UNIVERSITY OF CALIFORNIA,
IRVINE

Role Classification and Transition of OSS Developers

THESIS

submitted in partial satisfaction of the requirements
for the degree of

MASTER OF SCIENCE

in Software Engineering

by

Yunlong Song

Thesis Committee:
Professor David Redmiles, Chair
Associate Professor Hadar Ziv
Assistant Professor Iftekhar Ahmed

2021

TABLE OF CONTENTS

Page

LIST OF FIGURES **iv**

LIST OF TABLES **v**

ACKNOWLEDGMENTS **vi**

ABSTRACT OF THE THESIS **vii**

1 Introduction **1**

2 Research method **4**
2.1 Research questions 4
2.2 Literature search strategy 5

3 Results **10**
3.1 Overview 10
3.2 Role classification models 12
3.2.1 The onion model 15
3.2.2 Participative system 27
3.2.3 Modeling project organizations 27
3.2.4 Development maturity and specialized roles 28
3.2.5 Active contributors and supporting contributors 30
3.2.6 Project centric and community centric roles 30
3.3 Role identification methods 33
3.3.1 Membership based 34
3.3.2 Count based 40
3.3.3 Network based 40
3.3.4 Turnover based 41
3.3.5 Event based 48
3.3.6 Other methods 52
3.4 Role transition patterns 55
3.4.1 The onion model 55
3.4.2 Participative system 57
3.4.3 Within project organizations 57
3.4.4 Development maturity roles 59

3.4.5 Retiring path 59
3.4.6 Project-centric and community-centric roles 59

4 Discussion **62**
4.1 Disclose the method used for role identification 62
4.2 GitHub projects as study subject 63
4.3 The overlap and conflict of role identification 63

5 Conclusion **65**

Bibliography **67**

LIST OF FIGURES

Page

2.1 Query Performed for Searching Literature 8
2.2 Literature selection process 9

3.1 Temporal view of publications 13
3.2 Citation distribution of paper repository 13
3.3 Number of projects selected by each literature 14
3.4 Projects selected by multiple literature 14
3.5 The onion model. Adapted from "Evolution patterns of open-source software systems and communities." by Kumiyo Nakakoji, Yasuhiro Yamamoto, Yoshiyuki Nishinaka, Kouichi Kishida, and Yunwen Ye, 2002, In Proceedings of the International Workshop on Principles of Software Evolution (IWPSE '02). 16
3.6 A framework to describe participative system . Adapted from "Understanding the nature of collaboration in open-source software development," by C. Jensen and W. Scacchi, 2005, 12th Asia-Pacific Software Engineering Conference (APSEC'05), pp. 8 pp.- 28
3.7 The quality assurance paths. Adapted from "Role Migration and Advancement Processes in OSSD Projects: A Comparative Case Study," by C. Jensen and W. Scacchi, 2007, 29th International Conference on Software Engineering (ICSE'07), pp. 364-374 57
3.8 The development paths. Adapted from "Role Migration and Advancement Processes in OSSD Projects: A Comparative Case Study," by C. Jensen and W. Scacchi, 2007, 29th International Conference on Software Engineering (ICSE'07), pp. 364-374 58
3.9 Overview of career pathways reported by interviewees. Adapted from "Hidden Figures: Roles and Pathways of Successful OSS Contributors." by Bianca Trinkenreich, Mariam Guizani, Igor Wiese, Anita Sarma, and Igor Steinmacher, 2020, Proc. ACM Hum.-Comput. Interact. 4, CSCW2, Article 180 (October 2020), 22 pages. 60

