratitude to Associate Professor Kenji Hisazumi for his valuable questions and discussions for this work.

I also would like to express a deep appreciation to Associate Professor Yasutaka Kamei. His zealous coaching and support have strongly encouraged me, and his helpful comments and suggestions made it possible to complete this work.

A special thanks to Professor Ahmed E. Hassan, Shane McIntosh and Meiyappan Nagappan. Their technical and editorial advice was essential to the completion of this dissertation. They have taught me innumerable lessons and insights on the workings of academic research in general.

I would like to express my appreciation to the clerks of our laboratory, including Ms. Minori Yoshio and Ms. Aya Miura. Their kind support has been quite helpful for me to prepare this dissertation.

I want to thank to members of Principles of Software Languages Laboratory (POSL) and Software Analysis and Intelligence Laboratory (SAIL) who work with me.

Finally, without the support of my family and friends, this dissertation would not have been possible.

Related Publications

The following publications are related to this dissertation:

- **(Chapter 3) Magnet or Sticky?: An OSS Project-by-Project Typology.**
  Kazuhiro Yamashita, Shane McIntosh, Yasutaka Kamei, and Naoyasu Ubayashi. In Proceedings of the 11th Working Conference on Mining Software Repositories (MSR), pages 344–347, 2014.

- **(Chapter 5) Revisiting the Applicability of the Pareto Principle to Core Development Teams in Open Source Software Projects.**
  Kazuhiro Yamashita, Shane McIntosh, Yasutaka Kamei, Ahmed E. Hassan, and Naoyasu Ubayashi. In Proceedings of the 14th International Workshop on Principles of Software Evolution (IWPSE), pages 46–55, 2015.

- **(Chapter 4) Magnet or Sticky?: Measuring Project Characteristics from the Perspective of Developer Attraction and Retention.**
  Kazuhiro Yamashita, Shane McIntosh, Yasutaka Kamei, Ahmed E. Hassan, and Naoyasu Ubayashi. Journal of Information Processing, 24(2):339–348, 2016.

The following publications are not directly related to the material in this dissertation, but were produced in parallel to the research performed for this dissertation.

- **Studying Just-In-Time Defect Prediction using Cross-Project Models.**
  Yasutaka Kamei, Takafumi Fukushima, Shane McIntosh, Kazuhiro Yamashita, Naoyasu Ubayashi, and Ahmed E. Hassan. Journal of Empirical Software Engineering,

Volume 21, Issue 5, pages 2072–2106, 2016.

- **Thresholds for Size and Complexity Metrics: A Case Study from the Perspective of Defect Density.**

  Kazuhiro Yamashita, Changyun Huang, Meiyappan Nagappan, Yasutaka Kamei, Audris Mockus, Ahmed E. Hassan and Naoyasu Ubayashi. In Proceedings of the 2016 IEEE International Conference on Software Quality, Reliability & Security (QRS), pages 191–201, 2016.

- **An Empirical Study of Just-In-Time Defect Prediction Using Cross-Project Models.**

  Takafumi Fukushima, Yasutaka Kamei, Shane McIntosh, Kazuhiro Yamashita, and Naoyasu Ubayashi. In Proceedings of the 11th Working Conference on Mining Software Repositories (MSR), pages 172–181, 2014.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

Open Source Software (OSS) is computer software with source code that anyone can study, change, and enhance [5, 56]. The OSS development mainly relies on volunteer developers [76], although recently there have been cases of employed developers [55, 79] and commercial OSS entitles [78]. Figure 1.1 shows an overview of the lifecycle of the bazaar style development, which is a typical OSS development style. Community of users and developers submit various types of contributions (e.g., modification, bug fix and bug report) to the community and the contributions are integrated into the implementation after review by developers (mainly core developers).

Since OSS is available for free and for any purpose by following the license, not only individual users but also enterprises use OSS for reducing costs such as license fees and development costs. In particular, large system integrators (individuals or enterprises that build information systems for their clients by combining subsystems of hardware and software) and solution providers (individuals or enterprises that address problems of their client by building and maintaining information systems) stand to gain the most from OSS systems because they increase profits thorough direct cost savings and the ability to reach more customers through improved pricing flexibility [78]. According to a survey conducted by

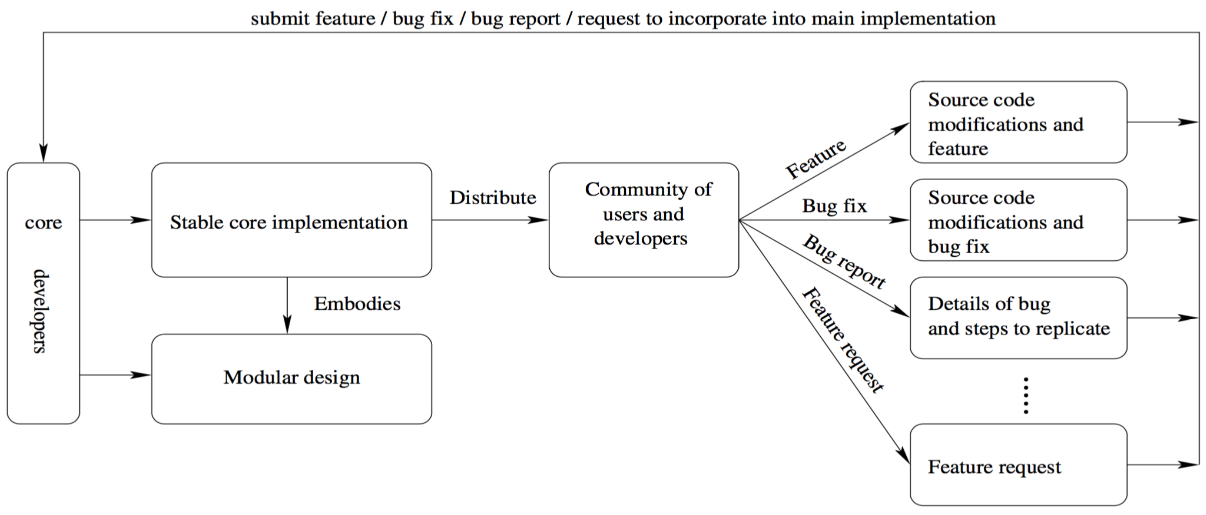

submit feature / bug fix / bug report / request to incorporate into main implementation
core developers
Stable core implementation
Embodies
Modular design
Distribute
Community of users and developers
Feature
Bug fix
Bug report
Feature request
Source code modifications and feature
Source code modifications and bug fix
Details of bug and steps to replicate
Feature request

Figure 1.1: Bazaar style development lifecycle [87]

Black Duck Software, which is a software company that assists companies to secure and manage their usage of OSS, 78% of enterprises run a part or all of their business operations on OSS systems.[1] In addition to the survey by Black Duck Software, another survey conducted by Information-technology Promotion Agency, Japan (IPA), shows that 66.8% of Japanese enterprises have introduced OSS systems [43].

As many enterprises use OSS systems, sustainability (i.e., the ability of an organism or an ecosystem to maintain its activity and productivity over time [9]) of OSS projects has become one of the most important problems. If the introduced OSS projects are abandoned, enterprises need to maintain the OSS systems on their own or replace them, and maintaining or replacing OSS systems takes additional costs. In fact, some enterprises hesitate to introduce OSS because of the concern "It is unclear how long the OSS project will be developed and maintained" [43]. Since OSS developers can leave the projects freely [25, 60], there is always the possibility that the activities for developing and maintaining the projects will stop. Furthermore, prior studies indicate that a large number of OSS projects fail (i.e., there is no activities) [8, 54]. Hence, in this dissertation, we focus on sustainability of OSS projects to avoid selecting projects whose maintenance and development will stop in near future.

[1]`https://www.blackducksoftware.com/future-of-open-source`

In this dissertation, we consider sustainability from the following two perspectives: evolvability (i.e., the ability to sustainably grow the project) and stability (i.e., the ability to sustain the current project). According to Kraut *et al.* [53], turnover of members in communities is inevitable. If a community does not obtain new members to substitute those who leave the project, the community is eventually abandoned. New members can also be a source of innovation, new ideas, work procedures, and other resources that the group needs. According to Robles *et al.* [80], most developers' contribution is only temporary. A lack of sustained developers (i.e., lack of stability) not only threatens the quality [94] and the release schedule [41] of an OSS project but also its entire existence [11, 86]. Therefore, the measure of sustainability of OSS projects should include the two perspectives of evolvability and stability.

For a better understanding of sustainability of OSS projects, it is also important to consider relationships between projects. Developers can participate in multiple projects [95] and their resources for OSS development are limited. Since developers want to devote their resources to projects that are more attractive to them, being more attractive than others may affect the sustainability of the projects.

In this dissertation, to measure the sustainability of OSS projects with consideration for the above-mentioned perspectives and relationships, we introduce the magnet and sticky metrics from social sciences [70] and demonstrate their applicability in the OSS field. These metrics are based on the transition of population between states and were used originally to categorize states in the United States of America into five types (e.g., high magnet/low sticky and neither here nor there). In the original study [70], magnet states are those in which a high share of adults who live there now have moved there from some other state and sticky states are those in which a high share of adults who were born there now live there.

In our study, the magnet and sticky metrics are based on the transition of developers between projects. For projects, new developers are a source of innovation, new ideas, and work procedures [53]. The magnet metric shows the ability to attract new developers and

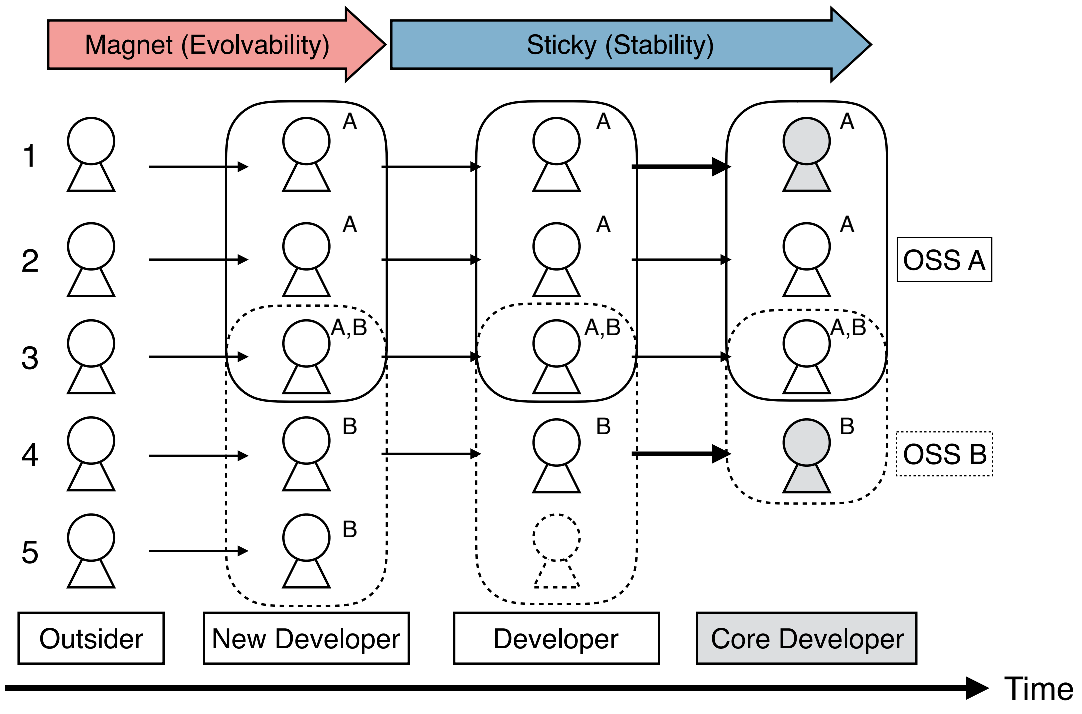


Figure 1.2: An overview of our study

corresponds to evolvability. For projects, losing existing developers affects performance and the quality of the work because of losing experience [42]. The sticky metric shows the ability to retain existing developers and corresponds to stability. We measure sustainability of OSS projects by using a combination of the magnet and sticky metrics.

Furthermore, we focus on core developers in a deeper analysis. Core developers are more active and play important role in OSS projects than other developers [64, 65]. Losing core developers has a larger impact on sustainability of OSS projects than losing other developers, so we decided to perform a deeper analysis focusing on core developers.

1.2 Overview of the Research

Now we present an overview of the dissertation. The dissertation presents a literature survey of related studies and the results of our three studies, all of which are closely related to sustainability of OSS projects. Before introducing our study, we show the results of a literature survey on sustainability of OSS projects. Our first study introduces the magnet and sticky metrics to understand project sustainability. Moreover, we demonstrate the applicability of the magnet and sticky metrics, which were originally proposed by Pew Research Center [70], in the OSS situation using a small set of sample projects. The second study expands the target projects of our first study to whole GitHub projects; then, we generalize the findings of the first study. The third study focuses on core developers and revisits a finding of prior studies about the proportion of core developers in successful (long-lived) OSS projects.

(Chapter 2) Literature Survey on Sustainability of Open Source Software Projects
We present a literature survey of researches from the perspectives of evolvability, stability, and core developers.

In this dissertation, we study evolvability using the magnet metric (attracting new developers) and stability using the sticky metric (retaining developers). Furthermore, we study the impact of core developers. In the following chapters, we describe our studies [105–107]. Figure 1.2 shows an overview of our study. The figure is based on the developer joining model, which is a model representing the stages and forces that influence the joining processes of a developer to an OSS project [91]. When developers are not joined with any projects, they are called outsiders. In the figure, three developers (1, 2, and 3) join OSS A and three developers (3, 4, and 5) join OSS B, and they are called new developers. As we explained above, developers can participate in multiple projects as developer 3 has done [95]. If the new developers continue their contribution, they come to be called developers. Developers can also leave the projects freely, as developer 5 has done. After sustained contributions, some developers become core developers in the projects (developers 1 and 4) [65].

(Chapter 3) Magnet or Sticky?: An OSS Project-by-Project Typology
This research introduces two metrics (i.e., magnet and sticky) into the software development context and demonstrates the applicability of these metrics with a small sample of GitHub projects. Using the MSR challenge dataset [31], which includes 90 sampled projects and fork repositories, we examine the applicability of the concept. Magnet projects are those where

a large proportion of the developers are new. Sticky projects are those in which a large proportion of developers have made contributions in the prior year. By observing transitions of the two metrics values and the state of projects, we capture the relationships between the metrics and the actual state of projects (i.e., the project the became popular or obsolete).

Our results showed the following findings:

- Stickiness is a more common project attribute than magnetism. Furthermore, we find that sticky projects such as `django` have a characteristic that they are used professionally by the most loyal developers and magnet projects such as `Homebrew` have simple contribution processes.
- 23% of terminal quadrant (i.e., low magnet and low sticky) projects eventually decay into a state where they have less than ten developers. Projects that are classified into other three quadrants do not decay into this state.
- 50% of attractive quadrant (i.e., high magnet and high sticky) projects remain in the attractive quadrant, suggesting that our quadrant analysis can successfully identify projects at the risk of becoming obsolete.
- Many quadrant transitions are accompanied by interesting events in a project's history.

From the second findings, projects with low magnet and low sticky values have a risk of becoming obsolete. On the other hand, if the magnet or sticky value is high, the projects retain their members. These findings suggest that combination of evolvability and stability is important for understanding sustainability of OSS projects.

(Chapter 4) Magnet or Sticky?: Measuring project characteristics from the perspective of developer attraction and retention

Since the results of Chapter 3 are based on 90 GitHub projects, the results are not sufficiently general. Therefore, in this research, we generalize the findings of the first study with all GitHub projects (16,552 projects are selected from 8,510,504 repositories using criteria described in Chapter 4).

The findings are as follow:

- Larger projects attract and retain more developers than smaller projects. 23% of developers remain with the same project irrespective of size (total number of developers), and new developers tend to join popular projects.
- 53% of the terminal projects eventually decay into a state of less than ten developers, while 55% of the attractive projects maintain their popularity.
- Only 13% of the projects in the first time period maintained ten or more developers in the second period.

These findings confirm our findings in Chapter 3 since these findings are similar to the findings in Chapter 3. For instance, in Chapter 3, we find that projects only from the terminal quadrant decay into a state of less than ten developers. In this chapter, we find that even though projects decay into such a state, the proportion of terminal projects (53%) that decay into that state is much higher than that of other projects (attractive: 3%, fluctuating: 8%, and stagnant: 28%). We also find that 55% of attractive projects remain in the quadrant, similar to the result of Chapter 3.

(Chapter 5) Revisiting the Applicability of the Pareto Principle to Core Development Teams in Open Source Software Projects

This research revisits the findings about the proportion of core developers in OSS projects. Our study finds that retaining existing developers is important for the sustainability of OSS projects. Since the proportion of core developers is one of the most popular metrics studied by previous works [19, 28, 30, 52, 64, 65, 83], we assume that core developers are more relevant to the sustainability than all developers. Prior studies have claimed that the proportion of core developers in successful (long-lived) projects follow the Pareto Principle [30, 52, 83]. Since prior studies were done on only 1–9 case study systems, their findings are not sufficiently general. Hence, in this research, we set out to replicate prior studies with a large set of GitHub projects.

The findings are as follow:

- Contrary to prior works, we find that there are several projects with a larger or smaller proportion of core developers than the one considered to be compliant with the Pareto principle. Moreover, we find that the core team of most projects has 15 or fewer members.
- The proportions of contributions of core and non-core developers are similar.

The results show that proportion of core developers doed not follow the Pareto principle in contrast to prior studies. These findings suggest that it is difficult to infer the sustainability of the projects based on whether or not the proportion of core developers follows the Pareto principle.

1.3 Dissertation Contribution

The major contributions of this dissertation are as follows:

- We introduce the magnet and sticky metrics to quantitatively evaluate project sustainability. For stability, we study the sticky metric, which is calculated as the proportion of developers who are retained in the same project. For evolvability, we study the magnet metric, which is calculated as the proportion of new developers who participate in a target project among all new developers. (Chapter 3, 4)
- We empirically demonstrate the applicability of the magnet and sticky measures with a large set of GitHub projects. We find that 53% of the projects with smaller magnet and sticky values than the median values of those metrics eventually decay into a state of less than ten developers. On the other hand, 55% of the projects with larger values than the median values maintain their popularity. These findings suggest that by using both evolvability and stability measures, it is possible to capture features of sustainability of

OSS projects. Moreover, it is possible to quantitatively measure the evolvability and stability of OSS projects by using these metrics. (Chapter 4)

- We empirically study the impact of the structural ratio of core developers on sustainability of OSS projects. In contrast to prior findings, we find that core team proportions do not follow the Pareto principle. Moreover, we find that the core team of most projects has 15 or fewer members. (Chapter 5)

1.4 Dissertation Organization

The remainder of the dissertation is organized as follows.
Chapter 2 presents the survey results on literature related to sustainability of OSS projects. In Chapter 3, we introduce the magnet and sticky metrics and show their applicability using a sample of projects. In Chapter 4, we generalize the results of Chapter 3 using all GitHub projects. Chapter 5 presents an empirical study of the impact of ratio of core developers on sustainability of OSS projects. In Chapter 6, we discuss our results and future directions of the research. Finally, Chapter 7 presents conclusions of this dissertation.

Chapter 2

A Literature Survey on Sustainability of Open Source Software Projects

2.1 Introduction

Open Source Software (OSS) development mainly relies on volunteer developers. Many unspecified developers join, collaborate, and develop OSS, Eric Raymond called the development style as “bazaar style” [76]. The aspects of the bazaar style are as follows [25, 60]:

1. Developers can join and leave from OSS projects freely.

   Although the software is developed mainly by the developers who start the project, the number of developers increases with growth of the project and large projects have more than 200 developers working on them [67].

2. Developers join OSS projects as primarily as volunteers.

   Developers employed in OSS projects are rare and most developers join OSS projects for collaborating with developers all over the world and satisfying their intellectual curiosity by, for example, studying and sharing development techniques [6, 16, 17].

3. There is no strict chain of command in OSS projects.

   As we mentioned in 1, developers can join and retire freely. Therefore, there is no strict chain of command, and developers contribute according to their own wishes similar to buying and selling in a bazaar [76].

4. User interactive communities.
   Users play an important role in OSS communities. The feedback from users is evaluation in an enterprise situation. The user feedback lets developers feel that they are contributing to meet social needs [76]. Moreover, reporting defects from users is important for OSS communities.

5. Distributed development environment through networks.
   In OSS projects, developers distributed geographically collaborate with each other through networks. In such a situation, non-face-to-face and non-synchronous communication such as mailing lists (ML) and boards is only way for communication. Therefore, the communication medium has a large impact on the communication networks [108].

2.1.1 Sustainability of OSS Projects

Because of concerns stemmed from aspects explained in Section 2.1, some companies hesitate to apply OSS systems even if there are advantages of introducing OSS systems. Examples of such concerns include "It is unclear which OSS system should be introduced because there are too many systems." "It is difficult to evaluate OSS systems and their maturity." "It is difficult to garner technical support when problems occur." and "It is unclear how long the OSS project will be developed and maintained" [43].

In this dissertation, we focus on the concern of "It is unclear how long the OSS project will be developed and maintained," i.e., sustainability of the OSS project, because this concern is the basis of other concerns. For instance, sustainability is one of the reasons of selecting a particular OSS system and one of the points of evaluation. This concern stems from the second aspect that we listed in Section 2.1. Developers being able to leave projects freely means the projects can stop when the developers leave. Therefore, several studies have focused on sustainability of OSS projects.

Sustainability is generally defined as the ability of an organism or an ecosystem to maintain its activity and productivity over time [9]. For maintaining activity and productivity over time, maintaining enough resources (i.e., developers and their contributions) for the activity and productivity is important. In this dissertation, we consider sustainability from the perspectives of evolvability and stability perspectives.

Evolvability is the ability to sustainably grow the projects. One of the factors related to evolvability is new members. New members can be a source of innovation, new ideas, and work procedures or the resources that the group needs [53].

Stability is the ability to maintain the projects in their current states. One of the factors related to stability is the retention of existing developers. High retention of existing developers might lead to a stable project because a lack of sustained developers threatens a project's existence [11, 86]. Additionally, high retention of core developers has more impact on the sustainability because core developers implement most of the new functionality [64] and are responsible for guiding and coordinating the development of an OSS project [65] .

To build a suitable measure for sustainability of OSS projects, we also need to consider the context. There are many current OSS projects in the world, many projects haring similar goals and features. Even though developers can participate in multiple projects [95], resources that the developers can devote to OSS activities are limited. Hence, developers prioritize OSS projects in line with their interests and purposes.

From the viewpoint of OSS projects, developers and their contributions are necessary to sustain the projects. To attract developers and their contributions, it is important that a project is developer-friendly (e.g., well documented and easy to extend) compared with other similar projects and such developer-friendly characteristics would lead to sustainable projects. Therefore, we assume that the measure should include information such as the relationship between projects.