LIST OF TABLES

Page

2.1 Keywords for Constructing the Searching Query 9
2.2 Searching Result of Conference Publications 9

3.1 Primary studies 15
3.2 Definitions of roles in the study of Nakakoji et al. 17
3.3 Definitions of roles in the study of Herraiz et al. 17
3.4 Definitions of roles in the study of Shibuya and Tamai 18
3.5 Definitions of roles in the study of Ko and Chilana 19
3.6 Definitions of roles in the study of Oezbek et al. 19
3.7 Definitions of roles in the study of Sinha et al. 21
3.8 Definitions of roles in the study of Lee and Carver 23
3.9 Definitions of roles in the study of Cheng et al. 24
3.10 Definitions of roles in the study of Nakakoji et al. 27
3.11 Definitions of roles in the study of Wagstrom et al. 31
3.12 Comparison of definitions of rockstars 32
3.13 Definitions of roles in the study of Cheng and Guo 32
3.14 Definitions of roles in the study of Trinkenreich et al. 34
3.15 Literature relevant to membership based role identification methods 35
3.16 Roles, identification criteria, and data source used in membership based methods 35
3.17 Literature relevant to count based role identification methods 40
3.18 Identification criteria, and data source used in count based methods 41
3.19 Literature relevant to network based role identification methods 41
3.20 Identification criteria, and data source used in network based methods 42
3.21 Literature relevant to onboarding based role classification 43
3.22 Definitions of roles in the study of Foucault et al. 47
3.23 Literature relevant to review based role identification methods 49
3.24 Identification criteria, and data source used in review based methods 49
3.25 Definitions of roles in the study of Terry et al. 53
3.26 Literature studied role classification models 61

ACKNOWLEDGMENTS

I would like to thank professor David Redmiles who provide insightful advice and authoritative guidance, and Zhendong Wang who offered sincere and generous help like a friend.

I would also like to thank my family and roommates who supported me all the time, their emotional support helped me to stay strong in this pandemic period.

ABSTRACT OF THE THESIS

Role Classification and Transition of OSS Developers

By

Yunlong Song

Master of Science in Software Engineering

University of California, Irvine, 2021

Professor David Redmiles, Chair

It has been acknowledged that many open source software (OSS) projects are successful even when compared with their commercial counterparts. Previous research has investigated the artifacts developed by OSS projects and the communities behind those OSS projects. The different roles that developers play have been studied to provide insights on developers' behavior and their impact on the OSS projects. By presenting the composition of developers in a project and a community, these studies can provide guidance for the development of a successful OSS project and a flourishing community. As for the individual developers, they can also benefit from obtaining awareness of their roles and the possible role evolution paths. By reviewing the existing role classifications and transition patterns, I aim to provide insights as a basis for construction of a systematic and universal role classification model and role transition patterns of OSS developers.

In this paper, I performed a systematic literature review of research on developers' roles and role transition in the OSS community. I identified six role classification models, five major role identification methods, and role transition patterns that have been studied. I also provide a series of discussions about methods used by these studies and the roles studied.

Chapter 1

Introduction

Open source software (OSS) has started to be well recognized as a successful software development model and has always been a vital part of software development over the past decades [32][43]. This area has also attracted more and more researchers, and their studies provide insights into all aspects of OSS projects. One research focus is the communities around OSS projects, which studies OSS projects from the human aspect. The communities around OSS projects are becoming increasingly heterogeneous, comprising not only developers but also designers, managers, and users with a wide-ranging level of expertise [11]. It would be hard to manage the development team without a clear view of the role composition. In addition, different from commercial projects, the composition of contributors are constantly changing, and failing to obtain the awareness of team structure could lead to projects' failure. According to the study of Coelho and Valente [12], there are nine reasons why open source projects fail and three of them are related to the team, they are lack of time, lack of interest, and conflicts among developers. These reasons could lead to the loss of contributors. By realizing the loss of contributors, especially these contributors' roles, it would be easier to find proper candidates and recruit them to make OSS projects sustainable. And if we want to obtain a precise extraction of contributors' roles within a

project, role classification models are essential.

The role of a developer is not stationary. The community would evolve as the development of software, and the evolution of the community results from the role changes of its members [32]. The onion model depicts an role transition pattern based on the observation that the project members would gravitate towards central roles over time [21]. This model can be used to describe the role transition of developers within a single project. However, it was later found to be limited [22] [34]. Other researchers have also proposed different role transition patterns based on their role definition and observation of various projects [11, 21, 31, 48, 52].

Through informal search, I found that there is still a lack of general understanding on role classification models and transition patterns from a systematic review perspective. In recent studies such as the work of Cheng and Guo[11] and the work of Trinkenreich et al.[48], they reviewed the related work but not with a systematic method. Therefore, it is necessary to perform a systematic review that aggregates the information regarding the role classification and transition that is currently dispersed across various studies.

The objective of this systematic review is to identify the role classification models and role transition patterns. I conducted the literature searching through a seven-step process, including querying digital libraries, snowball sampling and manual addition based on experts' recommendations. In this process, I identified 55 primary studies. These studies are selected based on the research topic and publication venue, I only include studies that investigate the roles of OSS developers and published in key conferences such as ICSE and FSE. After identifying the primary studies, I extracted the role studies in these papers, including role classification models and role transition patterns. I also compared the differences and similarities of these classification models. For the roles whose corresponding role classification is not specified, I analyzed their definition and identification criteria to find possible models they can be integrated into.

The major results of this study include six role classification models, five major role identification methods, and role transition patterns that have been proposed by ten papers. I categorize the role identification methods from the primary studies into five types: membership based, count based, network based, turnover based, and event based.

The contributions of this paper include: 1. summarizing existing role classification models, role identification methods and role transition patterns, and 2. providing a quick reference for researchers interested in conducting further studies on OSS developers' roles. This research can assist OSS communities and researchers in achieving a better understanding of OSS developers' roles and design strategies for projects and contributors.

The remainder of this paper is structured as follows. Section 2 gives an introduction of the background on OSS developers' roles. My research questions and the literature search strategy I used for this paper are presented in Section 3. In Section 4, I present the result of my study which includes my findings. In Section 5, I provide a series of discussions about methods used by these studies and the roles studied. I conclude my work in Section 6.

Chapter 2

Research method

2.1 Research questions

Developers play different roles in the development of open source software, and they can transit from one role to another. Both OSS developers and OSS projects can benefit from identifying developers' roles and role transition paths. Developers can recognize their position in the project and possible ways to transit to another role, while projects can stay sustainable by acknowledging the role composition and role transition of its contributors. I expect that by summarizing the role classification models and role transition patterns studied, this study can contribute to the construction of a comprehensive role classification model with corresponding role transition patterns. Therefore, I propose the following two research questions.

RQ1: What are the basic roles of developers in the development of open source software?

By answering the first research question, I aim to find the roles developers play in the development of OSS and also their roles in the OSS community.

RQ2: How do OSS developers transit or migrate between various roles in the community?

By answering the second research question, I intend to investigate the different patterns of role transition in one project or migration across multiple projects.

Through answering these two research questions, I expect to summarize the role classification models and role transitions of developers in the OSS community, so that this study could assist OSS developers to achieve awareness of their roles and find methods to obtain the roles they desire. Furthermore, tool designers can implement tools for project managers or communities to observe the role composition and transition of projects and communities, thus facilitate the projects' or communities' prosperity.

2.2 Literature search strategy

Our literature search approach is based upon the process established by Kitchenham et al. [24]. Before forming the query to construct the corpus for the research, I first obtain a small set of papers based on expert recommendation. This set of papers are used to as the initial set for initial information search. After I obtained a basic idea of the number of literature on my topic and the terms that could be used for query, I finally formed the proper query (Figure 2.1). After acquiring the corpus given by this query, I filter papers based on the venue type to include only papers published in key conferences. I analyzed the titles and abstracts of these papers to filter out those which are not relevant to my research. After that, I further analyzed the introduction and conclusion of each paper. Then, I performed data extraction on these papers. In addition, I also included several related studies manually. Finally, I perform a full literature reading for the papers in the corpus.