Table 2.1: An overview of prior studies about evolvability

Paper	Target	Metrics	Methodology	Relationship	Dataset
Colazo; 2009 [11]	Attract developers	License	Regression Model		62
Meirelles; 2010 [62]	Attract developers and users	Source code metrics	Regression Model		6,773
Santos; 2013 [85]	Attract new developers and users	OSS Characteristics	Structural Equation Modeling		4,000 <
Stewart; 2002 [92]	Attract users	Intended audience, License, Development Status and Sponsorship	ANCOVA		240
Ververs; 2011 [97]	Attract new developers	Events, releases, issues and services	Correlation		1
Wu; 2007 [103, 104]	Attract users	Communication pattern	Three-Stage Least Squares		56
Chengalur-Smith; 2003, 2010 [8, 9]	Number of artifacts	Developer Attraction	PLS Model	✓	2,772
Raja; 2012 [74]	Sustainability	Vigor, Resilience and Organization	Regression Model		290
Kraut; 2012 [53]	Dealing with new developers	-	Survey		-

2.1.2 Organization of Chapter

This chapter is organized as follows. In Section 2.2, we show prior works on sustainability of OSS projects from the evolvability perspective. Section 2.3 presents prior studies on sustainability from the stability perspective. Section 2.4 focuses on core developers. Finally, we summarize the chapter in Section 2.5.

2.2 Evolvability

In this section, we discuss prior studies focusing on evolvability. As we explained in Subsection 2.1.1, one of the factors related to evolvability is new members (i.e., developers and users). New developers are a source of innovation, new ideas, and work procedures [53], and a significant fraction of OSS projects fail due to their inability to attract a sufficient number of developers [57]. It is important for sustainable growth to attract new developers and users.

Table 2.1 an overview of shows prior studies focusing on evolvability. The prior studies can classified into two types. One is studying factors that affect attracting developers, users, and downloads (as a proxy of number of users). The other is investigating the impact of attraction of developers or users on development activities (e.g., number of closed and opened artifacts).

First, we discuss studies on factors that affect attracting new developers and users. Colazo and Fang [11] investigate the relationships between licenses and development activities. They divide licenses into two types (i.e., Copylefted and Noncopylefted) and study the effects on development activities such as total number of developers, coding activity, and development speed. Their results show that the total number of developers, coding activity, and speed are higher in copylefted OSS projects than in noncopylefted projects. Hence, the type of license affect the total number of developers.

Meirelles *et al.* [62] focus on the relationships between source code metrics (e.g., LOC and number of methods) and attractiveness (i.e., number of downloads and number of members). The results of their analysis of over 6,000 projects support two hypotheses. "OSS projects with higher structural complexity have lower attractiveness" and "OSS projects with more lines of code have higher attractiveness".

Santos *et al.* [85] consider attractiveness as an array of project values perceived by its potential and actual visitors, users and developers. As the indicators of visitors, users, and developers, they use number of hits of the project website, number of downloads, and number of registered members, respectively. One of their findings is that life-span (i.e., age) is a positive and statistically significant predictor of attractiveness.

Stewart *et al.* [92] determine factors that affect the Vitality (the number of announcements about a project and the time since its last release, i.e., project activity) and Popularity (the number of people who subscribe to the project as well as hits to the website, i.e., users) of OSS projects by conducting an exploratory study on a sample of 240 OSS projects listed on the *www.freshmeat.net.* They consider Vitality as an indicator of how much developer effort and attention is expanded on a project and Popularity as an indicator of how much user attention is focused on a project. From the result of ANCOVA analysis, they find that Vitality, development status, sponsorship, and category have a significant effect on Popularity.

Ververs *et al.* [97] focus on the factors affecting developer participation in the Debian project. By using correlation, they find that the most important factors are specific events such as CeBIT[1] (a computer exhibition) and Debian Day[2] (a day on which several speakers talk about current developments in Debian), new or stable releases, issues and the introduction of new developer services.

Wu *et al.* [103, 104] investigate the effect of communication patterns on the sustainability of OSS. Their findings suggest that communication patterns do not affect user attraction. They use Project Centrality and Project Density as indicators of communication patterns. Their results show that the communication patterns have significant effects on development activity (i.e., number of closed bug), but not on popularity (i.e., number of downloads). Angel Mary *et al.* [59] propose a communication validation tool as they consider effective communication between the OSS developer communities is one of the signposts that contribute to OSS success.

Next, we discuss studies that investigate the impact of evolvability factors such as attracting developers and users to sustainable development activities. Chengalur-Smith *et al.* [8, 9] investigate what characteristics of projects and projects' ecological environments influence their future sustainability by analyzing 2,772 OSS projects. In their study, they define a sustainable project as one that exhibits software development and maintenance activity over the

[1]`http://www.cebit.de/en/`
[2]`https://wiki.debian.org/DebianDay`

long run. They measure different factors in three different periods. In the first period, they measure Development Base (i.e., number of developers), Age and Niche Size (i.e., Audience Niche, Programming Language Niche and Operation System Niche). In the second period, they measure Developer Attraction and User Attraction. In the last period, they measure Sustainability (i.e., number of artifacts opened and closed). From the empirical study using a partial least squares (PLS) model, which is a regression-based structural equation modeling (SEM) technique [10], they find that Developer Attraction has the largest effect on project sustainability.

Raja and Tretter [74] propose a metric (Viability Index: VI) to measure project sustainability. The metric consists of three dimensions, i.e., Vigor (the ability to evolve), Resilience (the ability to respond to internal external perturbations), and Organization (the amount of structure in a project), which is calculated as $VI = -3.834 + 0.36Vigor + 0.232Resil + 0.5Org$.

Although many of the prior studies focus on new developers and users, they count new developers in a single project (i.e., target project). On the other hand, magnet is calculated as the proportion of new developers working on a target project among all new developers. Moreover, in prior studies, only Chengalur-Smith *et al.* [8, 9] use an indicator (Niche size) that considers the relationships between projects (relationship column in Table 2.1). In our study, we divide projects into four categories by comparing their magnet and sticky values to consider the relationships between the projects.

2.3 Stability

In addition to evolvability, stability is also important for sustainability [86]. Table 2.2 shows an overview of prior studies focusing on stability.

First, we discuss studies focusing on factors that affect retention of developers and users. Colazo *et al.* [11] investigate the relationship between license type and retention of developers as well. From their result, retention of developers is higher in projects that apply noncopylefted license, in contrast to their result on attraction of developers.

Table 2.2: An overview of prior studies about stability

Paper	Target	Metrics	Methodology	Relationship	Dataset
Colazo; 2009 [11]	Retention of developers	License	Regression Model		62
Zhou; 2012 [111]	Retention of new developers	Willingness, Macro-climate and Micro-climate	Regression Model		2
Fang; 2009 [24]	Sustained participation	Situated Learning and Identity Construction	Questionnaires		1
Qureshi; 2011 [73]	Sustained participation	Level of Socialization	Growth Mixture Modeling		40
Schilling; 2012 [86]	Sustained participation	Person-Job and Person-Team	Regression Model		1
Chengalur-Smith; 2003, 2010 [8, 9]	Number of artifacts	Development Base	PLS Model	✓	2,772
Samoladas; 2010 [84]	Survival (Existence of commit activities)	Number of developers	Survival Analysis		1,147
Midha; 2007 [63]	Retention of developers	Complexity and Modularity	Regression Model		70
Rastogi; 2014 [75]	Sustained participation		ARIMA model		1
Sharma; 2012 [88]	Turnover	Role of the developer, number of projects, past activity, tenure, project age and number of developers	Hierarchical Linear Model	✓	40
Steinmacher; 2013 [91]	Retention of new developers	-	Questionnaires		1

Zhou and Mockus [111] investigate the factors that make a new joiner into a long-term contributor (LTC). From the case study on Gnome and Mozilla projects, they find that the differences between developers are in their capacity, willingness and opportunity to contribute when they join. More specifically, the most important factor is the pro-community attitude represented by the first contribution. Von Krogh *et al.* [98] found that new developers to the Freenet project will more likely undertake certain actions than long-term developers.

Fang and Neufeld [24] also study sustained participation in OSS projects. By applying the theory of legitimate peripheral participation, they find that situated learning (the process of acting knowledgeably and purposefully in the world) and identity construction (the process of being identified within the community) are positively linked to sustained participation.

Qureshi and Fang [73] claim that motivating, engaging, and retaining new contributors promotes a sustainable community. Steinmacher *et al.* [91] find that the absences of response, politeness, usefulness, or the author of answers influence the retention of new developers.

Schilling *et al.* [86] predict which developer will be retained using the Person-Job (P-J) fit [22], which describes the suitability of an individual for a particular job and the Person-Team (P-T) fit [101], which describes the rational compatibility between an individual and the existing team. From their experiments, they find that level of relevant development experience is strongly associated with developers' project retention and familiarity with the coordination practices of the project team has a strong association with the retaining time.

There are also studies focusing on the impact of stability factors on sustainability. Since the number of developers is one of the basic indicators of stability, many studies have used that as a metric. Chengalur-Smith *et al.* [9] investigate the relationship between sustainability and development base (i.e., number of developers). From their results, there is no significant direct effect of the development base on sustainability, even though the development base affects developer attraction.

As another example, Samoladas *et al.* [84] apply survival analysis, which analyzes the expected time until one or more events such as death happen, to OSS projects and try to estimate future development of OSS projects. One of their findings is that the number of developers affects the sustainability of the project. More specifically, if the project has more than 20 developers, the probability of survival does not fall below 80%.

There are studies that focus on the opposite phenomenon (i.e., developer turnover and project termination). In contrast to studies on sustained participation, some studies focus on developer turnover [26, 44, 81, 88], since turnover may affect the performance and the quality of work because of losing experience [42]. According to Foucault *et al.* [26], turnover 1) has a positive impact since the most unsatisfied members leave the team and only the most motivated ones stay [18], 2) helps renew experience and knowledge on the team [45], and 3) increases social interactions [15]. Robles *et al.* [81] applied the concept of developer turnover to OSS projects. From a case study using long-lived projects with focus on core developers, they find that these projects suffered from yearly turnover in core teams and had to rely heavily on regeneration. Sharma *et al.* [88] examine developer turnover using the Hierarchical Linear Modeling (HLM) approach, which is an ordinary least square (OLS) regression-based analysis that takes the hierarchical structure of the data into account. They suggest that past activity, developer role, project size, and project age are the main predictors of turnover.

Many of the prior studies focus on how to retain existing developers. Some studies especially focus on new developers, since retaining new developers is more difficult than retaining experienced developers due to the lack of experience in the project [90]. Moreover, some studies focus on the opposite phenomenon, i.e., turnover of developers. For better understanding sustainability, the opposite phenomenon might be helpful.

Similar to evolvability studies, there are a few studies that consider relationships between projects (e.g., number of projects in Sharma *et al.*'s study [88]). Furthermore, to the best of our knowledge, there are surprisingly no studies focusing on the impact of the proportion of retaining developers on sustainability.

In this dissertation, we use the sticky metric as an indicator of stability and demonstrate

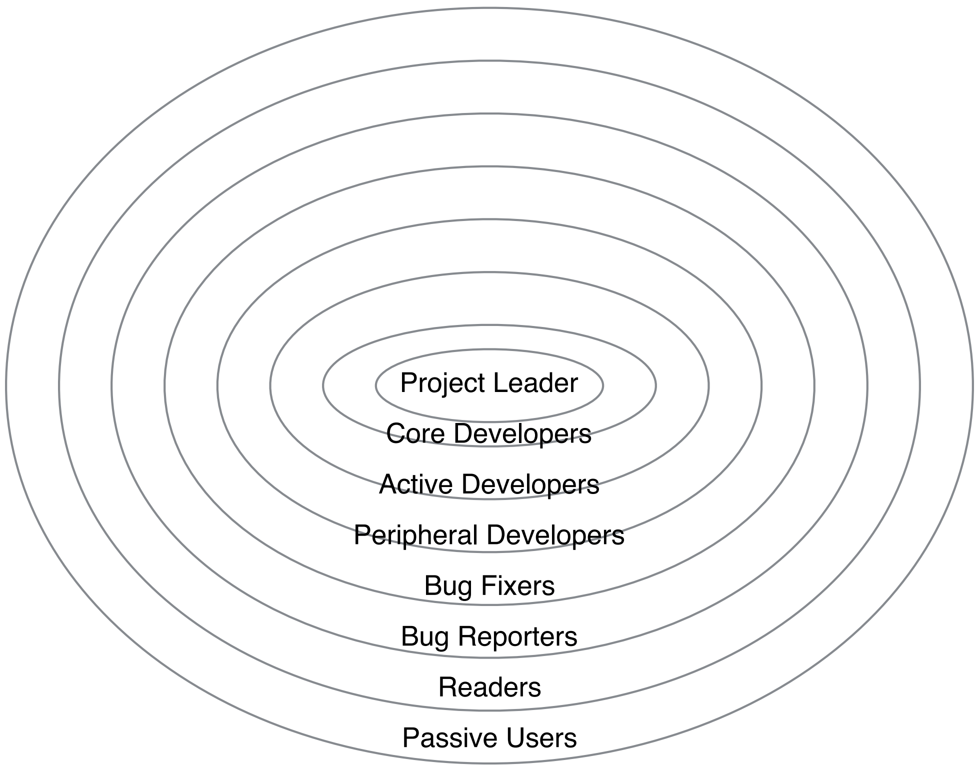


Figure 2.1: General structure of an OSS community based on the onion model [65]

the impact of the metric on sustainability. Furthermore, we consider the relationship between projects by comparing the values of sticky metric for the projects.

2.4 Core Developers

Although there is no strict chain of command, there are roles of developers (e.g., Leader, Core Developer and Peripheral Developer) and the developers who have more important roles may have larger impact to sustainability. Figure 2.1 shows a general structure of an OSS community based on the onion model [65].

In Chapter 5, we focus on the impact of proportion of core developers to the sustainability of OSS projects. Prior work has also analyzed the proportion of core developers in OSS projects. There are several definitions of core developer. Mockus *et al.* [64] hypothesize

Table 2.3: An overview of the results of prior work about core developers

Paper	Dataset	Result
Mockus *et al.* [64]	Apache and Mozilla	10 to 15 developers performed 80% of the contributions.
Dinh-Trong and Bieman [19]	FreeBSD	28 to 42 developers performed 80% of the contributions.
Koch and Shneider [52]	GNOME	52 developers (out of 301 developers) performed 80% of the contributions.
Goeminne and Mens [30]	Brasero, Evince and Wine	20% of developers performed 85%, 80% and more than 90% of the contributions in each project.
Robles *et al.* [83]	GNOME	The core group has been identified as the 20% most contributing committers.
Geldenhuys [28]	9 OSS projects	3%-9% of developers performed 80% of the contributions.

that (1) the open source development model would rely on a team of core developers who control code base and (2) these core developers create 80% or more of new functionality. Nakakoji *et al.* [65] state that core developers are responsible for guiding and coordinating the development of an OSS project.

Table 2.3 provides an overview of the results of the prior work and the datasets that were analyzed. Mockus *et al.* [64] hypothesize that the open source development model would rely on a team of core developers who control the code base and that these core developers would create 80% or more of the new functionality. Furthermore, Mockus *et al.* argue that the core team would be no larger than 10 to 15 people based on analysis of the Apache and Mozilla projects. Crowston *et al.* [14] compared three approaches to identify the core developers within 116 SourceForge projects using bug fixing activities. Although the results differ among the three studied approaches, all of the approaches indicate that the core developers make up a small fraction of the total number of contributors. Goeminne and Mens [30] found evidence for the Pareto principle in three activities (development, email discussion and bug tracker activity) in three OSS projects. Robles *et al.* [83] arrived at similar conclusions — the core team makes up roughly 20% of the contributing committers.

On the other hand, other studies arrive at contradictory conclusions. For example, Dinh-Trong and Bieman [19] replicated Mockus *et al.*'s work using data from the FreeBSD project, finding that 28-42 out of 161-265 developers perform 80% of the contributions. Koch and Shneider [52] find that 52 out of 301 developers make 80% of the contributions in the GNOME project. Through analysis of nine systems, Geldenhuys [28] finds that the proportion of core developers does not comply with the Pareto principle.

Much of the prior work analyzes a small number of subject systems. We assume that using small number of subject systems is the reason why prior works had contradictory evidences. Hence, we set out to analyze core teams in a large number of systems to find out the impact of proportion of core developers to sustainability of OSS projects in Chapter 5.

Some studies have investigated the role migration in OSS. Nakakoji *et al.* [65], Ye and Kishida [109] and Jensen and Scacchi [46] found that the extent to which each developer influences an OSS project establishes a hierarchy among the developers. Nakakoji *et al.* [65] claimed that a sustainable OSS project must evolve both the systems and the community. They identified three evolution patterns exploration-oriented, utility-oriented, and service-oriented. Ye and Kishida [109] sought to understand why people participate in OSS projects. They assume that learning in practice motivates OSS developers. Along with this learning process, a developer's role transformation in the OSS community provides extrinsic motivation. Jensen and Scacchi [46] investigated the role migration and project career advancement processes of OSS developers, focusing on three large OSS projects. They discussed the roles and layers in each projects and the migration between the roles and layers of developers who joined the projects.

Ducheneaut [20], Herraiz *et al.* [36], Bird *et al.* [3], and Shibuya and Tamai [89] also studied the role immigration process of OSS participants. Drawing on personal experience, Deucheneaut described the six steps toward becoming a Python developer. In their experiments on three large projects, Bird *et al.* found that a submission history of patch upgrades can effectively elevate a joiner to developer status. Herraiz *et al.* discovered two groups of role migration; volunteer developers who proceed in a step-by-step fashion, and sponsored developers who suddenly migrate their roles. Shibuya and Tamai studied the openness (transparency and accessibility [102]) of three projects. They found that each project facilitates participation of new developers in different ways.

In addition to the size of core teams and role migration in OSS projects, Mockus *et al.* [64] hypothesized that a group, which is larger by an order of magnitude than the core team, will repair defects. From this hypothesis, we derive that non-core developers focus more on maintenance activity (e.g., bug fixing) than implementation activity. Goeminne and Bieman [30] showed that 2-6 out of the top 20 developers also contributed to plenty of the bug report and email discussions.

Robles *et al.* [82], Hindle *et al.* [39], and Vasilescu *et al.* [96] studied the various activities in projects. Hindle *et al.* distinguished four types of files and Robles *et al.* proposed eight different activities. Recently, Vasilescu *et al.* extended this number to 14 activities and empirically studied how the workloads of projects/contributors varied across the software ecosystem.

Because of small number of case study systems, claims of proportion of core developers in successful projects vary. Our study generalize prior findings with large set of OSS projects which are hosted on GitHub. Furthermore, we quantitatively compare the contribution activity of core and non-core developers.

2.5 Summary

In this chapter, we discuss prior works related to this dissertation. From the results of this literature survey, we find that only a few studies consider relationships between projects to capture the situation. Our approach compares the values of the magnet and sticky metrics for

OSS projects. Furthermore, we combine the magnet and sticky metrics and do not consider these metrics individually.

For core developers, we focus on the proportion of core developers in projects as the first step of studying the impact of core developers on sustainability. According to the literature survey, we find that claims of prior studies about the proportion of core developers vary (e.g., some studies claims that the proportion follows the Pareto principle) because of the small number of case study systems. Hence, we generalize such observations about the proportion of core developers in projects with a large number of case study systems.

Chapter 3

Introduction of Magnet and Sticky For OSS Sustainability

3.1 Introduction

In this chapter, we introduce the magnet and sticky metrics and the way to measure the sustainability of open source software (OSS) projects, and demonstrate the applicability with a small sample of GitHub projects.

Using census data, the Pew Research Center, a research body that studies the issues, attitudes and trends shaping America and the world, launched a Social & Demographic Trends study. The study reports that just 28% of adults born in Alaska still live there. Furthermore, 86% of adult residents of Nevada migrated there from a different state, suggesting that Nevada is quite “magnetic”. Such population studies illustrate the migratory trends of citizens in America.

For software engineers, migratory trends of open source contributors is of interest. To become a popular and long-lived project, maintainers need to retain existing contributors and attract new ones. Mockus *et al.* [64] find that although the Apache and Mozilla open source projects have a small core team of developers, there is a larger community of contributors.

While prior work has explored contributor immigration [3] and Long Term Contributors (LTCs) [111], to the best of our knowledge, the "sticky" and "magnetic" nature of open source projects has not yet been explored. Hence in this study, we set out to adapt the "magnet" and "sticky" metrics described in the Pew Research study for the context of open source projects. We then use these metrics to explore the sticky or magnetic nature of open source projects in the MSR challenge dataset [31]. Using the dataset, we address two research questions that we describe in next section (Section 3.2).

3.1.1 Organization of Chapter

The remainder of the chapter is organized as follows. Section 3.2 provides our study design such as our definitions of magnet and sticky metrics for the open source context, our research questions and the areas of the MSR challenge dataset that we leverage for our study. In Section 3.3, we present the results of our study. Finally, Section 3.4 draws conclusions.

3.2 Study Design

3.2.1 Definition of Magnet and Sticky

The Pew Research Center report[1] defines *magnet states* as those states where a large proportion of adults who live there have moved from another state. Thus, the magnet metric for a state is the proportion of adult residents of a state who were not born in the state. Furthermore, the report also defines *sticky states* as those states where a large proportion of adults who were born there continue to live there. Thus, the sticky metric for a state is the proportion of adult residents who were born in the state.

These definitions are sound for a study of populations, where a single adult can only occupy one state at a time. However, the definition cannot be applied directly to open source projects, where a contributor can contribute to several projects at the same time. Therefore,

[1]`http://www.pewsocialtrends.org/2009/03/11/magnet-or-sticky/`

we expand original definition to apply to open source projects as follows:

Magnet projects are those that attract a large proportion of new contributors. Thus, we calculate the magnetism of a project as the proportion of contributors who made their first contribution in the time period under study who contribute to a given project.

Sticky projects are those where a large proportion of the contributors will keep making contributions in the time period the following and under study. Thus, we calculate the stickiness of a project as the proportion of the contributors in the time period under study who have also made contributions in the following time period.

While our definitions are based on contributors, in this study, we focus on developers. We plan to explore other types of contributions in future work. Furthermore, our definition is based on time periods in the abstract sense. We select development years as the granularity for this study because we believe that years provide a coarse enough granularity to observe high-level trends. Coarser or finer granularities can also be explored.

Figure 3.1 shows an example of our definition. In this example, we examine the 2011 time period. There are five developers (A, B, C, D and E) and two projects (1 and 2). To calculate the magnet metric, we observe that there are three new developers (B, C and D), and two of them contribute to project 1 (B and C), while one developer (D) contributes to project 2. In this case, magnet value of project 1 is 2/3 and project 2 is 1/3.