The literature discovery process can be organized as follow:

1. Initial informal search. I first obtained a set of papers recommended by experts, then performed an informal search based on these papers to validate the necessity of systematic literature review on this topic. I had 6 papers recommended by experts as initial set, then I read these papers and iteratively went through the references to find the relevant literature. This step also provides me the knowledge to construct a proper query for the next step.

2. Searching digital library. In this step, I formed the proper query to construct the corpus for this paper (see Figure 2.1), I performed this query in the ACM digital library. This query is connected by binary operators. The paper repository is managed by using a free and open source reference management tool, Zotero.

3. Title and abstract filtering. Although the search engine show only the papers whose abstracts contain the term specified in the query, some of these papers could still be not highly relevant to my focus. So, I performed an abstract analysis by reading the abstract of these papers and decided whether to include or exclude papers. This can reduce the time cost of reading full content of papers.

4. Introduction and conclusion analysis. I read the introduction and conclusion of the papers which are included after previous steps, papers will also be excluded if they are found to be irrelevant in this step.

5. Data extraction. I extracted the standard information and primary study specific data [24] for each paper. The standard information included title, authors, year of publication, publication type and number of citations. The primary study specific data extracted included studied roles, role scope, research method, results summary, future work and limitation.

6. Manual addition. I also included several studies that have solid contributions to this area, they were found in step 1 but not found in the query result since they are published

in small conferences, or indirectly related to my research focus. 10 papers were added manually, 7 of them are not published in the key conferences we selected. The rest papers each study a specific role but not referring to term "role" or "character". Data extraction is also performed on the papers I found in this step.

7. Citation search. I performed one round of forward snowball sampling based on the set of papers I have. I used the same analysis to filter out the irrelevant papers, including venue type filtering, title and abstract filtering, and introduction and conclusion analysis. Data extraction is also performed on the papers I found in this step.

The terms I used in the query are based on the research questions and papers I read in the initial informal search, there are three sets of keywords (Table 2.1). The first set of keywords (Domain 1) indicates that the underlying research area is Open Source Software, it contains Open Source Software and related abbreviations, OSS and FLOSS. The other sets of keywords (Domain 2 and Domain 3) indicate the subjects I are concerned of are roles of developers in the community, and each domain also contains some synonyms.

Papers are excluded when performing title and abstract filtering, introduction and conclusion analysis, and full literature reading. The exclusion criterion is that the papers not related to my subject, open source software developers' roles, will be excluded. The context of occurrence of keywords is restricted to be abstract, since if the keywords do not appear in the abstract, this paper is highly unlikely to be relevant. Keywords "oss" and "open source" are enclosed within quotation marks to search for exact matches, so that the search engine will not match terms like "possible" with "oss", or return papers with only "open" or "source" instead of "open source" as a whole.

Based on the research questions, I decide to restrict my research area within Software Engineering, Human Factors in Computing Systems, and Computer Supported Collaborative Work. The reason is that this paper focuses on the roles OSS developers play in software

{
(“Abstract”:“oss” OR “Abstract”:“open source” OR “Abstract”:floss)
AND
(“Abstract”:team OR “Abstract”:organization OR “Abstract”:community)
AND
(“Abstract”:role OR “Abstract”:character)
}

Figure 2.1: Query Performed for Searching Literature

development, while Software Engineering includes the papers study the process of software development, Human Factors include papers that study human factors, developers, in software development, and Computer Supported Collaborative Work include papers provide assistance tools for developers based on their roles. With that criterion, the conferences I decided to include are: ICSE, FSE, ESEM, CHI and CSCW. Figure 2.2 shows the distribution of papers' corresponding conferences.

After forming the query and determining publication venues, I choose ACM library to conduct the search. The reason is that it provides features to support my query, and provide paper access to the publication venues I choose.