To calculate the sticky metric, in project 1, three developers contribute in 2011 (A, B and C) two of them also contribute in 2012 (A and B). Hence, the sticky value of project 1 is 2/3. In project 2, one developer contributes to the project in 2011 (D) and two developers (D and E) contribute to in 2012. However, the sticky metric only considers the number of developers who contribute to the project in the studied time period and the next one. Hence, sticky value of project 2 is not 2/1, but rather 1/1.

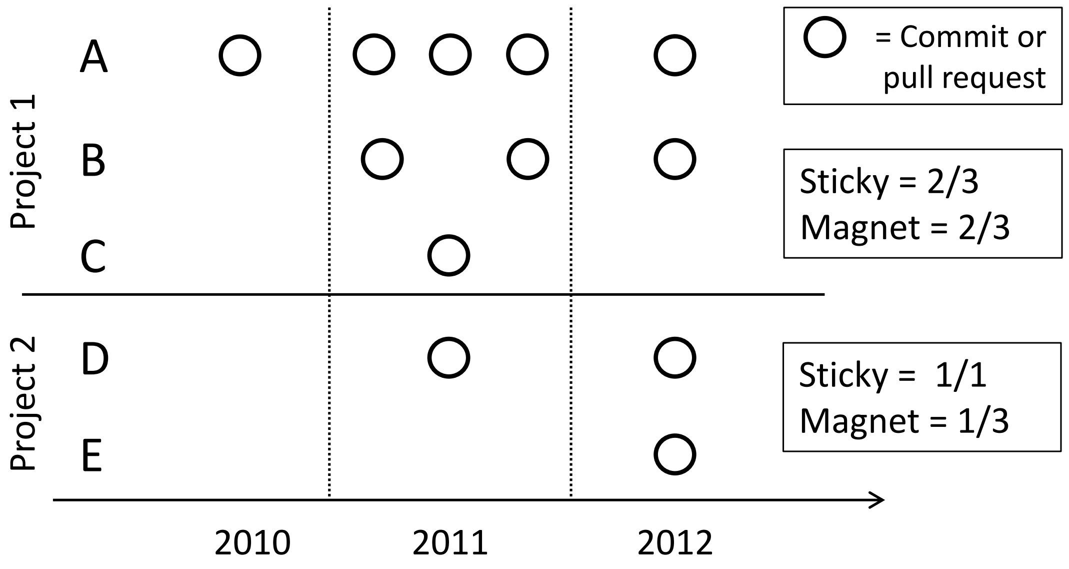


Figure 3.1: Example of magnet or sticky of values by our definition in 2011

3.2.2 Research Question

In this chapter, we first introduce magnet and sticky metrics to OSS context for understanding sustainability of OSS projects. Before starting deep analysis using these metrics, we need to evaluate the applicability to OSS context. Hence, we address the following two research questions.

(RQ1) What are typical values of magnet and sticky in OSS?

Although most of the studied projects tend to be more sticky than they are magnetic, highly magnetic projects like `Homebrew` consistently attract many developers by simplifying the contribution process.

(RQ2) How do magnet and sticky values change over time?

Our quadrant analysis can identify projects at risk of becoming obsolete. Furthermore, quadrant transitions are often accompanied by interesting events.

Table 3.1: An overview of the dataset collected using queries like: "select count(id) from TABLE"

# Users	# Projects	# Commits
499,485	108,718	555,325

3.2.3 Dataset

In this study, we analyze the GitHub dataset provided by Gousios [31]. The dataset includes a variety of software evolution data from 90 OSS projects, such as issues, pull requests, organizations, followers, stars and labels. However, in this study, we only operate on code authorship data in the commits and pull requests tables. Although these 90 OSS projects are forked thousands of times, we focus on the commit and pull request activity in the original repository.

An overview of the data we study is presented in Table 3.1. Note that each of the IDs are unique for each studied table. Therefore, Table 3.1 shows the number of unique users, projects and commits.

In this study, we consider a developer to be one who authors code changes to a project. In the GitHub dataset, a developer can either perform the commit himself or send a pull request to an upstream repository maintainer. We consider both actions as development activity for our magnet and sticky analysis.

3.3 Study Results

We now present the results of our study with respect to our two research questions. For each question, we discuss our approach, quantitative and qualitative results.

(RQ1) What are typical values of magnet and sticky in OSS?

Approach. We calculate magnet and sticky values for the studied OSS projects as described in Section 3.2. To visualize the data, we plot magnet and sticky values for each project against each other, and (similar to Khomh *et al.* [51]) divide the plot into four quadrants:

Attractive: Projects with high magnet and sticky values. Attractive projects are successful in both attracting new developers and retaining existing ones.

Fluctuating: Projects with high magnet values, but low sticky ones. Fluctuating projects are successful at attracting new developers, but unsuccessful at retaining existing ones.

Stagnant: Projects with low magnet values, but high sticky ones. Stagnant projects retain the existing development team well, but struggle to attract new members.

Terminal: Projects with low magnet and sticky values. Terminal projects struggle to retain existing developers and do not attract new ones.

The quadrant thresholds can be dynamically configured. In this study, we use the median magnet and sticky values as the thresholds, since the median is a robust measure that is not heavily influenced by outliers.

As described in Section 3.2, the sticky value of a year under study depends on the number of developers in that year and the following one. Hence, to address RQ1, we focus on the most recent year (2011) that has a complete year of historical data recorded after it (2012). We are not able to use the data of 2012 or other more recent years because the dataset only contains partial results from the 2013 (i.e., until October) and no results from 2014 yet.

Note that the sticky value depends on the number of developers who contribute in the target year (Figure 3.1). If the number of developers in the target year is small, the sticky value tends to be high. Therefore, to reduce the influence of noise on our results, we filter away projects that have ten or fewer developers.

Quantitative results. Figure 3.2 shows the magnet vs. sticky quadrant plot of OSS projects in 2011. Attractive projects land in the red quadrant (upper-right), fluctuating projects land in the green quadrant (upper-left), stagnant projects land in the blue quadrant (lower-right) and terminal projects land in the purple quadrant (lower-left).

Figure 3.2 shows that magnet values tend to be much smaller than the sticky values. Indeed, most projects have sticky values that are larger than their magnet values. In other

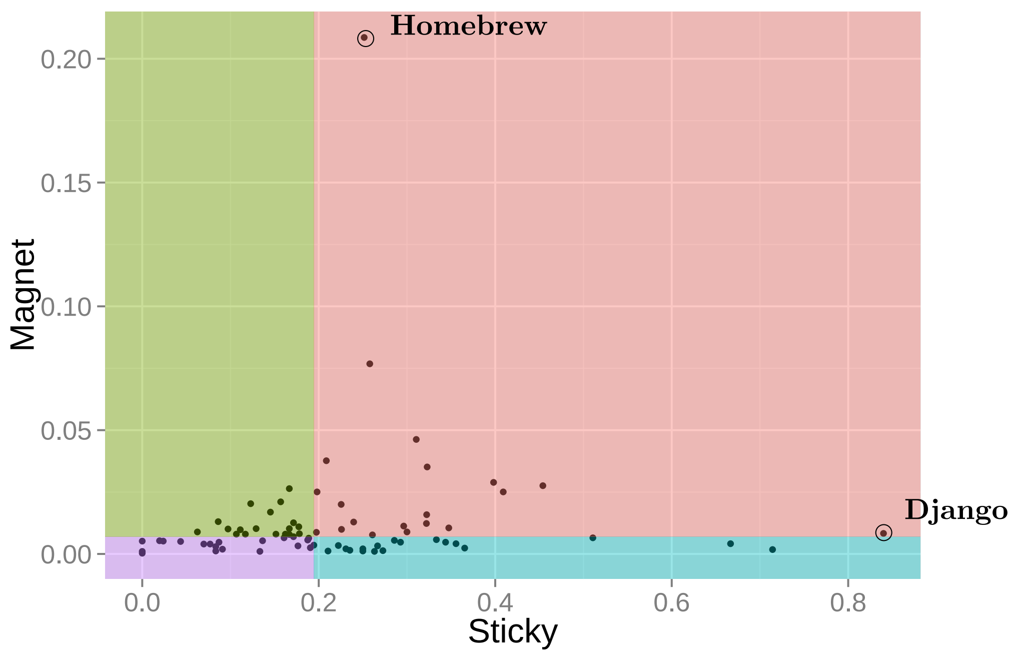


Figure 3.2: Distribution of magnet and sticky values for the studied projects

words, stickiness is a more common attribute of a software project than magnetism. This finding is in agreement with the original magnet vs. sticky study of American states, where the median of sticky value was 0.580 and the median of magnet value was 0.39.[2] Like citizens who become accustomed to their environment, developers who contribute to a project are likely to continue contributing to the same project.

Manual analysis. Figure 3.2 shows that there are projects with extreme magnet and sticky values. We manually inspect two such projects, i.e., the projects that have the highest magnet and sticky values.

The `Homebrew`[3] project has the largest magnet value in Figure 3.2. `Homebrew` is a popular package management tool for Mac OS X that began development in 2009. We suspect that in addition to `Homebrew`'s popularity, it is especially magnetic because it is relatively easy to contribute to it. For example, the smallest contribution that one can make to `Homebrew` is a “Formula cookbook”, i.e., a package description that specifies the URL of a package and how it can be installed. Since most package descriptions are written in high-level scripting

[2]`http://www.pewsocialtrends.org/2009/03/11/sticky-states/`
[3]`http://brew.sh/`

Table 3.2: Number of new developers and unique developers in each year

Year	New Devs	Unique Devs	Year	New Devs	Unique Devs
2003	17	17	2009	1,045	1,287
2004	9	21	2010	2,768	3,190
2005	45	54	2011	6,745	7,944
2006	55	91	2012	7,988	10,606
2007	66	128	2013	6,643	9,860
2008	483	565			

languages,[4] the barrier to entry for newcomers is relatively low.

The `django` project has the largest sticky value. `Django` is a high-level python web framework that began development in 2005. `Django` is used by several popular web applications, such as: (1) Instagram – a popular social image sharing application[5] and (2) ReviewBoard – a peer reviewing web application.[6] To investigate why the `django` project is so sticky, we studied the professional activity of the ten most active and loyal developers.

We find that although prior work by Zhou *et al.* reports that the most important factor to becoming a long term contributor to an open source project is a pro-community attitude [111], many of the top contributors are motivated by their professional activity. For example, many of the most loyal `django` developers began contributing during the initial development period or shortly after it. Of the ten top contributors, eight are paid to develop web applications professionally, and use `django` to do so. Their work on `django` is thus likely motivated by their professional activity.

Stickiness is a more common project attribute than magnetism. Especially sticky projects like `django` *are used professionally by many of the most loyal contributors. Especially magnetic projects like* `Homebrew` *have simple contribution processes.*

[4] https://github.com/Homebrew/homebrew/wiki/Formula-Cookbook
[5] http://instagram-engineering.tumblr.com/post/13649370142/what-powers-instagram-hundreds-of-instanc
[6] http://www.reviewboard.org/

Table 3.3: Quadrant transitions of long-lived open source projects.

Quadrant in 2011	Project Name	2004	2005	2006	2007	2008	2009	2010
	rails	*	Terminal	Stagnant	Stagnant	Fluctuating	Fluctuating	Attractive
Attractive	xbmc	-	Stagnant	Attractive	Attractive	Stagnant	Attractive	Terminal
	django	-	*	*	Stagnant	Stagnant	Stagnant	Stagnant
	jquery	-	-	Fluctuating	Terminal	*	Attractive	Fluctuating
Fluctuating	paperclip	-	-	-	*	Fluctuating	Terminal	Fluctuating
	compass	-	-	-	-	*	Fluctuating	Fluctuating
	scala	Terminal	Terminal	*	*	Attractive	Terminal	Fluctuating
Stagnant	memcached	*	*	Terminal	Fluctuating	Stagnant	Terminal	*
	clojure	-	-	*	*	*	Attractive	Attractive
	django-debug-toolbar	-	-	-	-	Terminal	Terminal	Terminal
Terminal	jekyll	-	-	-	-	Fluctuating	Fluctuating	Terminal
	blueprint-css	-	-	-	*	*	Terminal	*

(RQ2) How do magnet and sticky values change over time?

Approach. We first analyze how all of the studied projects transition among the quadrants of Figure 3.2 as they age. Since the boundaries of the quadrants will likely change, we recalculate the boundaries for each studied year.

Quantitative results. Figure 3.3 illustrates quadrant transition likelihood using a state transition diagram. Each value describes the likelihood of a transition from one quadrant to another (or the same) quadrant. The direction of the arrow indicates the direction of the quadrant change. For example, Figure 3.3 shows that the likelihood of transitioning from the attractive quadrant to the fluctuating one is 22%. We use the "*" state to represent the years when projects did not satisfy our filtering criteria (ten or more developers). To improve the readability of the figure, we plot two "*" states, however they are semantically identical.

This figure shows that terminal quadrant projects are the only ones that drop into the filtered away state ("*"). This agrees with our intuition, in that terminal quadrant projects are losing team members and struggling to attract new ones, which if continued for a long enough period of time would lead to the death of a project. Projects in the other quadrants are successful at either attracting new developers (high magnet) or retaining the existing ones (high sticky), and are unlikely to decay in size like those in the terminal quadrant.

We also find that fluctuating projects have same likelihood of transition to other three categories (i.e., attractive, stagnant, terminal). This also agrees with intuition, in that fluctuating projects have plenty of turnover in the development team, and hence could transition to any of the other quadrants at any time.

Figure 3.3 also shows that there is often a higher likelihood of transition from higher quadrants to lower ones than vice versa. This is likely due to the fact that increasing the stickiness or magnetic nature of a project requires effort to obtain (and retain) more developers. Nonetheless, in two out of nine cases (i.e., attractive-stagnant and fluctuating-terminal), projects are more likely to transition from the lower quadrant to the higher one than vice versa. While the difference in the fluctuating-terminal transition flow is minimal (i.e., only

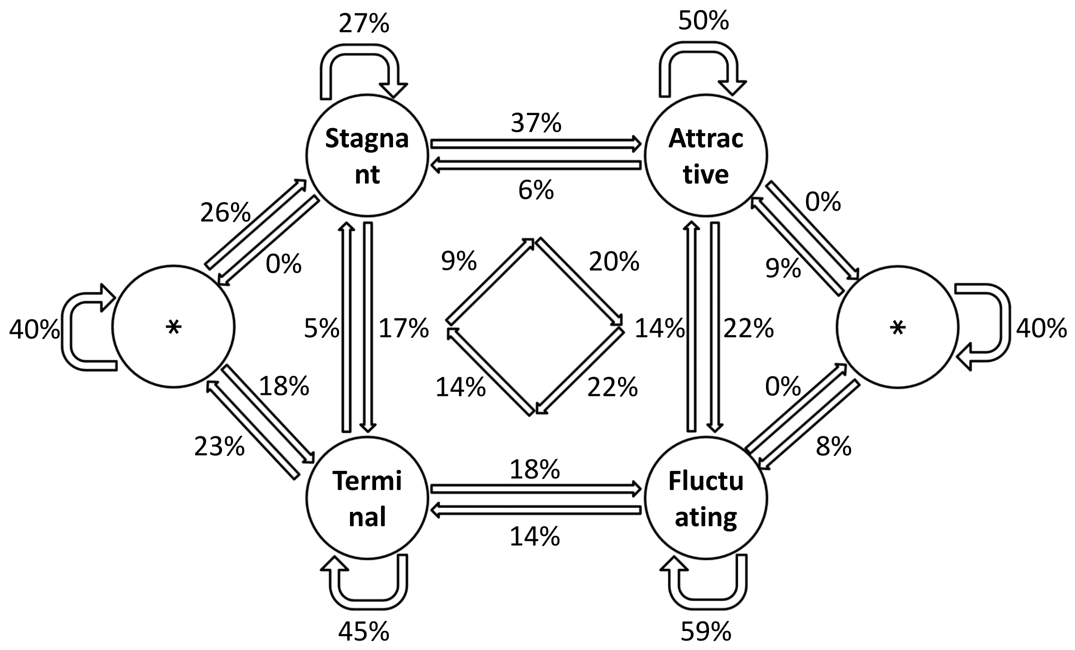


Figure 3.3: The likelihood of quadrant transitions

four percentage points differentiate the two directions), the attractive-stagnant case has a much broader separation. The reversal of transition flow in the attractive-stagnant case is likely because 70% of the studied projects only began development after 2009 (see Table 3.3). The short nature of much of the studied project history causes "trendy", short-lived, but highly popular applications to influence our results.

Furthermore, to confirm the risk of becoming obsolete, we check how many of the projects that enter the filtered state ("*") were in the terminal state just before. We find that two-thirds of filtered state projects originate from the terminal state.

Manual analysis. We select three projects from each quadrant that have the longest history, and study each quadrant transition that they make in depth. Table 3.3 shows the quadrant transition history of the 12 selected projects. In this table, "-" means that the project was not yet under development, and "*" means that the project started, yet, lacked a large enough contributing team to satisfy our filtering criteria (i.e., more than ten active developers).

We first discuss the `Rails` project, which is one of the most popular web application frameworks, as it transitions through the quadrant states. The first version of `Rails` was released in 2004, version 1.0 was released in Dec. 2005, version 2.0 was released in Dec. 2007 and version 3.0 released in Aug. 2010. From the Table 3.3, `Rails` became a highly magnetic project in 2008, which coincides with the release of version 1.2 (1.2.6 is a stable version in ver. 1.x) and 2.0. We suspect that the appearance of the stable version increased the visibility of `Rails`, which in turn increased developer interest. Furthermore, the `Rails` 2.0 introduced `SQLite3` as the backend datastore and encouraged the use of REpresentational State Transfer (REST), which also likely intrigued technology-motivated developers to participate.

Next, we discuss the `jQuery` project, which began development in 2006 and is one of the most popular JavaScript libraries today. Interestingly, Table 3.3 shows that `jQuery` has transitioned between three quadrants and "*" state during its lifespan. The bursty nature agrees with the hypothesis of Bird *et al.* [3] that the rate of immigration in open source projects is non-monotonic. Early in development, `jQuery` transitioned from the terminal quadrant to "*" in 2008 and then to the attractive quadrant in 2009. Although the transition from the terminal quadrant to "*" state is not uncommon, the transition from "*" state directly to the attractive quadrant is interesting and worth exploring. In fact, on Sep. 2008, Microsoft and Nokia announce their support for jQuery.[7] This news generated much interest in `jQuery`.

On the other hand, blueprint-css – a CSS framework designed to ensure cross-browser compatibility when working with CSS – is classified as a terminal project by our quadrant plots. Blueprint-css transitions from "*" to the terminal quadrant in 2009 and 2011. From the transition, we can suspect that this project will be closed before too long. In fact, the project's last release (version 1.0.1) was May 14, 2011.

[7]`http://blog.jquery.com/2008/09/28/jquery-microsoft-nokia/`

23% of terminal quadrant projects eventually decay into a state where they have ten or fewer contributors, suggesting that our quadrant analysis can successfully identify projects at risk of becoming obsolete. Furthermore, we find that many quadrant transitions are accompanied by interesting events in a project's history.

3.4 Summary

Migratory trends of open source developers are of interest for software engineers. Project maintainers would like to retain the active developers that they have and would also like to attract new ones to grow the community.

This study applied the magnet and sticky population concepts to a set of open source projects.

We find that:

- Stickiness is a more common project attribute than magnetism, which can be motivated by more than a pro-community attitude [111], but also by professional activity (e.g., `django`).
- Quadrant plots can effectively identify at-risk projects. Projects decay into a state that have less than ten developers only from terminal state.
- Quadrant transitions often coincide with interesting events in a project's history.

From the findings, we successfully show the applicability of the magnet and sticky metrics to OSS projects since the metrics can capture the project characteristics. Furthermore, we can find at-risk projects from quadrant analysis and transition analysis. It suggests that the metrics are useful for understanding sustainability of OSS projects.

Chapter 4

Studying the Applicability of Magnet and Sticky with Large Set of OSS

4.1 Introduction

In Chapter 3, we introduce the magnet and sticky metrics and the way to measure the sustainability of open source software (OSS) projects, and demonstrate the applicability of the metrics with a small sample of GitHub projects [31]. Since the dataset used in the preliminary study includes 90 projects that were not randomly selected and not representative of GitHub,[1] the findings of the preliminary study are not sufficiently general.

In this chapter, we extend our preliminary study (Chapter 3). The largest extension is for the dataset. We now analyze 16,552 GitHub projects that have more than ten forks and developers (cf., Section 4.2). GitHub is one of the largest social coding platforms in the world, hosting many types of OSS projects. Furthermore, we modify the definition of the sticky value to improve its usability and investigate the typical duration between releases to avoid ad hoc decision about duration.

The summary of extensions is as below:

[1] `http://2014.msrconf.org/challenge.php` (Accessed 2015-06-15)

- Modifying the definitions of sticky values (Section 4.2).
- Adding a pilot study to investigate the typical duration between releases (Section 4.3).
- Extending the number of target projects from 90 to 16,552 (RQ1, 2).
- Adopting the typical release duration as time period in our experiments (RQ1, 2).

To generalize the results of our preliminary study, we address same research questions:

(RQ1) What are the typical distributions of projects from the magnet and sticky perspectives?

Motivation: Applying the concepts of magnet and sticky in an OSS context, we seek the project distributions of these concepts.

Result: 23% of contributors remain with a project. Lager projects attract larger number of new contributors than smaller projects.

(RQ2) How do the Magnet and sticky values change over time?

Motivation: By investigating the transitions of magnet and sticky values, we can capture the temporal evolution and decay of the projects.

Result: 53% of terminal projects eventually decay into a state of have fewer than ten contributors. On the other hand, 55% of attractive projects keep their popularity. Furthermore, stagnant projects are more likely to decay than fluctuating projects.

4.1.1 Organization of Chapter

The remainder of the chapter is organized as follows. Section 4.2 describes our study design such as definitions of magnet and sticky metrics, research questions that we address and dataset that we use in this study. Section 4.3 and 4.4 present the results of pilot studies investigating the release duration in OSS projects and studies that we conduct using the large-scale of OSS projects, respectively. Then, we discuss our results in Section 4.5. Section 4.7 concludes the chapter.

4.2 Study Design

This section provides an overview of our study. First, we show definition of magnet and sticky metrics, develop our research questions and motivations, then describe our dataset.

4.2.1 Definition of Magnet and Sticky

Measuring Contributor Retention and Attraction in OSS

This subsection describes our measurements of personnel retention and attraction in OSS. In this study, we use the magnet and sticky metrics defined by the Pew Research Center [70] for illustrating the migratory trends of citizens in the United States. The sticky metric revealed that just 28% of people are born in Alaska, but more than 75% of those born in Texas, remain in their birth states as adults. Furthermore, the magnet metric revealed that 86% of adult residents of Nevada had migrated from a different state.