Given the paper repository, I read the full content of the literature and extracted the necessary data. As previously mentioned, I extracted the standard information and primary study specific data for each paper. The standard information were collected by using Zotero and manually from the digital library, while primary study specific data were collected manually from the content of papers. I first identify the roles mentioned in the literature. After that, I analyzed the scope of the roles, which is the projects or ecosystems that provide the data for these studies. Moreover, I extract the research methods used by these studies, including the type of data and analysis methods used. In addition, I summarized the results, future work and limitations of the literature.

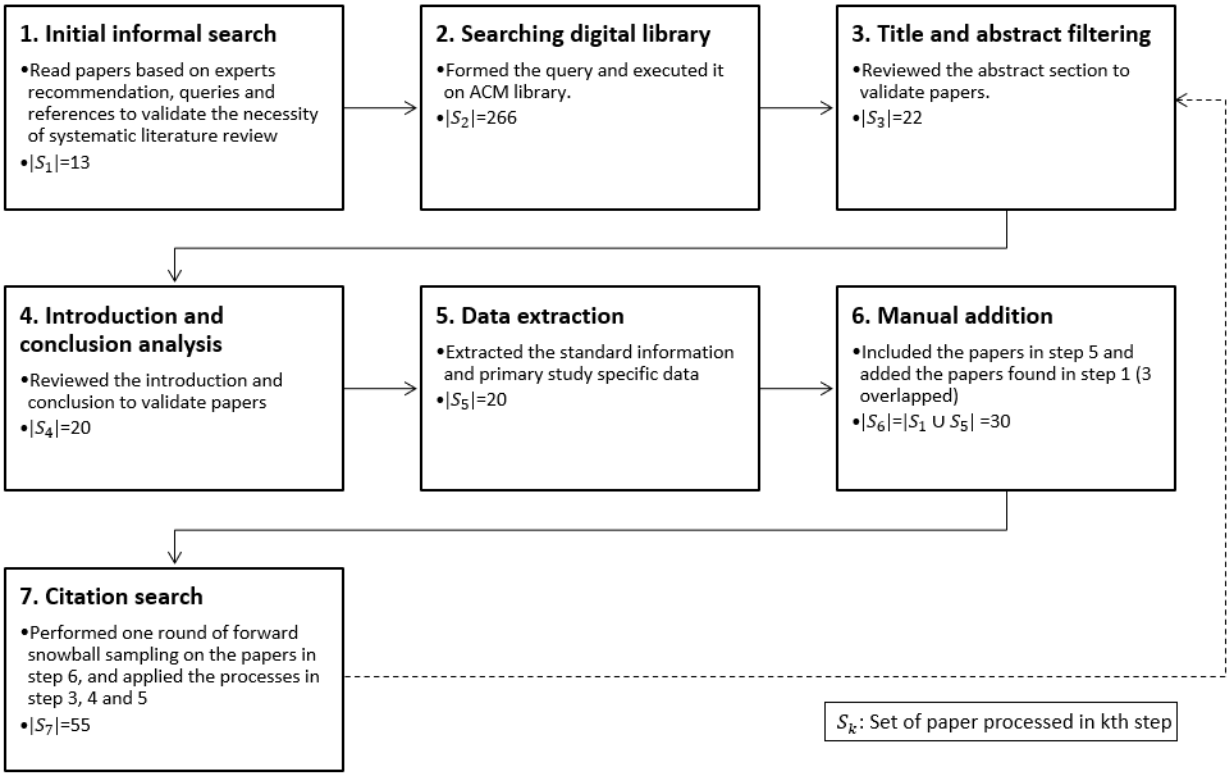


Figure 2.2: Literature selection process

Item	Keyword
Domain 1	OSS, Open Source, FLOSS
Domain 2	Team, organization, community
Domain 3	Role, character

Table 2.1: Keywords for Constructing the Searching Query

Conference	Searching results
ICSE	61
ESEM	11
FSE	8
CHI	4
CSCW	4

Table 2.2: Searching Result of Conference Publications

Chapter 3

Results

In this chapter, I present the results of the literature review. To answer RQ1, I extract the roles studied by each paper, categorize role classification models based on 5 types of role identification criteria, and recognize 6 roles that have not been integrated into any model. To answer RQ2, I present the role transition patterns that have been proposed by 10 papers.