Magnet and Sticky in State Populations

The Pew Research Center report [70] defines *Magnet states* as states that attract a large proportion of adults from other states. Thus, the magnet metric of a state is the proportion of adult residents who were not born in that state, relative to the total state population. The report also defines *Sticky states* as states that retain a large proportion of the people born in that state. Thus, the sticky metric of a state is the proportion of adult residents who were born in that state, relative to the residents born and living in the entire United States.

Magnet and Sticky in OSS Projects

The definitions of magnet and sticky are unambiguous in population studies, in which a single adult occupies only one state at a time, but are not directly applicable to open source projects, because contributors can contribute to several projects at the same time. Furthermore, the birth and current residence states of a single adult are identified from certificates of residence; however, no such document records the projects contributed by a developer. Therefore, if a developer commits to a project during a certain period, we identify that the developer has joined the project during that period. Therefore, the identification depends on the duration of the contribution period. In our preliminary study [107], we tentatively assigned the time window of the analysis as one year. In the present study, we more rigorously assess the time window as six months in a pilot study, see Section 4.3.

Using this duration, we divide time into periods. The period of interest is denoted the target period (p_i). The periods immediately preceding and succeeding the target period are called the previous period (p_{i-1}) and the following period (p_{i+1}), respectively.

In our preliminary study [107], the sticky metric was defined as the proportion of contributors in both p_i and p_{i+1}. In this study, we modify the definition to the proportion of contributors in p_{i-1} and p_i. In this manner, we can predict the status of the projects in p_{i+1} (i.e., the future status).

Therefore, we redefine the magnet and sticky metrics as follows:

Magnet projects are projects that attract a large proportion of new contributors. Thus, the magnetism of a project is the proportion of contributors who contributed during a particular period, but not during previous periods.

Sticky projects are projects in which many contributors continue making contributions. Thus, the stickiness of a project is the proportion of contributors who contributed during a particular period and also during previous periods.

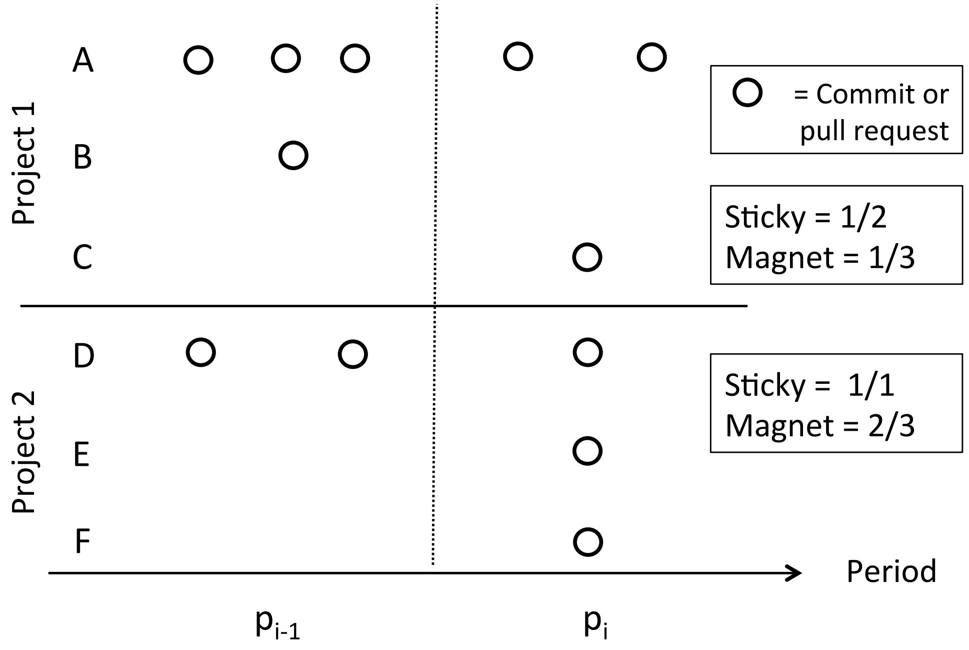


Figure 4.1: Calculation examples of our newly defined magnet and sticky values

Illustrative Example

The quantification of our definitions is demonstrated in Figure 4.1. In this example, we examine two projects during period p_i. There are six developers (A, B, C, D, E, and F) and two projects (1 and 2). Circles show the commits or pull requests contributed by the developers (listed down the left-hand side). For example, while developer A makes two contributions during period p_i, developer B makes no contributions during that period. To calculate the magnet metric, we observe that three new developers (C, E, and F) join the team at p_i, one of whom contributes to project 1 (C), while the others (E and F) contribute to project 2. In this case, the magnet values of projects 1 and 2 are $\frac{1}{3}$ and $\frac{2}{3}$, respectively.

To calculate the sticky metrics, we note that two developers (A and B) contributed to project 1 at p_{i-1}, one of whom (A) also contributes at p_i. Hence, the sticky value of project 1 is $\frac{1}{2}$. One developer (D) contributes to project 2 at p_{i-1}, and three developers (D, E, and F) contribute at p_i. However, the sticky metric only considers the number of contributors during p_{i-1} and p_i. Hence, the sticky value of project 2 is not $\frac{3}{1}$, but rather $\frac{1}{1}$.

4.2.2 Research Questions — Motivation and Approach

(RQ1) What are the typical distributions of projects from the magnet and sticky perspectives?

Motivation. First, we overview the trends of magnet and sticky values in the OSS context. This research question was addressed in our preliminary study [107], but here we expand the number of case study projects from 90 to 16,552. We also reconsider the time window. In this study, we established the time window as six months in a pilot study.

Approach. The magnet and sticky values of the studied OSS projects are calculated as described in Section 4.2.1. To visualize the data, we plot the magnet and sticky values of each project against each other project, and (similar to Khomh *et al.* [51]) divide the plot into four quadrants, as done in our preliminary study [107]:

Attractive projects (with high magnet and sticky values) successfully attract new developers while retaining their existing ones.

Fluctuating projects (with high magnet values, and low sticky values) successfully attract new developers but tend to lose existing ones.

Stagnant projects (with low magnet values, and high sticky values) retain their existing development team but struggle to attract new members.

Terminal projects (with low magnet and sticky values) struggle to retain existing developers while failing to attract new ones.

The quadrant thresholds can be dynamically configured. In this study, we use the median magnet and sticky values as the thresholds, as the median is a robust measure that is not heavily influenced by outliers.

As in our preliminary study [107], we focus on the latter six months of the most recently completed year of historical data (i.e., from July to December of 2013). The most recent dataset includes the largest number of projects.

Note that the sticky value depends on the number of contributors in both the target and the previous time periods (Figure 4.1). If few developers have contributed in the previous time period is small, the sticky value tends to be high. Therefore, to reduce the noise in our results, we filter out projects with less than ten developers in the previous time period. We also consider the time period in which the project started. The sticky value of a start-up project is 0, because all of the developers are new and no developer has contributed during the previous time period. Therefore, we filter out new projects in the target time period.

Besides an overview of the distribution, we also show typical values of differently sized projects. As mentioned above, the magnet and sticky metrics are influenced by the number of total developers in the target and previous time periods. Therefore, we divide projects according to their number of developers, and display the median magnet and sticky values of projects in each size category.

(RQ2) How do the Magnet and sticky values change over time?

Motivation. By investigating the changes in magnet and sticky values, we can capture the temporal evolution and decay of the projects.

Approach. We analyze how the aging projects transit among the quadrants of Figure 4.3. As the quadrant boundaries will likely change, the boundaries are recalculated in each time period. In this study, we track the magnet and sticky values from 2000 to 2013 (i.e., thorough 28 time periods).

4.2.3 Dataset

Our dataset is the GitHub dataset "GHTorrent" provided by Gousios [31].[2] Part of this dataset is provided in the MySQL database and includes diverse software evolution data from a large collection of OSS projects, such as issue reports, pull requests, organizations, followers, stars and labels. We focus on the code authorship data in the commits and pull

[2]We use MySQL databases dump at 2014/04/02

Table 4.1: Overview of the GHTorrent dataset used in this study

Dataset	#Users	#Repos	#Commits	#PullReqs
This Study	3,426,046	8,510,504	96,999,485	3,200,428
Preliminary Study	499,485	108,718	555,325	78,955

requests tables.

GitHub has unique features such as *fork* and *pull request* for collaborative development. GitHub describes Fork as a copy of a repository. Forking a repository allows you to freely experiment with changes without affecting the original project.[3] Pull request allows users to "tell others about changes you've pushed to a repository on GitHub. Once a pull request is sent, interested parties can review the set of changes, discuss potential modifications, and even push follow-up commits if necessary."[4] A typical development process on GitHub, driven by fork and pull requests, proceeds as follows:

1. A developer forks a repository to which he or she hopes to contribute.
2. The developer makes changes to the fork repository.
3. The developer sends a pull request to the original repository to reflect his or her changes to the original repository.
4. If the owner of the original repository allows the pull request, the changes are included in the original repository.

Table 4.1 overviews the dataset used in our study. This dataset is 800 times larger than that accessed in our preliminary study [107]. Besides the higher number of repositories, the current dataset includes more users, commits and pull requests than the dataset of the preliminary study.

As described above, the entire dataset is divided into sub-datasets covering different time periods. Therefore, we examine the column `created_at`. However, this column contains uninterpretable or nonsensical dates such as '0000-00-00', '0000-00-00 00:00:00', and 2025 (as

[3]https://help.github.com/articles/fork-a-repo/ (Accessed 2015-06-15)
[4]https://help.github.com/articles/using-pull-requests/ (Accessed 2015-06-15)

the commit year). As the dates of commit and pull requests are critical to our analysis, we filtered out such cases.

4.2.4 Developers

In this study, a developer is a person who alters software code. In the GitHub dataset, developers can either perform the commit themselves or send a pull request to an upstream repository maintainer. Both actions are viewed as developmental activity in our magnet and sticky analyses. According to Kalliamvakou *et al.* [48], most of the accepted pull requests sent from fork repositories are absent in the histories of original repositories. Therefore, we mine the developer information from both the original and fork repositories. In particular, we obtain the author information from the retrieved commits. From the pull requests, we obtain the information of actors who send (i.e., open) the pull requests.

The GitHub system identifies authors as registered or non-registered from the email addresses of the commits.[5] If the author of a commit is not registered, GitHub records the author information that can be obtained from Git, such as name and email address, along with a unique id. In this system, some developers are assigned multiple user ids. Therefore, we clean the data using the tool[6] that matches users with their information recorded in GitHub (e.g., login name, actual name, email address and location).

4.2.5 Projects

Not all of the repositories included in our dataset are software projects [48]. Other repository categories include, but are not limited to, *Experimental* (e.g., examples, demonstrations and samples,) and *Storage* (e.g., configuration files and personal use). We assume that the number of fork repositories and developers is negligible in these categories, since these repositories do not require collaboration with others. To identify software projects, we note the number

[5]https://help.github.com/articles/why-are-my-commits-linked-to-the-wrong-user/ (Accessed 2015-06-15)
[6]https://github.com/bvasiles/ght_unmasking_aliases (Accessed 2015-06-15)

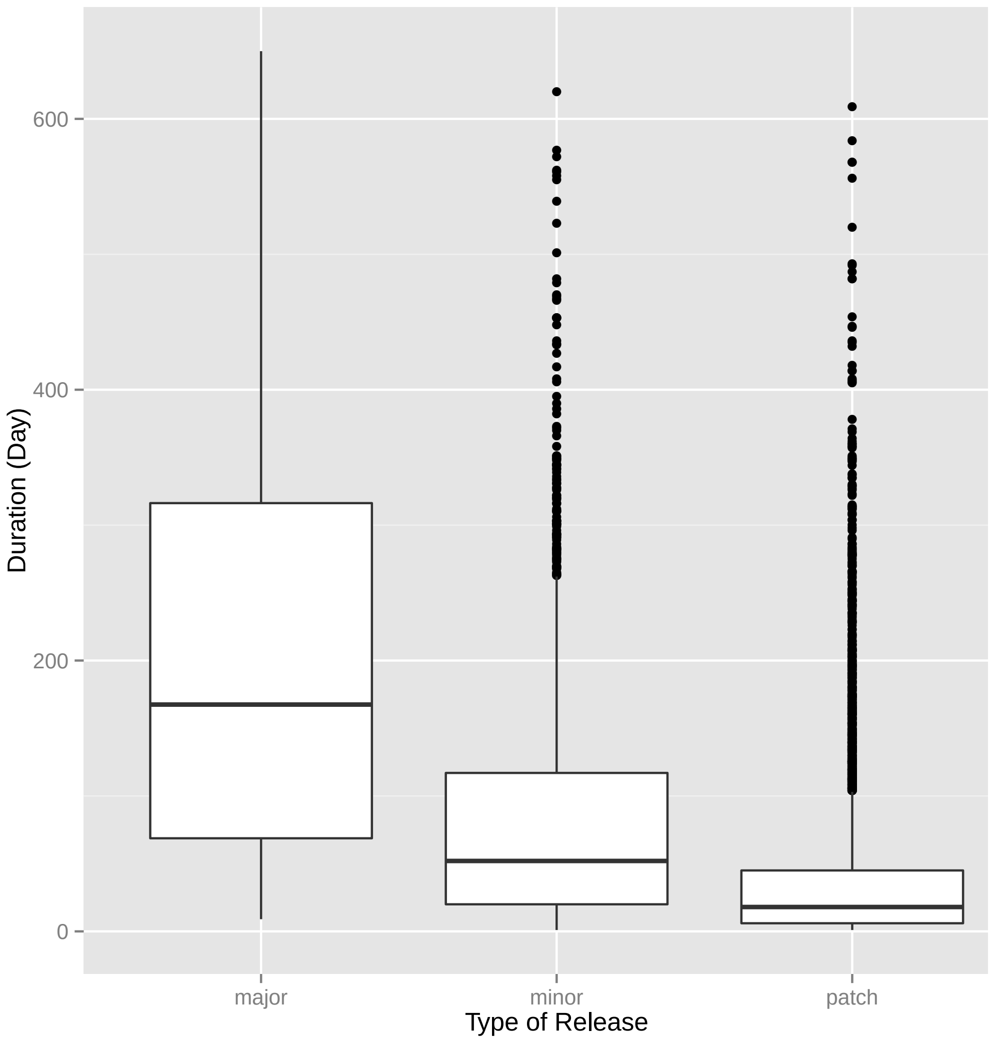


Figure 4.2: Release duration (days)

of fork repositories and number of developers, both of which indicate collaborative activity. Projects with less than 10 fork repositories and 10 developers are filtered out. The post-filtered dataset includes 16,552 original repositories.

Our study focuses on projects adopting the pull-based model, which excludes the 55% of the GitHub projects using shared repository models [32]. Moreover, we filtered projects with fewer than 10 forks. Therefore, our findings are not generalizable to shared repository models.

4.3 Pilot Study

In our preliminary study [107], we tentatively assigned the target period of the magnetism and retention calculations as one year. However, the validity of this assignment was not discussed. In the present study, the appropriate period is identified in a pilot study.

In defect prediction, code review and other studies relying on Mining Software Repositories (MSR), experiments are conducted at the release-level [49, 61]. However, when conducting experiments across multiple projects, the release-level is inappropriate for two reasons. First, we desire to compare metrics at the same time; second, multiple projects are not released simultaneously.

Instead of the release-level, we therefore adopt the representative release duration. Some of the large projects regularly update their products [50]; Google Chrome and Mozilla Firefox update their products every six weeks (i.e., adopt a rapid release model). If all the projects in our dataset are periodically updated at the same rate, that period becomes a useful parameter in the magnetism and retention calculations. Therefore, we manually inspect some projects to determine the constancy of their update periods. Unfortunately, unlike Google Chrome and Mozilla Firefox, most projects are not regularly upgraded. Hence, to identify the typical release period of the GitHub projects, we calculate the duration between the releases of each project.

Approach. GitHub releases the products[7] and provides the API to access the released information. We extract the release information (version number and published date) of all target projects from the GitHub API. The published and git tag dates are independent, although both dates have the same version name and release date of the updated version onto GitHub. Although GitHub recommends the semantic labeling of new versions (in MAJOR.MINOR.PATCH number format) [71], some projects do not follow this recommendation. Projects not adopting the semantic versioning system are removed from our analysis. We also remove alpha versions and release candidates (e.g., 1.0.0-alpha, 1.0.0-pre), because such

[7]https://help.github.com/articles/creating-releases/ (Accessed 2015-06-15)

Table 4.2: Release duration of major, minor, patch upgrades of GitHub projects (days)

	Major Release	Minor Release	Patch Release
Min	9	1	1
1st Qu.	68.8	20	6
Median	167.5	52	18
Mean	202.8	84.2	38.5
3rd Qu.	316.3	117	45
Max	650	620	609
NumberOfUpdate	98	2,021	6,092

versions are candidates rather than official releases. After filtering, we extract the release information of 16,682 versions of 1,778 projects. From this information, we calculate the duration of *Major*, *Minor* and *Patch* releases.

In the semantic versioning system [71], a major release denotes an update of incompatible API changes, and the version number changes from (`x.0.0`) to (`x+1.0.0`). Minor releases add functionality to a project in a backwards-compatible manner, and alter the version number from (`x.y.0`) to (`x.y+1.0`). Patch releases correct backwards-compatible bugs, and are marked by version number changes from (`x.y.z`) to (`x.y.z+1`).

Multiple versions (even major upgrades) were occasionally released on the same day, and in different order from their version numbers. We presumed that such projects had been moved to GitHub from another hosting service (e.g., SourceForge), and had been previously released. A developer could then release all versions onto GitHub on the same day. Therefore, we filter out updates with duration below one day and released in different order from their version numbers.

Results. The duration distributions of the major, minor, and patch updates are presented in Figure 4.2 and Table 4.2. To improve the accuracy of the pilot study, we focus on duration between the 1st and 3rd quantiles.

Figure 4.2 reveals clear duration differences between the major, minor, and patch releases. At the patch level, the duration at the 1st and 3rd quantiles are 6 days and 45 days, respectively, with a median of 18 days (approximately half a month). For minor upgrades, the

Table 4.3: Median values of magnet and sticky OSS projects released on GitHub

Metrics	# of Total Developers in Project				
	10-50	51-100	101-500	501-	Total
Median Magnet	0	2.9e-04	7.5e-04	9.1e-03	4.9e-05
Median Sticky	0.23	0.23	0.24	0.51	0.23
# of Projects	4,275	217	112	8	4,612

duration at the 1st and 3rd quantiles are 20 days and 117 days, respectively, with a median of 52 days (approximately two months). At the major level, the duration of the 1st and 3rd quantiles are 69 days and 316 days, respectively, and the median is 168 days (approximately half a year).

New versions of GitHub projects are released in 18 days at the patch level, 52 days at the minor level and 168 days at the major level.

The pilot study revealed the typical duration of each level of releases. In the following study, we adopt the median duration of the major release as the time window, because the major release is the most important update of a project.

4.4 Study Results

(RQ1) What are the typical distributions of projects from the magnet and sticky perspectives?

Figure 4.3 presents a magnet vs. sticky quadrant plot of the OSS projects released on GitHub during the latest time period (July to December of 2013). Attractive, fluctuating, stagnant, and terminal projects land in the red (upper-right), green (upper-left), blue (lower-right), and purple (lower-left) quadrants, respectively. The names of the extremely attractive projects are annotated in the figure.

The median magnet value is quite small, and the median sticky value is only 0.23 (Figure 4.3). Although the magnet value is typically below 0.005 (marked by the horizontal

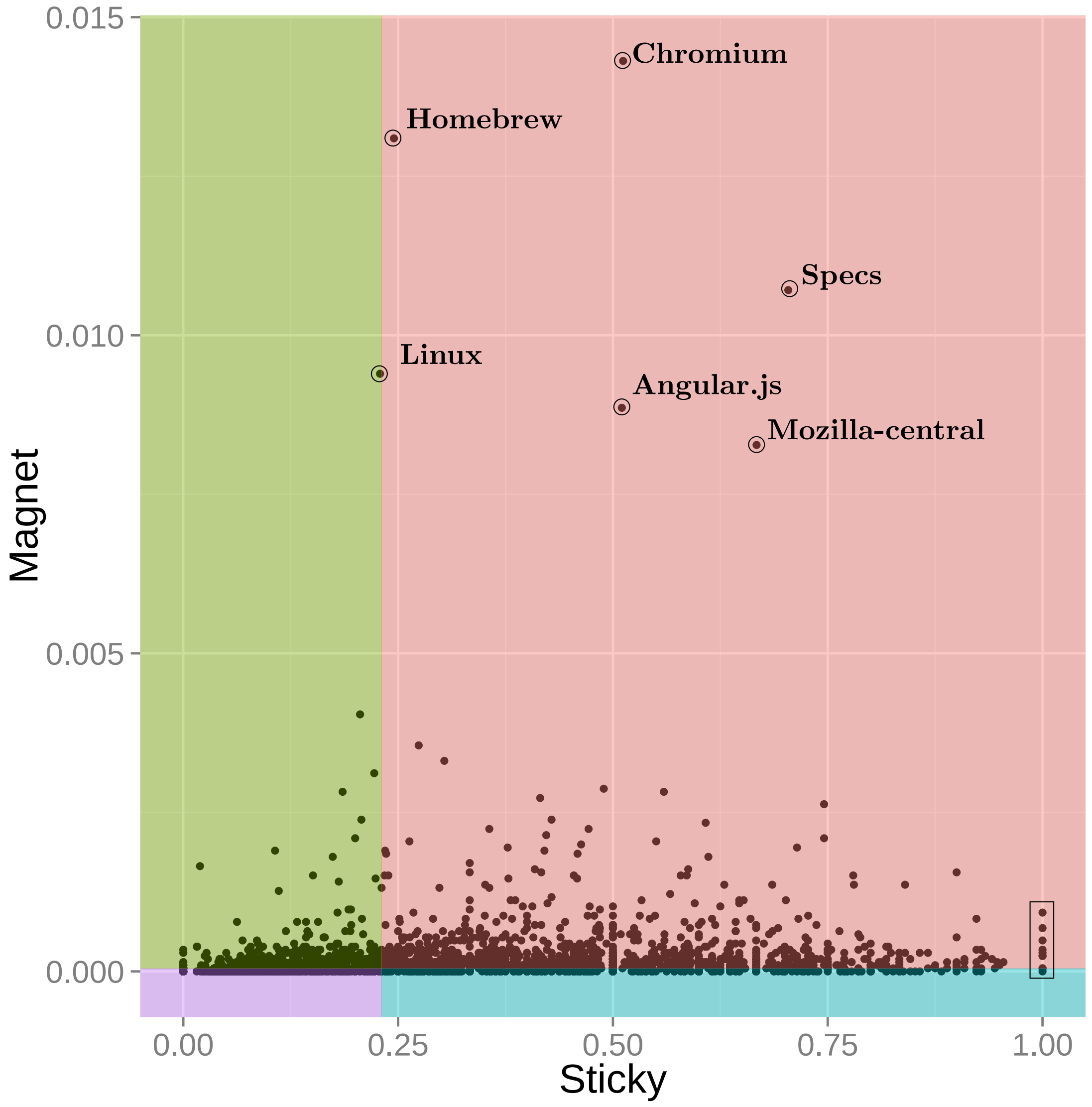


Figure 4.3: Distribution of magnet and sticky values for the studied projects

division on the plot), some projects have large magnet values. These findings suggest that the distribution of the number of new developers in each project is highly skewed, and that approximately 23% of developers remain in the same projects.

The results are similar to our preliminary study [107]. In preliminary study, magnet values are much smaller than sticky values and only a few projects have large magnet value. Furthermore, the median sticky value is approximately 20%.

Six of the projects have exceptionally high magnet, namely, `Linux`, `Homebrew`, `Chromium`, `Angular.js`, `Specs`, and `Mozilla-central`. `Linux` is among the most famous projects, and

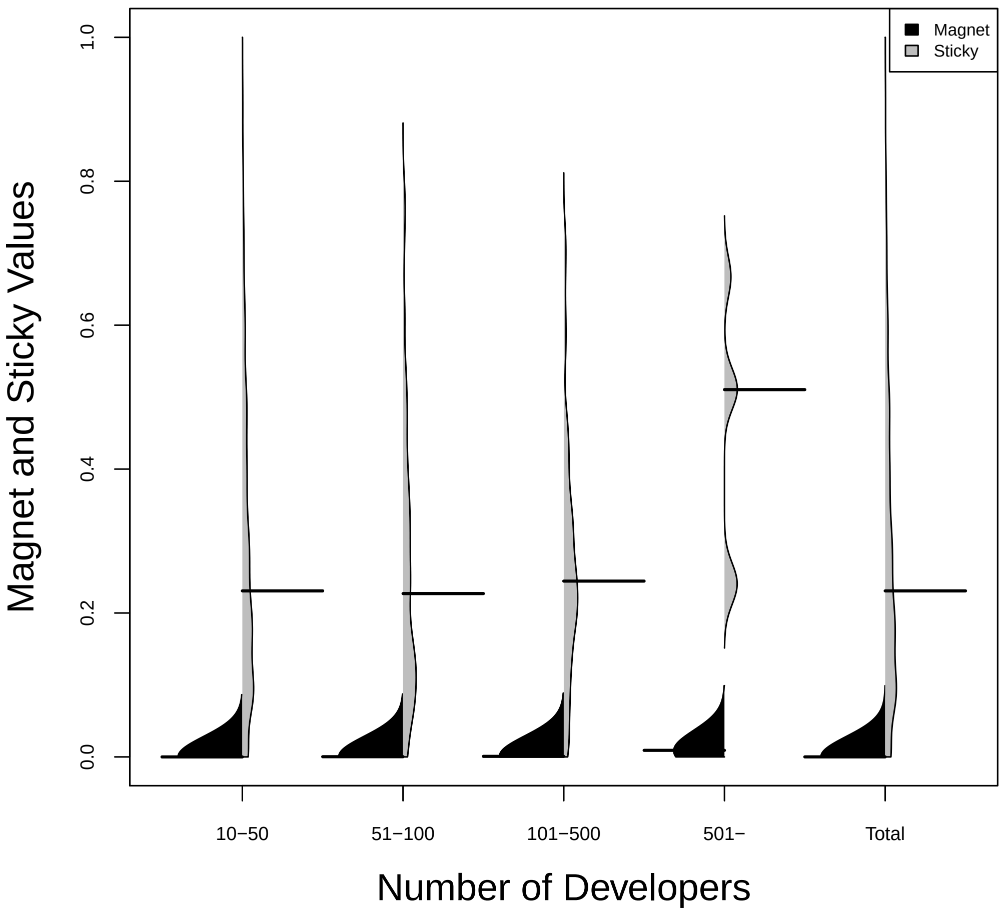


Figure 4.4: Beanplots of magnet and sticky values, grouped by developer size

`Homebrew` is a popular package management tool for Mac OS X. The web browser project `Chromium` is basis of Google Chrome. `Angular.js` is the web framework for JavaScript, `Specs` is a repository for the public CocoaPods[8] specification, and `Mozilla-central` is a repository for source codes implemented by the Mozilla foundation such as Firefox web browser.

The `Linux`, `Chromium`, `Homebrew` and `Mozilla-central` projects are well-known and universally popular. Therefore, many developers are expected to join these projects.

The popularity of `Angular.js` and `CocoaPods` during the analysis period was checked by Google Trends[9], which records the number of query searches on Google in chronological order. In papers [13, 100], the popularity of a project is assessed from the numbers of web

[8]CocoaPods is the dependency manager for Swift and Objective-C.
[9]`https://www.google.co.jp/trends/` (Accessed 2015-10-15)

Table 4.4: Projects with sticky values of 1.0

Name
DIRACGRID/DIRAC
JetBrains/MPS
georchestra/georgestra
dxw/wordpress
virtual-world-framework/vwf
open-mpi/ompi-svn-mirror
rose-compiler/edg4x-rose
stackforge/savanna
PCGen/pcgen
crosswalk-project/crosswalk

pages indexed by Google and views of the project page. However, the popularity trends of the projects are difficult to identify by these indicators. Therefore, we identify the popularities of the projects thorough Google Trends. The search numbers of both `Angular.js`[10] and `CocoaPods`[11] were increasing from 2013. Therefore, we assume that as the projects gained popularity from 2013, they increasingly attracted new-comers to their development. This finding suggests that the magnet and sticky values well-indicate the fame and popularity of a project.

Ten projects in Figure 4.3 have a sticky value of 1.0 (we put a framed box around the projects). The names of these ten projects are listed in Table 4.4. To identify the reason for such high sticky values, we check their web pages and the developers' affiliations to find out the primary developers and maintainers of the projects. If more than half of developers belong to companies, we consider that the projects are supported by those companies. All the projects in Table 4.4 are found to be developed or supported by companies or laboratories. In general, non-company developers are likely to join OSS projects as hobbyists [55], but company and laboratory developers probably join OSS projects as part of their work [55, 79]. Therefore, projects supported by company or laboratory developers are more likely to be constantly contributed by the same developers than projects supported by non-company

[10]`https://www.google.co.jp/trends/explore#q=angularjs` (Accessed 2015-10-15)
[11]`https://www.google.co.jp/trends/explore#q=CocoaPods` (Accessed 2015-10-15)

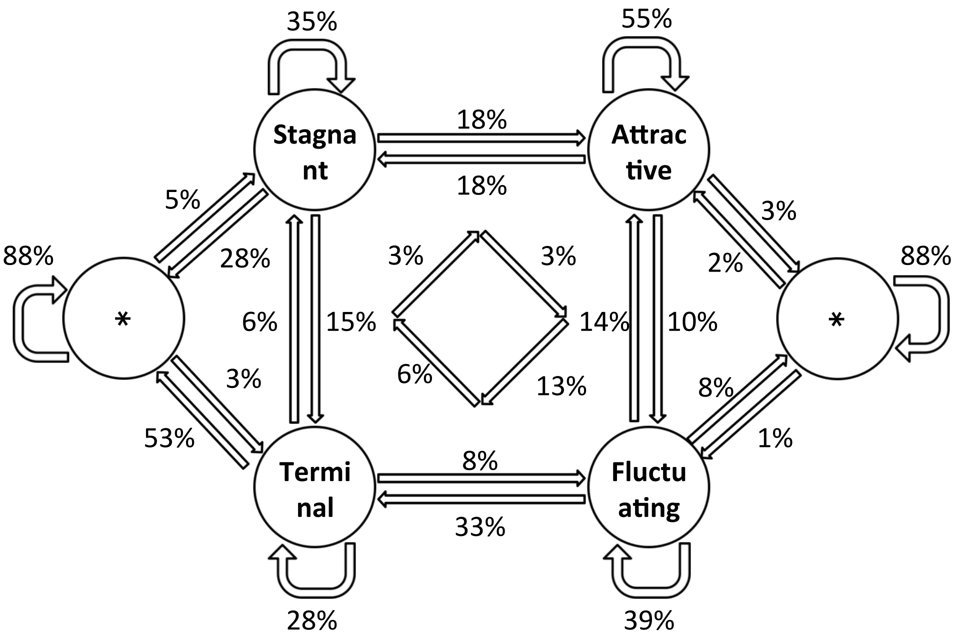


Figure 4.5: The likelihood of quadrant transitions

developers.

We then study the impact of the number of project developers on the magnet and sticky values. Figure 4.4 shows beanplots of the magnet and sticky values of differently sized projects (the medians are listed in Table 4.3). In these plots, the left (black) regions and right (gray) regions indicate the magnet and sticky values, respectively. From left to right, the number of developers is binned into 10–50, 51–100, 101–500, 501– plus, and all sizes.

From Figure 4.4 and Table 4.3, we find that the magnet and sticky values are generally higher for larger projects than for smaller projects. As the denominator of the sticky value is the total number of developers in the previous time period, the sticky value is inversely proportional to the number of developers. However, large projects tend to have large sticky values, consistent with our intuition that developers prefer to join and contribute long-term to such projects.

Larger projects attract and retain more developers than smaller projects. 23% of developers remain with the same project irrespective of size (total number of developers), and new developers tend to join popular and famous projects.

(RQ2) How do the Magnet and sticky values change over time?

Figure 4.5 illustrates the quadrant transition likelihood on a state transition diagram. Percentages describe the likelihood of a transition from one quadrant to another (or the same) quadrant. The direction of the arrow indicates the direction of the quadrant change. For example, the likelihood of moving from the attractive to the terminal quadrant is 13%. States marked with "*" indicate projects that failed our filtering criteria (ten or more developers) during some time periods. To improve the readability of the figure, we plot two "*" states, although these states are semantically identical.

According to this figure, 3%, 8%, 28%, and 53% of the attractive, fluctuating, stagnant, and terminal projects entered the filtered out state ("*"). Although any project can drop into the "*" state, the probability is much higher for terminal projects than for projects in other quadrants. Therefore, terminal projects are very likely to decay into the "*" state. Intuitively, we expect that as terminal quadrant projects are losing team members and struggling to attract new ones, they will eventually die.

This result is different from our preliminary study [107]. In our preliminary study, projects decay into "*" state only from terminal quadrant, however, in this study, we found that projects that are in other three quadrants decay into "*" state.

Interestingly, 28% of the stagnant projects, but only 8% of the fluctuating ones, decay into the "*" state. In both quadrants, one of the two metrics (magnet or sticky) is high; therefore, we expected that both quadrants would enter the "*" state with similar likelihood. The observed asymmetry might reflect the impact of number of developers. As fluctuating (stagnant) projects are characterized by high (low) magnet and low (high) sticky values, it appears that magnet measure is more affected by number of developers than sticky.

Moreover, projects in the fluctuating, stagnant, and terminal quadrants do not easily transit to the attractive quadrant. Only 18% of the projects entered the attractive quadrant from other quadrants, but 55% of the attractive projects maintained their high magnetism and stickiness. This phenomenon indicates that attractive projects are more stable than

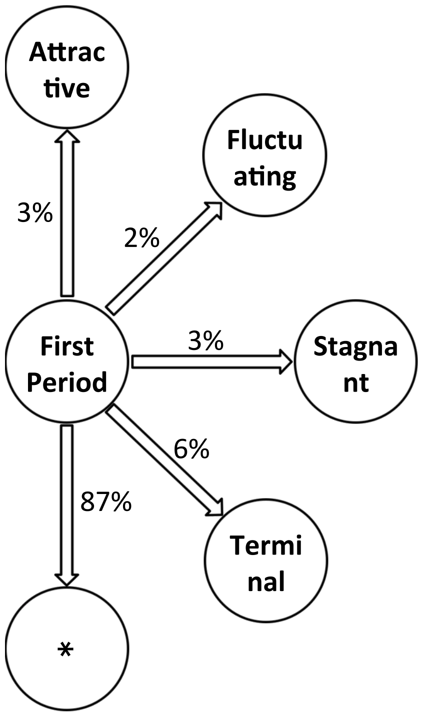


Figure 4.6: Likelihood of quadrant transitions from the first period

projects in other quadrants.

In Figure 4.5, we filtered start-up projects during the time period because the sticky value of such projects is 0, as earlier described in RQ1. However, the status transitions from the first time period to the next warrant investigation. Figure 4.6 shows the likelihood of quadrant transitions from the first to the second time period. Only 13% of the projects maintained ten or more developers in next one, indicating the difficulty of retaining and acquiring developers after initiating a project.

53% of the terminal projects eventually decayed into a state of ten or fewer contributors, while 55% of the attractive projects maintained their popularity. Only 13% of the projects identified in the first time period had maintained ten or more developers in the second period.

4.5 Discussion

This section discusses our analysis and results.

4.5.1 Discussion of RQ1

From the result of calculating magnet and sticky values at latest time period, we obtained the distributions of the values of projects such as the median magnet value is 0.05 and sticky value is 0.23. The results are similar to our preliminary study [107]. Furthermore, we found that larger projects attract and retain larger number of developers. These findings fit our intuition. The large projects are already known by many people and there are more information of the projects compared to small projects. Hence, we assume that new developers can find the projects and the information easily. For existing developers, contributing at fame and popular project is proud thing and motives them. From these expectations, we assume that both types of developers (new developers and existing developers) have good motivation to contribute to the projects in larger projects.

Also, we showed the median of magnet and sticky values at the latest time period. We assume that the values act as a gauge of project health. If magnet and sticky values of a project are lower than the median values, the project faces a risk of decaying. In particular, sticky value is stable across total number of developers. Hence, projects that have lower sticky values are especially risky.

4.5.2 Discussion of RQ2

From the result of calculating likelihood of quadrant transitions, we found that 53% of terminal projects eventually decay into a state where they have fewer than ten contributors and 55% of attractive projects keep the popularity. We also revealed some different trends from our preliminary study [107]. In preliminary study, only terminal projects decay into the "*" state, but in this study, attractive, fluctuating, and stagnant projects also decayed into the

"*" state. We attribute these differences to the much larger dataset in this study.

We plan to study project survivability (i.e., project keep maintaining) using the transition in our future work. Chengalur-Smith *et al.* [9] showed that the number of developers positively correlates with project survivability. Therefore, we expect that project survivability can be predicted from the analyzed trends and the definition of project failure, as proposed by English and Schweik [23]. This estimation is planned for future work, but we must consider the definition of project death. In this study, the "*" state represents projects with fewer than ten developers and we consider projects moved to the "*" state as a type of obsolete projects. However, some projects with few developers are robustly sustained. Therefore, a small number of developers does not signify that the project will die (i.e., project stops its development). We must consider the definition of obsolete project in future work.

4.6 Threats to Validity

We now discuss the threats to the validity of our work.

4.6.1 Construct Validity

In this study, we used the number of developers (i.e., related to sustainability of organization) as a proxy of sustainability and did not evaluate factors showing a lack of development and maintenance activities, such as the number of commits. Furthermore, we did not evaluate the availability of magnet and sticky metrics by using statistical test or models, even though we showed state transition diagrams. However, both the activity and the number of developers have been used as a proxy of sustainability in several studies [7, 9, 27, 33, 35, 58, 91]. Hence, we believe that our study demonstrated sufficient applicability of our approach. But, to show the effectiveness of our approach more clearly, these two points are also important.

4.6.2 Internal Validity

In this study, we analyzed the GitHub dataset published as GHTorrent [31]. In the dataset, we analyzed MySQL dump. As we explained at Section 4.2, we divided the dataset into sub datasets of each time period. We used a column `created_at` to divide the dataset. However, there are some strange data in the column. For example, the date is '0000-00-00' or '0000-00-00 00:00:00' or the date does not make sense, e.g., the year of the commit is 2025. In this study, since the date of commit and pull request are an important factors, we removed such cases.

4.6.3 External Validity

In GitHub, 55% of the projects do not use pull-based models but use shared repository models [32]. Our study focuses on projects using pull-based models, and we filtered projects with less than 10 forks. Therefore, our findings are not generalized to projects of shared repository models.

4.7 Summary

Building on our preliminary study (Chapter 3), we aimed to better understand the sustainability of OSS projects. First, we extended the dataset from 90 to 16,552 projects to generalize our preliminary results. Second, we redefined the sticky metric to better suit our purpose. Third, we experimentally identified the typical duration between product releases.

In this study, we obtained results similar to and different (RQ2) from our preliminary study [107]. Roughly, the results of RQ1 are similar to and those of RQ2 are different from our preliminary study.

The main results of the experiments are summarized below as follows:

- 23% of developers remain in the same projects.
- Larger projects attract and retain more developers.

- 53% of terminal projects eventually decay into a state of fewer than ten contributors.
- 55% of attractive projects remain in the attractive quadrant.

These findings suggest that it is possible to capture features of sustainability of OSS projects using both evolvability (magnet) and stability (sticky) measures. Moreover, it is possible to quantitatively measure the evolvability and stability of OSS projects by using these metrics.

As mentioned in Section 4.5, our future work will investigate the relationship between the magnet and sticky metrics and analyze sustainability based on activities (such as the number of commits). We found that terminal projects are abandoned and attractive projects are frequently sustained. These findings indicate a relationship between sustainability and the proposed metrics, but cannot quantify this relationship. In particular, we did not define the failure of projects. Therefore, when investigating this relationship in future work, we should adopt a definition of project failure, as proposed by English and Schweik [23].

Chapter 5

Revisiting the Proportion of Core Developers for OSS Sustainability

5.1 Introduction

Understanding open source software (OSS) communities, i.e., the groups that are responsible for developing and maintaining an OSS system, is as important as understanding OSS systems themselves for the sustainability of OSS projects. By studying OSS communities, we accumulate knowledge about how these communities manage highly distributed development teams [64, 76]. Such knowledge enables the OSS development model to augment or replace development models in proprietary settings.

At the heart of OSS communities are *core developers*, i.e., the developers who take a leading role in the development and maintenance of a software project. For instance, Nakakoji *et al.* [65] state that core developers are *responsible for guiding and coordinating the development of an OSS project. Core Members are those people who have been involved with the project for a relative long time and have made significant contributions to the development and evolution of the system.* Mockus *et al.* [64] define core developers as the most productive developers who have made roughly 80% of the total contributions. Although these heuristics slightly differ, researchers agree that the impact that core developers have on a project is large.

Recent studies have shown that a small number of developers make a large proportion of the code contributions [19, 28, 64]. Moreover, it has been shown that the number of core developers follows the *Pareto principle* (a.k.a., the 80-20 rule), i.e., 80% of the contributions are produced by roughly 20% of the contributors [30, 52, 83].

Although the prior work makes important strides towards understanding core teams in OSS, the conclusions are drawn based on a small sample size (i.e., 1-9 studied systems). Therefore, in this chapter, we set out to revisit how the Pareto principle applies to core teams in a large sample of 2,496 GitHub projects. We study GitHub projects because GitHub is one of the most popular social coding platforms, and many successful OSS systems are developed on GitHub (e.g., `Rails`[1]). Through analysis of the 2,496 GitHub projects, we address the two research questions.

The main contributions of this chapter are:

- A large-scale analysis of the core teams of 2,496 GitHub projects.

- A comparative analysis of three heuristics for identifying core developers.

5.1.1 Organization of Chapter

The remainder of the chapter is organized as follows. Section 5.2 provides an overview of the study design including our heuristics for identifying core developers, research questions and dataset. Section 5.3 shows results of our case study. Section 5.4 discusses findings from our study. Section 5.5 discloses the threats to the validity of our study. Finally, Section 5.6 draws conclusions.

[1]`https://github.com/rails/rails`

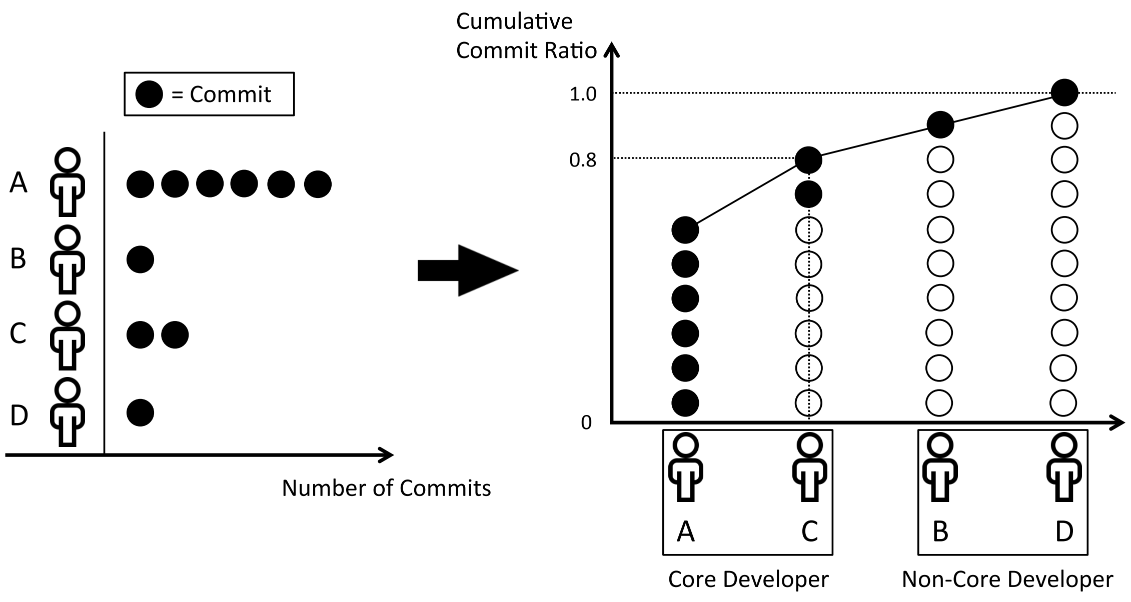


Figure 5.1: Identifying core developers using an example project

5.2 Study Design

5.2.1 Definition of Core Developers

In order to perform our study, we need to define heuristics to identify core developers. Inspired by previous studies, this study explores three heuristics to identify core developer from the perspective of contributions as described below.

Commit-Based Heuristic

Several papers [30, 64, 83] use the heuristic to defines core developers as those who produce roughly 80% of the total contributions. In this study, we adopt this heuristic — after sorting the developers by their number of contributions in descending order, the core developers are those who have produced 80% of the project contributions, cumulatively.

For instance, Figure 5.1 shows an example project with four developers: A, B, C, and D. In order to determine core developers, we first sort the developers by the number of commits in descending order (A: 6, C: 2, B: 1 and D: 1). Next, we calculate the percentage of total commits that each developer has produced (A: 60%, C: 20%, B: 10%, and D: 10%). Then, we calculate the cumulative percentage (A: 60%, C: 80%, B: 90%, and D: 100%). Finally, we select core developers, one at a time, moving left to right, until we reach a cumulative percentage of 80%. In this example, A and C are identified as core developers.

In our algorithm, we do not handle the special case where there are some developers who have same number of commits on the border of core and non-core developers. We do not suspect that who we select to be a member of the core team should have a significant impact on the results, since: (a) these developers have produced the same number of contributions and (b) they are at the tail end of the core team contributions.

LOC-Based Heuristic

Similar to the commit-based heuristic, the LOC-based heuristic defines core developers according to the size of the contributions that they make [52, 64]. While we conduct our experiments using three size metrics, i.e., *the number of added lines*, *the number of deleted lines* and *the churn* (the sum of the number of added and deleted lines), we find that the results are similar across the three metrics. Therefore, to conserve space, we show only the results for churn in the remainder of the study. Similar to commit-based heuristic, we identify core developers as those who cumulatively contribute 80% of the churn.

Access-Based Heuristic

Core developers can also be defined as those who have been given direct write access to the main VCS repository. For example, in projects like PostgreSQL, only core members can record changes directly in the main VCS repository — other contributors must convince core developers to record their changes on their behalf [29]. Hence, we can also identify core developers from a VCS access perspective.

In GitHub, project owners can grant write access to the project's main repository to other contributors. GHTorrent collects this information using the *collaborators* API and stores it in the *project_members* column [31]. According to the description of the collaborators API, the list includes all organization owners and users with access rights.[2] Since this list may include users who are members of an organization, but who did not contribute to a project, we define the access-based core developers as those who appear in the access list and have also made at least one commit.

Unfortunately, we find that roughly half of the studied projects do not use the access-based feature of GitHub. These projects are filtered out of our analysis when we use the access-based heuristic.

5.2.2 Research Question

From the survey that we described in Chapter 2, we build the following research questions.

(RQ1) Does the proportion of core developers of GitHub projects follow the Pareto principle?

While the actual proportion of core developers varies depending on the heuristic of core developers that we use, 26%-58% of projects have core teams that are too small ($\leq$ 10% of active contributors) or 5%-28% have core teams that are too large ($\geq$ 30% of active contributors) to be considered compliant with the Pareto principle.

(RQ1) Is there any difference between the contributions of core and non-core developers?

Surprisingly, we find that the proportions of core and non-core developer activity are very similar when we normalize them by their contribution rates. For example, bug fixing activity accounts for 18%-20% of core developer contributions and 21%-22% of non-core developer contributions.

[2]`https://developer.github.com/v3/repos/collaborators/`

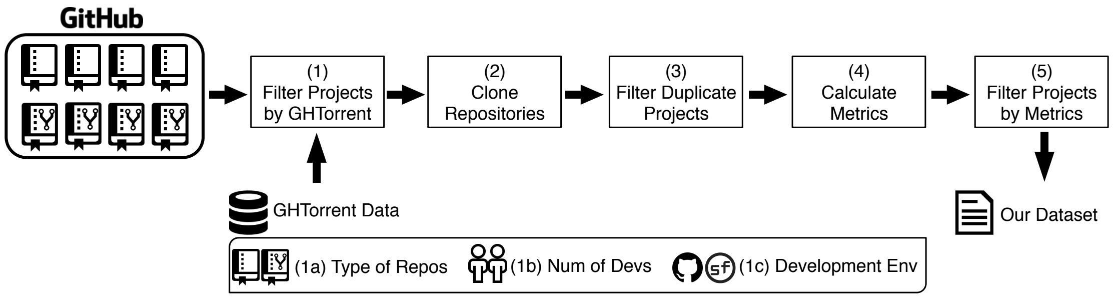


Figure 5.2: An overview of our data extraction approach

Table 5.1: Finding self-identified mirror projects.

Category	Used regular expression [48]	#Projects
Mirror Of	$mirror\ of\ .*repo\|git\ repo\ of$	10
Sourceforge	$sourceforge\|sf\backslash.net$	6
Bitbucket	$bitbucket$	2
Subversion	$\backslash W(svn\|subversion)\backslash W$	4
Mercurial	$\backslash W(mercurial\|hg)\backslash W$	0
CVS	$\backslash Wcvs\backslash W$	0
Total	-	23

5.2.3 Dataset

In this subsection, we describe how we prepare the dataset of GitHub projects for our study. Figure 5.2 provides an overview of our dataset preparation steps.

We begin our study with the collection of GitHub project data that is available via GHTorrent [31]. However, GitHub hosts a large number of repositories, many of which are not software projects. Hence, we filter the GHTorrent data according to the suggestions of Kalliamvakou *et al.* [48]. We take three steps to create our dataset from the available GitHub projects. Initially, GHTorrent includes 8,510,504 repositories.

(1) Filter Projects by GHTorrent Data

(1a) Type of Repository. In GitHub, there are two types of repositories: *main repositories* and *fork repositories*. A fork is a working copy of a main repository. Forking a repository allows developers to freely experiment with changes without interfering with the ongoing development of the original project.[3] In GitHub, fork repositories can contribute changes back upstream to the main repositories that they are forked from by issuing pull requests. If the maintainers of the upstream repository agree with the changes that are proposed by a pull request, the request is accepted, and the changes are integrated into the main repository. As all accepted pull requests are stored in main repository, we only extract commits from the main repository, ignoring commits that only appear in forks.

(1b) Number of Developers. Two types of authorship data are recorded in Git repositories. The committer is the team member who recorded the changes in the repository using the `git commit` command. The author is the team member who produced the code change itself. In this study, we focus on the authors of the changes, ignoring the committer data, since the author is the team member who actually produced the changes, while the committer is the team member responsible for the integration work.

Furthermore, since projects with a small number of developers can easily achieve extreme core team proportions, we filter away projects that have too few developers (number of developers < 10).

(1c) Development Environments. In this study, we would like to investigate core developers especially in projects that are developed on GitHub. Kalliamvakou *et al.* [48] find that GitHub is not only a popular social coding platform, it also serves as a popular host for mirrored repositories.[4] Since such mirrored projects may not be developed in the same manner as projects on GitHub, we need to filter them out of our dataset. To do so, we heed the advice of Kalliamvakou *et al.* [48]:

1. Avoid projects that have a large number of committers who are not registered GitHub users.
2. Avoid projects that explicitly state that they are mirrors in their description.

To address item 1), we filter away projects where less than 90% of the committers are

[3]`https://help.github.com/articles/fork-a-repo/`
[4]e.g., `https://github.com/apache`

registered GitHub users. To address item 2), we filter away projects with descriptions that match the regular expressions listed in Table 5.1, as proposed by the prior work [48].

After applying the filters of steps (1a)-(1c), 4,618 projects remain in our dataset.

(2) Clone Projects

Now that the number of projects has become manageable, we clone the selected repositories into our local environment to calculate the metrics that we use for our case study. Unfortunately, some of the projects that we select from the GHTorrent dataset are no longer available to be cloned (e.g., deleted repositories). Thus, we cannot include such projects in our dataset. Nonetheless, we could clone 4,154 projects.

(3) Filter Duplicated Projects

Even after handling explicitly forked repositories, there are still some duplicate repositories hosted on GitHub (i.e., cloned and registered repositories that were not created using the GitHub fork feature). Such projects do not count as fork projects, but those projects have largely the same history as their originals. Since such projects will introduce noise in our dataset, we first detect them using the steps below, and then filter them out of our dataset.

We use the hashes of commits (SHAs) recorded in the Git repositories to identify duplicated projects. We consider any repositories that shares more than 70% of the same commit SHAs as a copied repository. We remove both repositories from our dataset because it is often difficult to determine which repository is the original one and which one is the copy.

After removing these repositories, 3,533 projects remain in our dataset.

(4) Calculate Metrics from Repositories

For the remaining projects, we calculate the metrics that are listed below in order to perform our case study.

LOC. We use *cloc*[5] to calculate LOC. Our LOC count does not include code comments or blank lines.

Total Commits. We count the number of commits by using the `git log` command with the `--no-merges` option.

Total Authors. We identify the unique authors by author name and email address, which we are able to extract from the commit logs. We use a tool[6] to disambiguate author names and email addresses. We disambiguate names and email addresses of authors because some developers appear with slightly different forms [28].

Age. We calculate the age of a project (in days) by subtracting the time of the latest commit from the time of the initial commit.

(5) Filter Projects by Metrics

We not only filter projects that have fewer than 10 developers, but similar to Bissyande *et al.* [4], we also filter projects that have fewer than 1,000 LOC.

Finally, we obtain a dataset that includes 2,496 GitHub projects for the commit-based and LOC-based core developer heuristic. Since 1,284 of these projects do not have the information that is needed to detect contributors with write access (*cf.* Section 5.2.1), only the remaining 1,212 projects are studied using the access-based heuristic.

5.3 Study Results

In this section, we present the results of our study with respect to our two research questions. For each research question, we present our approach and our results.

[5]`http://cloc.sourceforge.net/`
[6]`https://github.com/bvasiles/ght_unmasking_aliases`

(RQ1) Does the proportion of core developers of GitHub projects follow the Pareto principle?

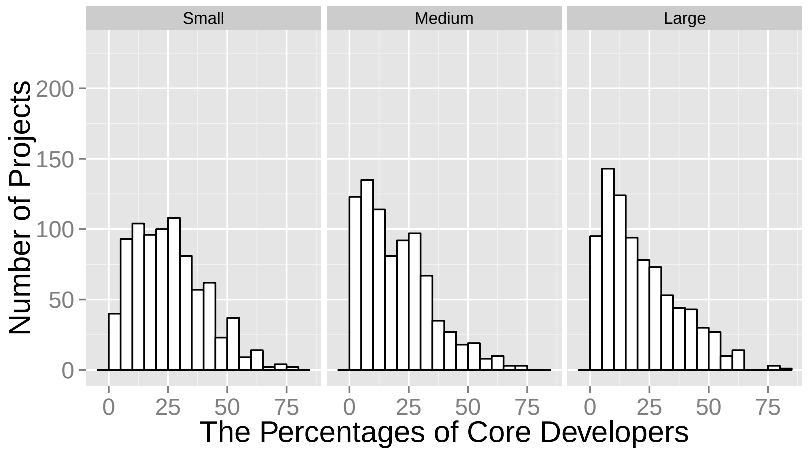
We begin our study by measuring the proportion of developers who are active enough to be considered core developers.

Approach. To address our first research question, we calculate the proportion of the development team that is considered to be part of the core team (*cf.* Section 5.2) of each studied project. Then, we use histograms to study the distribution of core team sizes in the studied projects.

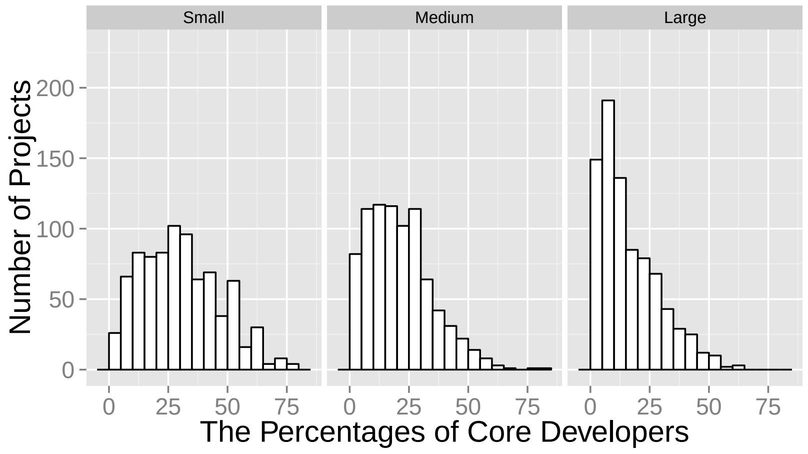
The Pareto principle or the so-called "80-20 rule" states that 80% of the contributions are performed by roughly 20% of the contributors. In this study, similar to prior work [64, 80], we add a window of flexibility, considering projects where the core team proportion is 20% ± 10% as being compliant with the Pareto principle. Indeed, Mockus *et al.* [64] showed that the core team proportions of modules in the Mozilla project are roughly 19%-25%. Moreover, Robles *et al.* showed that 10%-20% of developers produced more than 50% activities (in many cases as much as 90% or 95%).

We address RQ1 using two analyses. First, we analyze the distributions of proportions of core developers. Then, we split up the projects according to three confounding factors. Since the core team characteristics of smaller projects likely differs from those of larger projects, we divide the dataset into three strata (small, medium, and large) along three confounding factors (system size, team size and project age). We evenly divide the dataset accordingly, i.e., each stratum includes 832 projects for the commit-based and LOC-based heuristics, and each stratum includes 404 projects in the access-based heuristic. We then plot histograms of the core team proportions of projects in each of these nine strata. In this chapter, we do not show the plots of overall distribution to conserve space because we find that the distributions of the medium strata follow the same trends as the overall distributions.

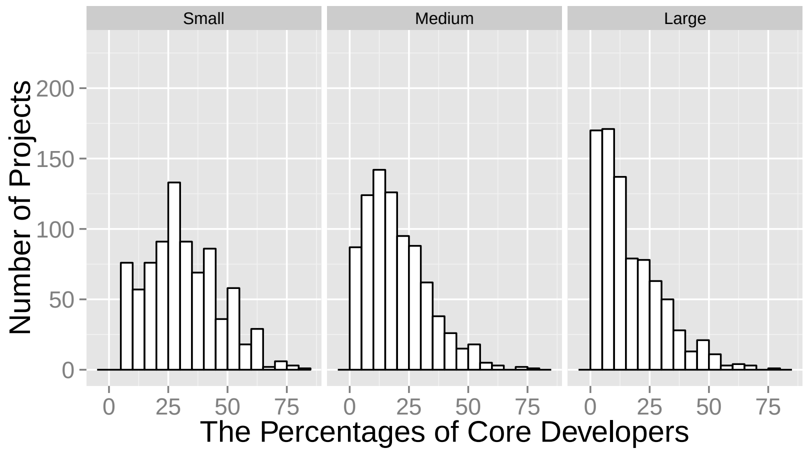
Results. Figure 5.3-5.5 show the core team distributions of the studied projects. Table 5.2



(a) LOC

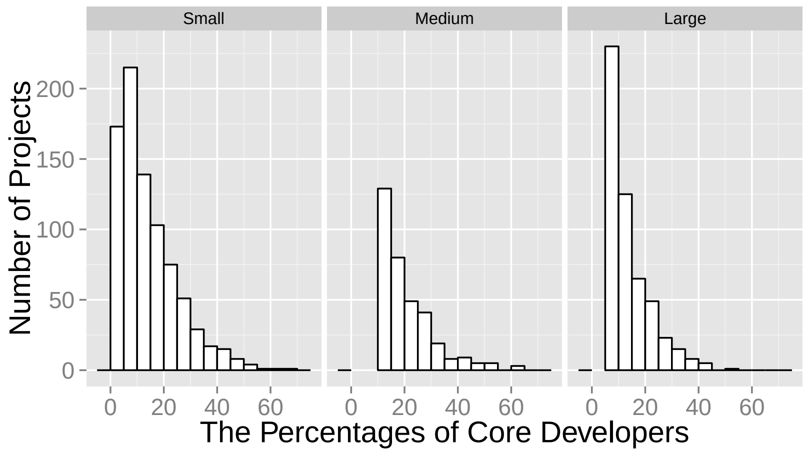


(b) Age

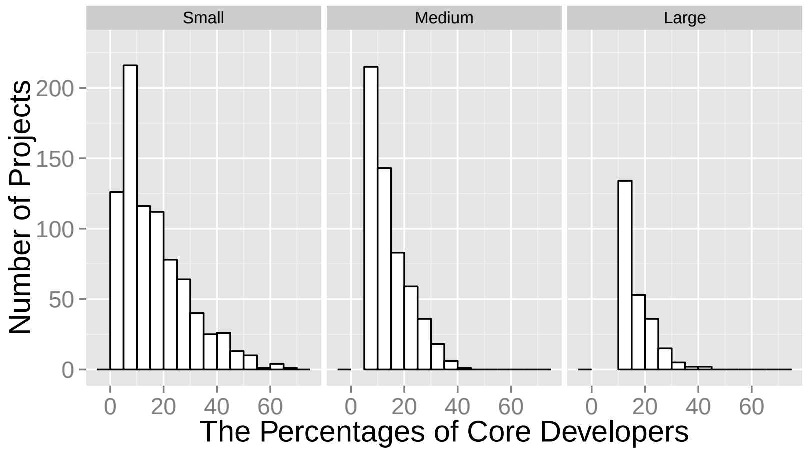


(c) Total Authors

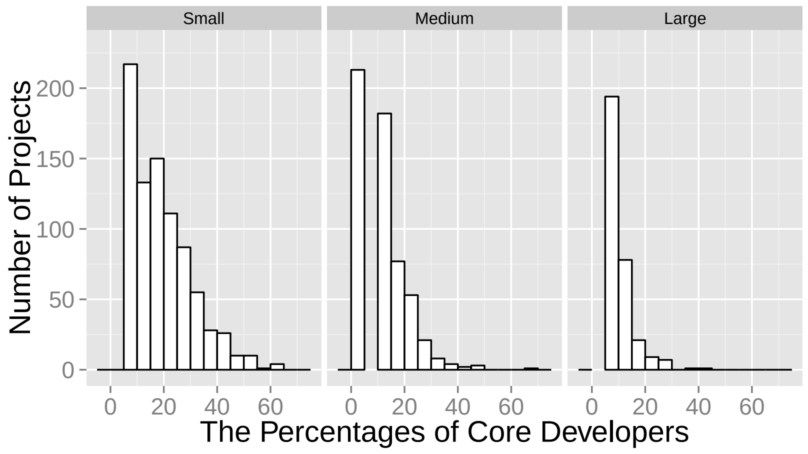
Figure 5.3: Distribution of projects of each size categories (Commit-Based)



(a) LOC

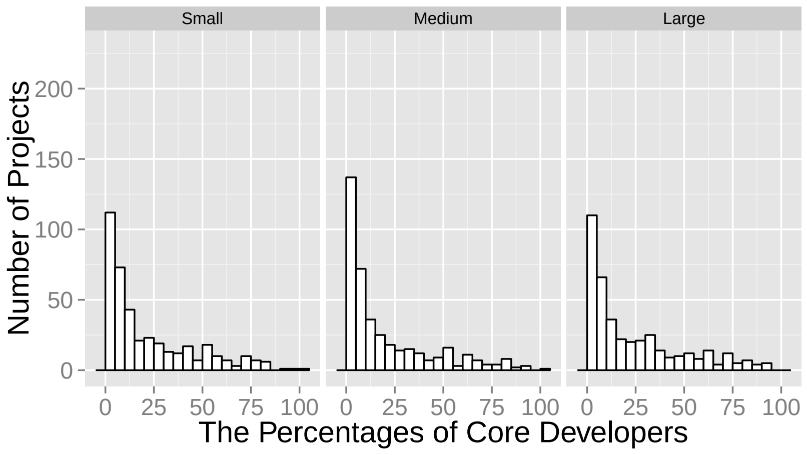


(b) Age

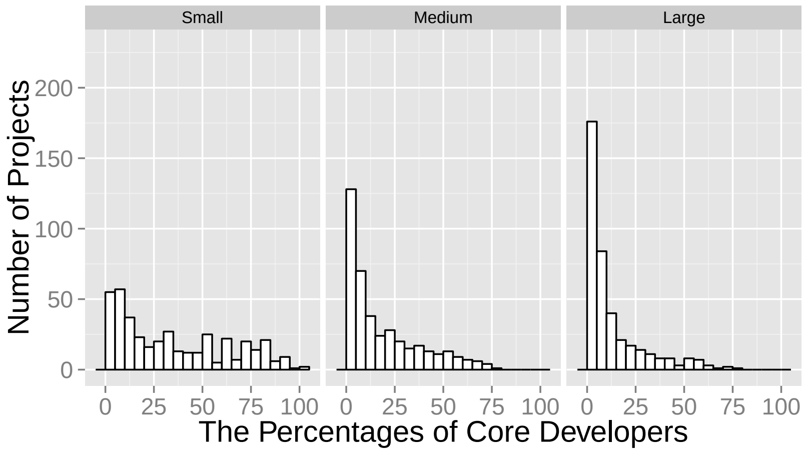


(c) Total Authors

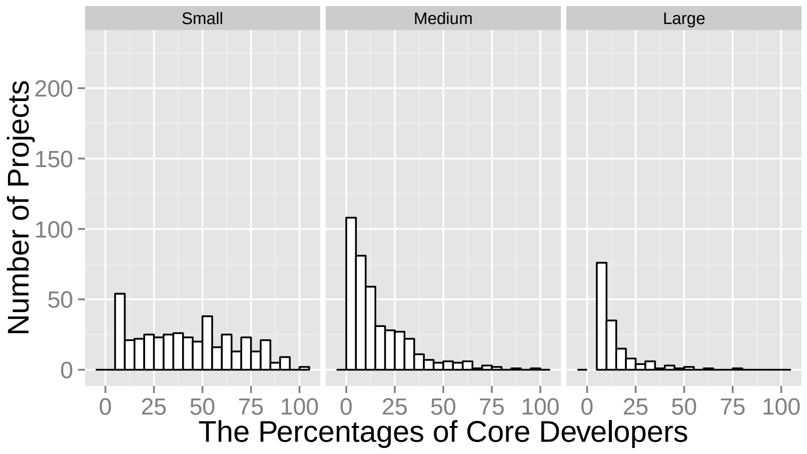
Figure 5.4: Distribution of projects of each size categories (LOC-Based)



(a) LOC



(b) Age



(c) Total Authors

Figure 5.5: Distribution of projects of each size categories (Access-Based)

Table 5.2: The spread of projects among strata of project size and age

Heuristic	Size Metrics	Stratum	Proportion of Core Developers		
			0%-10%	10%-30%	30%-100%
Commit-Based	LOC	Small	143 (17%)	411 (49%)	278 (33%)
		Medium	264 (32%)	389 (47%)	179 (22%)
		Large	242 (29%)	368 (44%)	222 (27%)
	Age	Small	94 (11%)	359 (43%)	379 (46%)
		Medium	203 (24%)	447 (54%)	182 (22%)
		Large	352 (42%)	362 (44%)	118 (14%)
	Total Authors	Small	80 (10%)	365 (44%)	387 (47%)
		Medium	224 (27%)	449 (54%)	159 (19%)
		Large	345 (41%)	354 (43%)	133 (16%)
	General		649 (26%)	1,168 (47%)	679 (27%)
LOC-Based	LOC	Small	403 (48%)	367 (44%)	62 (7%)
		Medium	487 (59%)	304 (37%)	41 (5%)
		Large	557 (67%)	253 (30%)	22 (3%)
	Age	Small	354 (42%)	376 (45%)	102 (12%)
		Medium	501 (60%)	316 (38%)	15 (2%)
		Large	592 (71%)	232 (28%)	8 (1%)
	Total Authors	Small	227 (27%)	497 (60%)	108 (13%)
		Medium	502 (60%)	315 (38%)	15 (2%)
		Large	718 (86%)	112 (13%)	2 (0.2%)
	General		1,447 (58%)	924 (37%)	125 (5%)
Access-Based	LOC	Small	192 (48%)	100 (25%)	112 (28%)
		Medium	211 (52%)	92 (23%)	101 (25%)
		Large	177 (44%)	98 (24%)	129 (32%)
	Age	Small	115 (28%)	94 (23%)	195 (48%)
		Medium	202 (50%)	107 (26%)	95 (24%)
		Large	263 (65%)	89 (22%)	52 (13%)
	Total Authors	Small	60 (15%)	87 (22%)	257 (64%)
		Medium	191 (47%)	143 (35%)	70 (17%)
		Large	329 (81%)	60 (15%)	15 (4%)
	General		580 (48%)	290 (24%)	342 (28%)

shows the exact numbers of projects of each category and percentile. In Table 5.2, the gray colored columns show the Pareto-compliant range.

Contrary to prior results, we find that the core team size of projects distributes broadly. Figure 5.3-5.5 and Table 5.2 show that the distributions are different according to the heuristic. Indeed, unlike prior work [30, 52, 83], we find that there are many projects that fall outside of our range of Pareto compliance (10%-30%).

When we focus on each heuristic and confounding factor, we observe the following trends.
Commit-Based: Table 5.2 shows that, irrespective of the stratum, 43%-54% of the studied projects are Pareto compliant. When controlling for project age and team size, we find that the number of projects with the smallest core team size (i.e., 0%-10%) increases as we shift from the smallest to largest strata. On the other hand, this trend is not as extreme in the system size strata. Therefore, we conclude that project age and team size have a larger impact on the core team proportion than system size does.
LOC-Based: The LOC-based heuristic is more right skewed than the commit-based heuristic. Similar to the commit-based heuristic, Table 5.2 shows that the right skew increases as the system size increases. Moreover, the total number of authors seems to impact to the core team proportion because the difference between small and large stratum is the largest among the three studied metrics.
Access-Based: Figure 5.5 shows that the distributions of the access-based heuristic are similar to those of the LOC-based heuristic. However, there are more projects that fall in the 30%-100% range for the access-based heuristic than the LOC-based heuristic. Similar to the commit-based heuristic (Table 5.2), age and team size also appear to have an impact on the core team proportion of the access-based heuristic.

Figure 5.6 shows the number of core developers. In Figure 5.6, the x-axis shows the number of core developers and the y-axis shows the number of projects. Table 5.3 shows the breakdown of projects stratified by the number of core developers.

From the perspective of core team size, we find support for the findings of prior studies [19,

Table 5.3: Distributions of projects according to the number of core developers

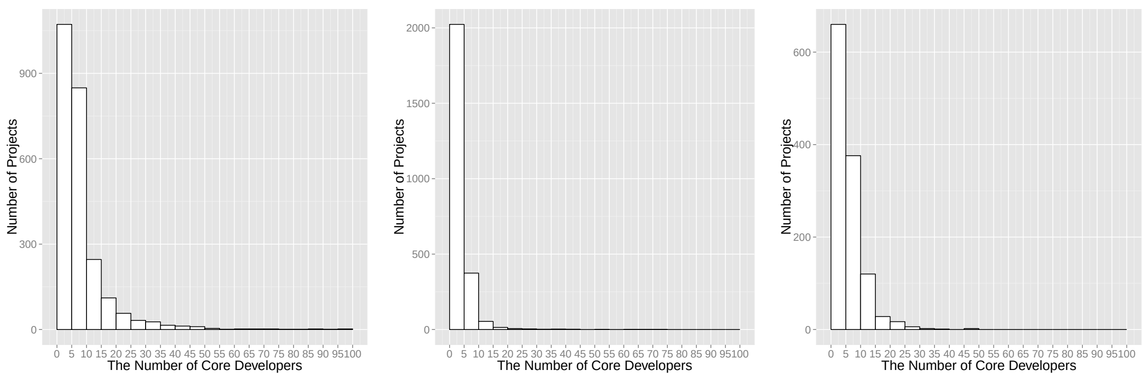
Number of Core Developers	1-9	10-15	16-20	21-50	51-100	101-
Commit-Based	1,924 (77%)	273 (11%)	98 (4%)	137 (5%)	17 (0.7%)	47 (2%)
LOC-Based	2,397 (96%)	57 (2%)	15 (0.6%)	13 (0.5%)	4 (0.1%)	10 (0.5%)
Access-Based	1,036 (85%)	128 (11%)	24 (2%)	24 (2%)	0 (0%)	0 (0%)

52, 64]. Mockus *et al.* argue that if the core team uses only an informal means of coordinating, the group will be no larger than 10-15 people [64]. Conversely, Dinh-Trong and Bieman [19] find that 28-42 developers provide 80% of the contributions in the FreeBSD project. Koch and Schneider [52] find that 52 developers provide 80% of the contributions in GNOME project.

88%-98% of projects have fewer than 16 core developers. Unlike the proportion of core developers, the distributions of the number of core developers are similar across the studied heuristics. Indeed, Table 5.3 shows that 88%-98% of the studied GitHub projects have fewer than 16 core developers.

We further analyze the 2%-12% of projects that have more than 15 core developers to find out what kind of projects have larger core teams. When using the commit-based heuristic, 275 out of the 299 projects that have more than 15 core developers are categorized in large stratum of total authors and the remaining 24 projects are in medium stratum. When using the LOC-based heuristic, 41 of the 42 projects are in large stratum of total authors and the remaining one project is in medium stratum. When using the access-based heuristic, 27 of the 48 projects are in large stratum of total authors, and 20 of the remaining 21 projects are in medium stratum. These observations indicate that most of the projects that have many core developers also tend to a larger pool of contributors than the other projects.

Contrary to prior work, we find that there are several projects that have larger or smaller core team proportion than we consider to be compliant with the Pareto principle. Moreover, we find that most projects have 15 or fewer members of the core team.



(a) Commit-Based (b) LOC-Based (c) Access-Based

Figure 5.6: The distribution of projects according to the number of core developers

Table 5.4: Keywords used to classify commits [34]

Development	
Activity Type	Keywords
Forward Engineering	implement, add, request, new, test,start, includ, initial, introduc, creat, increas
Maintenance	
Activity Type	Keywords
Reengineering	optimiz, adjust, update, delet, remov, chang, refactor, replac, modif, (is, are) now, enhance, improv, design change, renam, eliminat, duplicat, restructur, simplif, obsolete, rearrang, miss, enhanc
Corrective Engineering	bug, fix, issue, error, correct, proper, deprecat, broke
Management	clean, license, merge, release, structure, integrat, copyright, documentation, manual, javadoc, comment, migrat, repository, code review, polish, upgrade, style, formatting, organiz, TODO

(RQ2) Is there any difference between the contribution activity of core and non-core developers?

To address this RQ, we compare the types of contributions that are performed by core and non-core developers.

Approach. Previous studies have explored the purposes of changes [34, 38, 72]. In this study, we adopt Hattori and Lanza's approach to identify the purpose of changes. Hattori and Lanza [34] proposed a lightweight approach to classify each commit into development or maintenance activities based on the accompanying commit messages. They defined four main activities: *forward engineering* as development activity; and *reengineering*, *corrective engineering* and *management* as maintenance activity. They also provide keywords that are indicative of the type of activity (Table 5.4). *Forward engineering* activities implement new requests and add new features. *Reengineering* activities are related to refactoring, redesign and other actions to enhance the quality of the code. *Corrective engineering* activities fix defects. *Management* activities are other general maintenance activities that are not related to system functionality, such as code reformatting and documentation.

To ensure that the classification provided by Hattori and Lanza is sufficient for our dataset, we manually analyze a randomly selected sample of 384 commit comments. The sample is selected such that it provides a confidence level of 95% with a confidence interval of ±5%. The manual analysis reveals that some commits have an empty commit comment. We classify such commits as *empty*.

Hattori and Lanza's approach searches for keywords in commit messages in the following order: empty comments, management, reengineering, corrective engineering and forward engineering. The commit comments of so-called tangled changes [37] can match multiple purpose keyword types. For example, a developer can clean up code and fix a bug within one commit. For these commits, the approach classifies the commit according to the keyword that is found first (e.g., the commit described above is classified into reengineering activity). The commits that could not be classified into any of the classes are marked as *unknown*.

Using Hattori and Lanza's approach, we classify and compare the distributions of activities of core and non-core developers. In total, our commit-based and LOC-based heuristic datasets includes 5,746,523 commits, and our access-based one contains 2,865,461 commits.

Results. Table 5.5 shows the distribution of activities of core and non-core team members. Interestingly, the total number of commits that are contributed by core and non-core team members are very similar when we use either commit-based and LOC-based heuristics. On the other hand, the access-based heuristic shows that the number of commits of core developers is

Table 5.5: Developer activity

Type of Activity	Commit-Based		LOC-Based		Access-Based	
	Core	Non-Core	Core	Non-Core	Core	Non-Core
Forward Engineering	15%	18%	16%	18%	17%	16%
Reengineering	29%	30%	29%	30%	24%	30%
Corrective Engineering	20%	21%	20%	22%	18%	21%
Management	14%	13%	14%	13%	12%	15%
Empty	0.1%	0.1%	0.1%	0.1%	0.3%	0.1%
Unknown	22%	17%	22%	17%	30%	19%
Total #of Commits	4,692,063	1,054,460	4,739,121	1,007,402	931,265	1,934,196

less than that of non-core developers. In this study, we only consider the authors of commits. Hence, this discrepancy between core and non-core contributions might show that many of the access-based core developers focus on integration work rather than writing code.

The proportions of contribution activity of core and non-core developers are similar. Irrespective of the core team heuristic, we find that the distributions of activities are very similar. Reengineering accounts for the largest proportion of activity for both core and non-core developers, with proportions ranging between 24%-30%. In the other type of activities, the difference between the proportion of activity of core and non-core developers is at most 6 percentage points. Therefore, we conclude that the difference in activity proportions between core and non-core is negligible.

The proportions of contribution activity of core and non-core developers are similar.

5.4 Discussion

5.4.1 The Bus Factor

We find that more than half of the studied projects have a core team comprised of (at most) 20% of the pool of active developers and more than 88% of the studied projects have a core team of (at most) 15 developers. These results indicate that many projects have a low *bus factor* [12, 77, 93], i.e., face the risk of key personnel leaving the project. Ye and Kishida [109] find that development of GIMP was once halted because a key core developer left the project. To avoid such cases, projects must share knowledge among developers.

On the other hand, similar to the work of Dinh-Trong and Bieman [19], we find that there are projects that have large core teams. In this study, we just show the distribution and do not investigate each of the projects deeply. In future work, we plan to conduct a deeper analysis of projects with large core teams. For example, investigating whether or not such projects have well-defined mechanisms for developer promotion rather than the informal arrangements that Mockus *et al.* [64] hypothesized could yield fruitful results.

5.4.2 Core and Non-core Developer Activity

Prior work [64] hypothesized that a group larger by an order of magnitude than the core team will repair defects. If the hypothesis is true, we assumed that the proportion of maintenance activity of non-core developers is large. However, our results show that both types of developers have similar proportions of development activities. Furthermore, when we consider the number of corrective engineering commits, the number of the commits by core developers is much larger than that by non-core developers.

Our results may be a characteristic of the GitHub development environment. With the growth of social coding platforms (e.g., GitHub), the nature of core teams in modern OSS projects may have changed. For example, GitHub projects boast a higher rate of acceptance for contributions than the OSS projects of the past did. Indeed, while Jiang *et al.* [47] find

Table 5.6: The proportion of projects that are Pareto compliant when we use other threshold values.

Metric	Threshold	#ofProjects	Proportion
Total Authors	5	2,526	46%
	20	1,664	49%
LOC	500	2,685	46%
	2,000	2,220	47%

that only 33% of contributions are eventually integrated into the Linux kernel (one of the largest OSS projects, which mainly developed by outside of GitHub), Gousios *et al.* [32] find that 84% of contributions are eventually integrated into GitHub projects.

5.4.3 The Impact of Thresholds

In this study, we filter projects to remove immature software projects by using some thresholds, i.e., the total authors and LOC (*cf.* Section 5.2.3). As such, our results may be sensitive to these thresholds. To check for threshold sensitivity, we re-apply our analysis using other threshold values (total authors = 5, 20 and LOC = 500, 2,000) and discuss changes to our results below.

Table 5.6 shows the proportion of projects that are Pareto compliant when we vary the thresholds. Irrespective of the threshold, similar to our results in Section 5.3, we observe that more than half of projects are not Pareto compliant. These results suggest that while our results slightly vary when the thresholds change, the main conclusions are not heavily impacted.

5.5 Threats to Validity

5.5.1 Construct Validity

In this chapter, we adopt three heuristics to identify core developers. The commit-based and LOC-based heuristics are based on the amount of contribution to the product. Even though there are a lot of metrics that can capture contribution units, the amount of contribution is one of the most basic metrics that is used to identify core developers. Moreover, previous studies that focus on core contributors [19, 28, 30, 52, 64, 83] also conduct their analysis from the perspective of the amount of contribution. Therefore, we feel that these heuristics are appropriate for our context.

On the other hand, the access-based heuristic does not depend on the amount of contribution. However, the access-based definition is also one of the most basic indicators of core developers. Indeed, the developers who have write access to the main repository have enough knowledge about the product to manage other developers' contributions.

5.5.2 Internal Validity

Our results for RQ1 are dependent on our heuristics for identifying core developers. In this study, we used 80% of the total contributions as our threshold for identifying core developers, since this threshold was also used by previous studies [19, 28, 30, 52, 64, 83]. While we begin a threshold sensitivity analysis in Section 5.4, we plan to perform a carefully controlled sensitivity analysis in future work.

Furthermore, our analysis is time-agnostic. Since development teams are changing over time, the number of core developers may vary as well. We plan to conduct a temporal analysis of core teams in future work.

5.5.3 External Validity

In this study, we filter away projects that have less than 10 developers or less than 1,000 LOC to remove projects that are immature [4, 48]. Therefore, our results may not generalize to legitimate software projects with a small number of contributors.

5.6 Summary

Open Source Software (OSS) projects depend heavily on core developers, i.e., team members that produce 80% of the contributions to a project. Prior studies have found that core development teams tend to follow the Pareto principle (a.k.a., the 80-20 rule), i.e., 80% of the contributions are produced by roughly 20% of the contributors. However, these prior studies were performed on small samples of systems. With the recent growth in popularity of the social coding paradigm, a plethora of data is becoming available for researchers to explore core team dynamics within. Therefore, we revisit the analyses of previous work on a large sample of GitHub projects.

To that end, in this chapter, we study core development teams on GitHub. Through a case study of 2,496 GitHub projects, we observe that:

- The core teams of many GitHub projects are not compliant with the Pareto principle.
- While some GitHub projects have core teams that are too large to be Pareto compliant, many more have very small core teams, consisting of fewer than 10% of the pool of contributors.
- From the perspective of number of core developers, more than 88% of projects have less than 15 core developers.
- Core and non-core developers participate in maintenance and future development activities in similar proportions.

These observations suggest that it is difficult to infer the sustainability of the projects based on whether or not the proportion of core developers follows the Pareto principle. On the other hand, the number of core developers might be an indicator of sustainable OSS projects as we discussed in Section 5.4.

Chapter 6

Discussion

6.1 Introduction

In this chapter, we summarize our findings, discuss the results of this dissertation, and provide implications for researchers, OSS developers, users, and enterprises.

6.1.1 Summary of findings

In Chapters 3 and 4, we introduced magnet and sticky metrics and demonstrated their applicability. We not only provided the distribution of these metrics, but also classified projects into four categories (i.e., attractive, fluctuating, stagnant and terminal) by comparing the metrics values of projects and the transitions between the categories.

From the study, we found that

- Larger project attract and retain more developers
- 53% of terminal projects eventually decay into a state of fewer than ten developers
- 55% of attractive projects remain in the attractive quadrant
- Only 13% of projects that are in the first period (first 6 months) become projects that have more than ten developers

These findings suggest that it is possible to capture features of sustainability of OSS projects by using both evolvability and stability measures. Moreover, it is possible to infer the sustainability of OSS projects by comparing the metrics values of OSS projects.

In Chapter 5, we focused on core developers and revisited the results of prior studies on the Pareto principle. From the study, we found that

- The proportions of core developers in OSS projects do not follow the Pareto principle
- More than 88% of projects have less than 15 core developers

With regard to the proportions of core developers, the finding is different from prior findings [30, 52, 83]. Prior studies had claimed that the proportion of core developers in successful projects follow the Pareto principle [30, 52, 83]. As most OSS projects have a small number of core developers, we suppose that losing core developers would have a large impact to the sustainability of the OSS projects.

6.2 Implication and Future Research Direction

6.2.1 Researchers

The results discussed in Chapter 5 that the proportions of core developers do not follow the Pareto principle, in contrast to prior studies. Hence, it is difficult to identify the sustainability of OSS projects from the Pareto principle. As further research, studying the impact of core developer retention might be interesting (i.e., combination of sticky and core developers). Since core developers are more active than other developers, their knowledge about the project might be greater. When core developers leave the project, the project suffers more compared with when other developers leave. Therefore, it is important to investigate the effects of core developer retention.

As discussed in Chapter 5, investigating the impact of the number of core developers in the project is another research direction. Because of the similarity of concepts between bus factor [2, 12, 77, 93] and core developers, a small core team means that the project has a low bus factor, i.e., it faces the risk of key personnel leaving the project. Hence, we assume that the number of core developers becomes an indicator of sustainability.

Furthermore, applying the concept of relationship between projects for understanding the sustainability of OSS projects would be important. In this dissertation, we introduced the magnet and sticky metrics from social sciences [70] and demonstrated their applicability. Since resources that developers can devote to OSS projects are limited, we assumed that the relationship between OSS projects is important for understanding their sustainability. Therefore, we classified OSS projects into four categories by comparing values of these metrics in addition to just measuring them. From the results of a case study, these metrics successfully measure the sustainability of OSS projects. We believe that considering the relationship between OSS projects is important for understanding their sustainability.

As mentioned in Chapter 2, most prior studies did not consider relationships between projects. Therefore, applying relationships between OSS projects to indicators that were proposed and evaluated in prior studies is another future research direction.

6.2.2 OSS Developers

OSS developers can be classified into two types: those who manage OSS projects, such as project leaders and core developers, and those who are just participants.

Our approach would be helpful for OSS developers who manage OSS projects to understand current situations of their projects. By understanding the current situation, they can respond to their problems. If the magnet value of the project is low, they need to attract new developers [53, 62, 85]. If the sticky value of the project is low, they need to remove social barriers for new developers to retain new developers [90] and operate the project to increase the number of long-term contributors [24, 63, 73, 86, 111].

In addition to the magnet and sticky metrics, it is important to consider the number of core developers, especially OSS developers who lead the project, such as a project leader.

If the number of core developers in a OSS project is low, the OSS project faces the risk of abandonment because the number of core developer is a similar concept to the bus factor (also known as truck factor) [2, 12, 77, 93, 110], as mentioned in Chapter 5. Although the relationship between the bus factor and sustainability of an OSS project is not empirically evaluated, the importance of the bus factor has been pointed out in the agile field anecdotally.[1] Hence, in such situation, OSS developers should operate the project to increase the number of core developers or retain the core developers.

For OSS developers who are just participants, the magnet and sticky metrics could be an indicator to select the OSS projects suitable for their goals. One of the goals of an OSS developer is learning [40]. For developers with such a goal, projects with high magnet and high sticky values may be able to provide greater learning opportunities because as that the projects become larger, there are more opportunities for developers to learn from one another [9]. Another point for such developers is the experiences of developers. A high sticky value indicates the presence of experienced developers in the project. Therefore, OSS developers can learn about the project from the current developers.

A high magnet value indicates the presence of new developers in the project. Since there are new developers, the project may have influx of new ideas from the new developers and therefore the project might be active. Hence, the OSS developers can learn about new ideas and submit new ideas easily in such a project.

6.2.3 Users and Enterprises

End-users and enterprises want to avoid selecting OSS projects that will be abandoned in near future or have low quality. Therefore, it is important to assess OSS projects before applying them to their systems.

Prior studies have proposed OSS quality assessment models, which provide a set of indicators for the quality of OSS [1, 21, 66, 68, 69, 99]. Similar to quality assessment, sustainability

[1]`http://www.agileadvice.com/2005/05/15/agilemanagement/truck-factor/`

assessment is important for industrial developers to avoid additional costs. By using the magnet and sticky (i.e., evolvability and stability) measures, industrial developers can assess the sustainability of OSS projects. For example, we found that 53% of terminal projects eventually decay into a state of a small number of developers. End-users and enterprises should avoid such terminal OSS projects.

On the other hand, 55% of attractive projects remain attractive. Furthermore, only 3% of attractive projects decay into the state of a small number of developers. Therefore, end-users and enterprises can consider such attractive projects as safer projects (i.e., they might not be abandoned).

Moreover, if the target projects are in the first period, end-users and enterprises should not consider applying them because only 13% of new projects attract large communities. In addition to the current situation of OSS projects that is measured in the latest period, industrial developers can know the transitions of states. Such information would be helpful for selecting OSS projects.

Chapter 7

Conclusion

7.1 Contributions

In this dissertation, we studied sustainability of OSS projects from the perspectives of evolvability and stability.

In summary, the contributions are as follows:

- We introduced the magnet and sticky metrics to quantitatively evaluate project sustainability. For evolvability, we study the magnet metric, which is calculated as the proportion of new developers who participate in a target project among all new developers. For stability, we study the sticky metric, which is calculated as the proportion of developers who are retained in the same project. (Chapter-3, 4)
- We empirically demonstrated the applicability of the magnet and sticky measures with a large set of GitHub projects. We find that 53% of the projects with smaller smaller magnet and sticky values than the median values of those metrics eventually decay into a state of less than ten developers. On the other hand, 55% of the projects with larger values than median values maintain their popularity. These findings suggest that by using both evolvability and stability measures, it is possible to capture features of sustainability of OSS projects. Moreover, it is possible to quantitatively measure the

evolvability and stability of OSS projects by using these metrics. (Chapter-4)

- We empirically studied the impact of structural ratio of core developers on sustainability of OSS projects. In contrast to prior findings, we find that core team proportions do not follow the Pareto principle. Moreover, we find that the core team of most projects has 15 or fewer members. (Chapter-5)

In addition to the above-mentioned contributions, we also provided implications and future research directions for each of stakeholders (i.e., researchers, OSS developers, users, and enterprises) in Chapter 6.

Bibliography

[1] Atos. Qualification and selection of open source software (QSOS), version 2.0. 2013. `http://backend.qsos.org/download/qsos-2.0_en.pdf`.

[2] G. Avelino, M. T. Valente, and A. Hora. What is the truck factor of popular github applications? a first assessment, 2015. `https://doi.org/10.7287/peerj.preprints.1233v2`.

[3] C. Bird, A. Gourley, P. Devanbu, A. Swaminathan, and G. Hsu. Open borders? immigration in open source projects. In *Proceedings of the Fourth International Workshop on Mining Software Repositories (MSR)*, pages 6–13, 2007.

[4] T. Bissyande, D. Lo, L. Jiang, L. Reveillere, J. Klein, and Y. le Traon. Got issues? who cares about it? a large scale investigation of issue trackers from github. In *Proceedings of the 24th IEEE International Symposium on Software Reliability Engineering (ISSRE)*, pages 188–197, Nov 2013.

[5] A. Bonaccorsi and C. Rossi. Why open source software can succeed. *Research Policy*, 32(7):1243–1258, 2003.

[6] A. Bonaccorsi and C. Rossi. Comparing motivations of individual programmers and firms to take part in the open source movement: From community to business. *Knowledge, Technology & Policy*, 18(4):40–64, 2006.

[7] B. S. Butler. Membership size, communication activity, and sustainability: A resource-

based model of online social structures. *Information Systems Research*, 12(4):346–362, 2001.

[8] S. Chengalur-Smith and A. Sidorova. Survival of open-source projects: A population ecology perspective. In *Proceedings of International Conference on Information Systems*, 2003.

[9] S. Chengalur-Smith, A. Sidorova, and S. L. Daniel. Sustainability of free/libre open source projects: A longitudinal study. *Journal of the Association for Information Systems*, 11(11), 2010.

[10] W. W. Chin and P. R. Newsted. Structural equation modeling analysis with small samples using partial least squares. *Statistical strategies for small sample research*, 2:307–342, 1999.

[11] J. Colazo and Y. Fang. Impact of license choice on open source software development activity. *Journal of the American Society for Information Science and Technology*, 60(5):997–1011, 2009.

[12] V. Cosentino, J. L. C. Izquierdo, and J. Cabot. Assessing the bus factor of git repositories. In *Proceedings of International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, pages 499–503, 2015.

[13] K. Crowston, J. Howison, and H. Annabi. Information systems success in free and open source software development: Theory and measures. *Software Process–Improvement and Practice*, 11(2):123–148, 2006.

[14] K. Crowston, K. Wei, Q. Li, and J. Howison. Core and periphery in free/libre and open source software team communications. In *Proceedings of Hawai'i International Conference on System Science (HICSS)*, 2006.

[15] L. Dabbish, C. Stuart, J. Tsay, and J. Herbsleb. Social coding in github: Transparency

and collaboration in an open software repository. In *Proceedings of Conference on Computer Supported Cooperative Work (CSCW)*, pages 1277–1286, 2012.

[16] P. A. David and J. S. Shapiro. Community-based production of open-source software: What do we know about the developers who participate? *Information Economics and Policy*, 20(4):364 – 398, 2008. Empirical Issues in Open Source Software.

[17] P. A. David, J. S. Shapiro, and S. Arora. FLOSS-US: the free/libre/open source software survey for 2003. `http://www-siepr.stanford.edu/programs/OpenSoftware_David/FLOSS-US-Report.pdf`.

[18] L. W. P. David Krackhardt. When friends leave: A structural analysis of the relationship between turnover and stayers' attitudes. *Administrative Science Quarterly*, 30(2):242–261, 1985.

[19] T. Dinh-Trong and J. Bieman. The freebsd project: a replication case study of open source development. *IEEE Transactions on Software Engineering*, 31(6):481–494, June 2005.

[20] N. Ducheneaut. Socialization in an open source software community: A socio-technical analysis. *Comput. Supported Coop. Work*, 14(4):323–368, 2005.

[21] F.-W. Duijnhouwer and C. Widdows. Open source maturity model. In *Capgemini Expert Letter*. 2003. `https://jose-manuel.me/thesis/references/GB_Expert_Letter_Open_Source_Maturity_Model_1.5.3.pdf`.

[22] J. Edwards. Person-job fit: A conceptual integration, literature review, and methodological critique. *International Review of Industrial and Organizational Psychology*, 1991.

[23] R. English and C. M. Schweik. Identifying success and tragedy of floss commons: A preliminary classification of sourceforge.net projects. In *Proceedings of International*

Workshop on Emerging Trends in FLOSS Research and Development (FLOSS), pages 54–59, 2007.

[24] Y. Fang and D. Neufeld. Understanding sustained participation in open source software projects. *Journal of Management Information Systems*, 25(4):9–50, Apr. 2009.

[25] J. Feller and B. Fitzgerald. *Understanding Open Source Software Development.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002.

[26] M. Foucault, M. Palyart, X. Blanc, G. C. Murphy, and J.-R. Falleri. Impact of developer turnover on quality in open-source software. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering (ESEC/FSE)*, pages 829–841, New York, NY, USA, 2015. ACM.

[27] M. J. Gallivan. Striking a balance between trust and control in a virtual organization: a content analysis of open source software case studies. *Information Systems Journal*, 11(4):277–304, 2001.

[28] J. Geldenhuys. Finding the core developers. In *Proc. of the 36th Euromicro Conference on Software Engineering and Advanced Applications*, pages 447–450. IEEE Computer Society, Sept. 2010.

[29] D. M. German. A study of the contributors of postgresql. In *Proceedings of International Workshop on Mining Software Repositories (MSR)*, pages 163–164, 2006.

[30] M. Goeminne and T. Mens. Evidence for the pareto principle in open source software activity. In *Joint Porceedings of the 1st International Workshop on Model Driven Software Maintenance and 5th International Workshop on Software Quality and Maintainability*, pages 74–82, 2011.

[31] G. Gousios. The ghtorrent dataset and tool suite. In *Proceedings of International Working Conference on Mining Software Repositories (MSR)*, pages 233–236, 2013.

[32] G. Gousios, M. Pinzger, and A. v. Deursen. An exploratory study of the pull-based software development model. In *Proceedings of International Conference on Software Engineering (ICSE)*, pages 345–355, 2014.

[33] R. Grewal, G. L. Lilien, and G. Mallapragada. Location, location, location: How network embeddedness affects project success in open source systems. *Management Science*, 52(7):1043–1056, 2006.

[34] L. Hattori and M. Lanza. On the nature of commits. In *Proceedings of International Conference on Automated Software Engineering (ASE) - Workshops*, pages 63–71, Sept 2008.

[35] J. D. Herbsleb and A. Mockus. An empirical study of speed and communication in globally distributed software development. *IEEE Transactions on Software Engineering*, 29(6):481–494, June 2003.

[36] I. Herraiz, G. Robles, J. J. Amor, T. Romera, and J. M. González Barahona. The processes of joining in global distributed software projects. In *Proceedings of Int'l Workshop on Global Software Development for the Practitioner (GSD)*, pages 27–33, 2006.

[37] K. Herzig and A. Zeller. The impact of tangled code changes. In *Proceedings of the 10th Working Conference on Mining Software Repositories*, pages 121–130, 2013.

[38] A. Hindle, D. M. German, and R. Holt. What do large commits tell us?: A taxonomical study of large commits. In *Proceedings of the 2008 International Working Conference on Mining Software Repositories*, pages 99–108, 2008.

[39] A. Hindle, M. Godfrey, and R. Holt. Release pattern discovery via partitioning: Methodology and case study. In *Proceedings of International Workshop on Mining Software Repositories (MSR)*, pages 19–19, 2007.

[40] E. v. Hippel and G. v. Krogh. Open source software and the “private-collective” innovation model: Issues for organization science. *Organization science*, 14(2):209–223, 2003.

[41] B. Hosack and G. Sagers. Participation in oss projects: Does it support release early release often? In *MWAIS 2011 Proceedings*, 2011.

[42] M. A. Huselid. The impact of human resource management practices on turnover, productivity, and corporate financial performance. *Academy of management journal*, 38(3):635–672, 1995.

[43] Information-technology Promotion Agency, Japan. A field survey on businesses adopting open source software, 2010. (in Japanese).

[44] A. Iqbal. Understanding contributor to developer turnover patterns in oss projects: A case study of apache projects. *ISRN Software Engineering*, 2014:1–10, 2014.

[45] J. E. D. Jason D. Shaw, Nina Gupta. Alternative conceptualizations of the relationship between voluntary turnover and organizational performance. *The Academy of Management Journal*, 48(1):50–68, 2005.

[46] C. Jensen and W. Scacchi. Role migration and advancement processes in ossd projects: A comparative case study. In *Proc. Int'l Conf. on Software Engineering (ICSE)*, pages 364–374, 2007.

[47] Y. Jiang, B. Adams, and D. German. Will my patch make it? and how fast? case study on the linux kernel. In *Proceedings of the 10th IEEE Working Conference on Mining Software Repositories (MSR)*, pages 101–110, May 2013.

[48] E. Kalliamvakou, G. Gousios, K. Blincoe, L. Singer, D. M. German, and D. Damian. The promises and perils of mining github. In *Proceedings of Working Conference on Mining Software Repositories (MSR)*, pages 92–101, 2014.

[49] Y. Kamei, S. Matsumoto, A. Monden, K. Matsumoto, B. Adams, and A. E. Hassan. Revisiting common bug prediction findings using effort aware models. In *Proceedings of International Conference on Software Maintenance (ICSM)*, pages 1–10, 2010.

[50] F. Khomh, B. Adams, T. Dhaliwal, and Y. Zou. Understanding the impact of rapid releases on software quality. *Empirical Software Engineering*, 20(2):336–373, 2015.

[51] F. Khomh, B. Chan, Y. Zou, and A. E. Hassan. An entropy evaluation approach for triaging field crashes: A case study of mozilla firefox. In *Proceedings of International Working Conference on Reverse Engineering (WCRE)*, pages 261–270, 2011.

[52] S. Koch and G. Schneider. Effort, cooperation and coordination in an open source software project: Gnome. *Information Systems Journal*, 12(1):27–42, 2002.

[53] R. Kraut, M. Burke, J. Riedl, and P. Resnick. The challenges of dealing with newcomers. *MIT Press*, pages 179–230, 2012.

[54] S. Krishnamurthy. Cave or community? an empirical examination of 100 mature open source projects (originally published in volume 7, number 6, june 2002). *First Monday*, 0(0), 2005.

[55] K. Lakhani and R. Wolf. *Why Hackers Do What They Do: Understanding Motivation and Effort in Free/Open Source Software Projects.* MIT Press, Cambridge, 2005.

[56] A. M. S. Laurent. *Understanding Open Source and Free Software Licensing.* O'Reilly Media, Inc., 2004.

[57] J. Lerner and J. Tirole. Some simple economics of open source. *Journal of Industrial Economics*, 50:197–234, 2002.

[58] G. Madey and S. Christley. F/OSS research repositories & research infrastructures. *NSF Workshop on Free/Open Source Software Repositories and Research Infrastructures (FOSSRRI)*, 2008.

[59] A. A. Mary and S. K. Prasanth. Open source software survivability analysis using communication pattern validation. *IOSR Journal of Computer Engineering*, 12:114–118, 2013.

[60] S. Matsumoto, Y. Kamei, M. Ohira, and K. Matsumoto. Understanding open collabolation in OSS communities. *Journal of The Infosocionomics Society*, 3(2):29–42, 3 2009. (in Japanese).

[61] S. McIntosh, Y. Kamei, B. Adams, and A. E. Hassan. The impact of code review coverage and code review participation on software quality: A case study of the qt, vtk, and itk projects. In *Proceedings of Working Conference on Mining Software Repositories (MSR)*, pages 192–201, 2014.

[62] P. Meirelles, C. Santos Jr., J. Miranda, F. Kon, A. Terceiro, and C. Chavez. A study of the relationships between source code metrics and attractiveness in free software projects. In *Proceedings of the 2010 Brazilian Symposium on Software Engineering*, SBES '10, pages 11–20, Washington, DC, USA, 2010. IEEE Computer Society.

[63] V. Midha and P. Palvia. Retention and quality in open source software projects. *AMCIS 2007 Proceedings*, page 25, 2007.

[64] A. Mockus, R. T. Fielding, and J. D. Herbsleb. Two case studies of open source software development: Apache and mozilla. *ACM Trans. on Software Engineering and Methodology*, 11(3):309–346, 2002.

[65] K. Nakakoji, Y. Yamamoto, Y. Nishinaka, K. Kishida, and Y. Ye. Evolution patterns of open-source software systems and communities. In *Proceedings of International Workshop on Principles of Software Evolution (IWPSE)*, pages 76–85, 2002.

[66] O. Northeast Asia. RepOSS: A flexible OSS assessment repository. 2012.

[67] M. Ohira, N. Ohsugi, T. Ohoka, and K.-i. Matsumoto. Accelerating cross-project knowledge collaboration using collaborative filtering and social networks. In *Proceed-*

ings of the 2005 International Workshop on Mining Software Repositories (MSR), pages 1–5, 2005.

[68] E. Petrinja, R. Nambakam, and A. Sillitti. Introducing the opensource maturity model. In *Proc. of the International Workshop on Emerging Trends in Free/Libre/Open Source Software Research and Development*, pages 37–41, 2009.

[69] E. Petrinja, A. Sillitti, and G. Succi. Comparing OpenBRR, QSOS, and OMM assessment models. In *Proc. of the IFIP International Conference on Open Source Systems (OSS)*, pages 224–238. 2010.

[70] Pew Research Social & Demographic Trends. Magnet or sticky?: A state-by-state typology. `http://www.pewsocialtrends.org/2009/03/11/magnet-or-sticky/` (Accessed 2015-06-15).

[71] T. Preston-Werner. Semantic versioning 2.0.0. `http://semver.org` (Accessed 2015-06-15).

[72] R. Purushothaman and D. Perry. Toward understanding the rhetoric of small source code changes. *IEEE Transactions on Software Engineering*, 31(6):511–526, 2005.

[73] I. Qureshi and Y. Fang. Socialization in open source software projects: A growth mixture modeling approach. *Organizational Research Methods*, 14(1):208–238, 2011.

[74] U. Raja and M. Tretter. Defining and evaluating a measure of open source project survivability. *IEEE Transactions on Software Engineering*, 38(1):163–174, Jan 2012.

[75] A. Rastogi and A. Sureka. What community contribution pattern says about stability of software project? In *2014 21st Asia-Pacific Software Engineering Conference*, volume 2, pages 31–34, Dec 2014.

[76] E. S. Raymond. *The Cathedral and the Bazaar: Musings on Linux and Open Source*

by an Accidental Revolutionary. O'Reilly & Associates, Inc., Sebastopol, CA, USA, 1st edition, 1999.

[77] F. Ricca, A. Marchetto, and M. Torchiano. On the difficulty of computing the truck factor. In *Product-Focused Software Process Improvement*, volume 6759 of *Lecture Notes in Computer Science*, pages 337–351. 2011.

[78] D. Riehle. The economic motivation of open source software: Stakeholder perspectives. *Computer*, 40(4):25–32, 2007.

[79] D. Riehle, P. Riemer, C. Kolassa, and M. Schmidt. Paid vs. volunteer work in open source. In *Proceedings of Hawaii International Conference on System Sciences (HICSS)*, pages 3286–3295, 2014.

[80] G. Robles, J. Gonzalez-Barahona, and I. Herraiz. Evolution of the core team of developers in libre software projects. In *Proceedings of International Working Conference on Mining Software Repositories (MSR)*, pages 167–170, May 2009.

[81] G. Robles and J. M. Gonzalez-Barahona. *Contributor Turnover in Libre Software Projects*, pages 273–286. Springer US, Boston, MA, 2006.

[82] G. Robles, J. M. Gonzalez-Barahona, and J. J. Merelo. Beyond source code: The importance of other artifacts in software development (a case study). *Journal of Systtem Software*, 79(9):1233–1248, Sept. 2006.

[83] G. Robles, S. Koch, J. M. Gonzlez-Barahona, and J. Carlos. Remote analysis and measurement of libre software systems by means of the cvsanaly tool. In *Proceedings of the 2nd ICSE Workshop on Remote Analysis and Measurement of Software Systems (RAMSS)*, pages 51–55, 2004.

[84] I. Samoladas, L. Angelis, and I. Stamelos. Survival analysis on the duration of open source projects. *Infomation Software Technology*, 52(9):902–922, Sept. 2010.

[85] C. Santos, G. Kuk, F. Kon, and J. Pearson. The attraction of contributors in free and open source software projects. *J. Strateg. Inf. Syst.*, 22(1):26–45, Mar. 2013.

[86] 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. In *Proceedings of Hawaii International Conference on System Science (HICSS)*, pages 3446–3455, Jan 2012.

[87] A. Senyard and M. Michlmayr. How to have a successful free software project. In *Proceeding of the 11th Asia-Pacific Software Engineering Conference*, pages 84–91, 2004.

[88] P. N. Sharma, J. Hulland, and S. Daniel. *Examining Turnover in Open Source Software Projects Using Logistic Hierarchical Linear Modeling Approach*, pages 331–337. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[89] B. Shibuya and T. Tamai. Understanding the process of participating in open source communities. In *Proc. Int'l Workshop on Emerging Trends in Free/Libre/Open Source Software Research and Development (FLOSS)*, pages 1–6, 2009.

[90] I. Steinmacher, T. Conte, M. A. Gerosa, and D. Redmiles. Social barriers faced by newcomers placing their first contribution in open source software projects. In *Proc. Conf. on Computer Supported Cooperative Work and Social Computing (CSCW)*, pages 1379–1392, 2015.

[91] I. Steinmacher, I. Wiese, A. Chaves, and M. Gerosa. Why do newcomers abandon open source software projects? In *Proceedings of International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE)*, pages 25–32, May 2013.

[92] K. Stewart, T. Ammeter, K. J. Stewart, and T. Ammeter. An exploratory study of factors influencing the level of vitality and popularity of open source projects. In

Proceedings of the Twenty-Third International Conference on Information Systems, pages 14–17, 2002.

[93] M. Torchiano, F. Ricca, and A. Marchetto. Is my project's truck factor low?: Theoretical and empirical considerations about the truck factor threshold. In *Proceedings of the 2Nd International Workshop on Emerging Trends in Software Metrics (WETSoM)*, pages 12–18, 2011.

[94] D. W. van Liere. How shallow is a bug? why open source communities shorten the repair time of software defects. *ICIS 2009 Proceedings*, page 195, 2009.

[95] B. Vasilescu, K. Blincoe, Q. Xuan, C. Casalnuovo, D. Damian, P. Devanbu, and V. Filkov. The sky is not the limit: Multitasking across github projects. In *Proceedings of the 38th International Conference on Software Engineering*, ICSE '16, pages 994–1005, New York, NY, USA, 2016. ACM.

[96] B. Vasilescu, A. Serebrenik, M. Goeminne, and T. Mens. On the variation and specialisation of workload – a case study of the gnome ecosystem community. *Empirical Software Engineering*, 19(4):955–1008, 2014.

[97] E. Ververs, R. van Bommel, and S. Jansen. Influences on developer participation in the debian software ecosystem. In *Proceedings of the International Conference on Management of Emergent Digital EcoSystems*, MEDES '11, pages 89–93, New York, NY, USA, 2011. ACM.

[98] G. von Krogh, S. Spaeth, and K. R. Lakhani. Community, joining, and specialization in open source software innovation: a case study. *Research Policy*, 32(7):1217–1241, 2003.

[99] T. Wasserman and A. Das. Using flossmole data in determining business readiness ratings. In *Proc. of the Workshop on Public Data about Software Development (WoPDaSD)*, 2007.

[100] D. Weiss. Measuring success of open source projects using web search engines. In *Proceedings of International Conference on Open Source Systems*, pages 93–99, 2005.

[101] J. D. Werbel and D. J. Johnson. The use of persongroup fit for employment selection: A missing link in personenvironment fit. *Human Resource Management*, 40(3):227–240, 2001.

[102] J. West and S. O'mahony. The Role of Participation Architecture in Growing Sponsored Open Source Communities. *Industry and Innovation*, 15(2):145–168, 2008.

[103] J. Wu, K. Goh, and Q. C. Tang. Investigating success of open source software projects: A social network perspective. In *Proceedings of the International Conference on Information Systems, ICIS 2007, Montreal, Quebec, Canada, December 9-12, 2007*, page 105, 2007.

[104] J. Wu and Q. C. Tang. Analysis of survival of open source projects: a social network perspective. In *Pacific Asia Conference on Information Systems, PACIS 2007, Auckland, New Zealand, July 4-6, 2007*, page 19, 2007.

[105] K. Yamashita, Y. Kamei, S. McIntosh, A. E. Hassan, and N. Ubayashi. Magnet or sticky? measuring project characteristics from the perspective of developer attraction and retention. *Journal of Information Processing*, 24(2):339–348, 2016.

[106] K. Yamashita, S. McIntosh, Y. Kamei, A. E. Hassan, and N. Ubayashi. Revisiting the applicability of the pareto principle to core development teams in open source software projects. In *Proceedings of International Workshop on Principles of Software Evolution (IWPSE)*, pages 46–55, 2015.

[107] K. Yamashita, S. McIntosh, Y. Kamei, and N. Ubayashi. Magnet or sticky? an oss project-by-project typology. In *Proceedings of Working Conference on Mining Software Repositories (MSR)*, pages 344–347, 2014.

[108] Y. Yamauchi, M. Yokozawa, T. Shinohara, and T. Ishida. Collaboration with lean media: How open-source software succeeds. In *Proceedings of the 2000 ACM Conference on Computer Supported Cooperative Work*, CSCW '00, pages 329–338, 2000.

[109] Y. Ye and K. Kishida. Toward an understanding of the motivation open source software developers. In *Proceedings of International Conference on Software Engineering (ICSE)*, pages 419–429, 2003.

[110] N. Zazworka, K. Stapel, E. Knauss, F. Shull, V. R. Basili, and K. Schneider. Are developers complying with the process: An xp study. In *Proc. Int'l Symposium on Empirical Software Engineering and Measurement (ESEM)*, pages 14:1–14:10, 2010.

[111] M. Zhou and A. Mockus. What make long term contributors: Willingness and opportunity in OSS community. In *Proceedings of International Conference on Software Engineering (ICSE)*, pages 518–528, 2012.