In the following sections, I first present the overview of the literature in our paper repository. Then, based on the extracted data, I categorize the role classification models used in these studies, and other roles that have not been integrated into any model. After that, I summarize the role transition patterns proposed. Finally, I present a series of discussions about the findings from the literature review.

3.1 Overview

I identified 55 literature as primary studies, and the selected studies are presented in Table 3.1.

With the data extracted, I can present an overview of the studies. First, I checked the time distribution of the publication, which is depicted in Figure 3.1. It can be observed that the study of roles starts in 2002 and only a few papers appear until 2009. And the number of studies is increasing gradually, and half of the papers were published after 2018, which implies that this area has gained more attention recently.

The citation count of the paper repository is shown in Figure 3.2. I can see that half of the papers have less than or equal to 10 citations.

Most of the literature are empirical studies, except one study that designed and implemented one system but has not performed evaluation. I found 43 research (78%) conducted case studies, 13 research (23%) conducted one-on-one interviews, 10 research (18%) conducted survey research, while 2 research (3%) conducted text analyses. These research could use multiple methods, 15 research used more than one research method, which are case studies with either one-on-one interviews or survey research. In terms of analysis method, I found 21 research (38%) used only quantitative analysis, 14 research (26%) used only qualitative analysis, while 20 research (36%) used both quantitative and qualitative analysis. It can be observed that there are fewer studies that use qualitative analysis methods, but there is no significant difference between the number of studies using quantitative analysis methods and those using qualitative analysis methods.

The number of projects studied varies greatly in different literature. Figure 3.3 shows the distribution of number of selected projects in the paper repository. Not all papers provided the number of projects they selected, and some of the papers collect data from individual developers instead of from the projects' perspective, these literature are not used in this figure. I can see that most research (31 out of 44) selected less than 50 projects as subjects, and the half of these research (16 out of 31) selected less than 5 projects. Considering the effort to conduct an in-depth study of one project, it is reasonable to choose a relatively small number of projects. There are also studies that selected more than 1,000 projects, and

one study that used more than 10,000 projects as subjects. The study that used more than 10,000 projects are conducted by Tsay et al., and they gathered information of all 659,501 closed pull requests and 95,270 related users [49]. The authors of this study also conducted another study which selected a sample of pull requests from the dataset of the former study, they did not disclose the number of projects related so that study is not included in this figure.

I also extracted the projects' name when provided in the research, and found 181 unique projects. Among those projects, I found 26 projects selected by more than one studies, as shown in Figure 3.4. Although GNOME is regarded as an ecosystem in some literature, I consider GNOME as project for studies that regard GNOME as a large project, the studies take GNOME as ecosystems are not counted for Figure 3.4. Among all 55 studies, I found 20 of them collected projects' data from GitHub. The possible reasons are the large available samples and convenient collection. Due to the large amount of projects hosted on GitHub, it is easier to find available and representative projects. GitHub also provides convenient API for researchers to collect all sorts of information related to projects or users, and there are sources to collected dumped data like GHTorrent.

3.2 Role classification models

Contributors can have various type of interactions with projects, such as using the product, making code contribution, reporting issues, training newcomers, and so on. Researchers have proposed different models to classify contributors in the community of an OSS project or ecosystem, such as the onion model. In this section, I present six different models that have been investigated by previous studies. There are twenty one studies either proposed or investigated a role classification model, these studies are shown in Table 3.26.

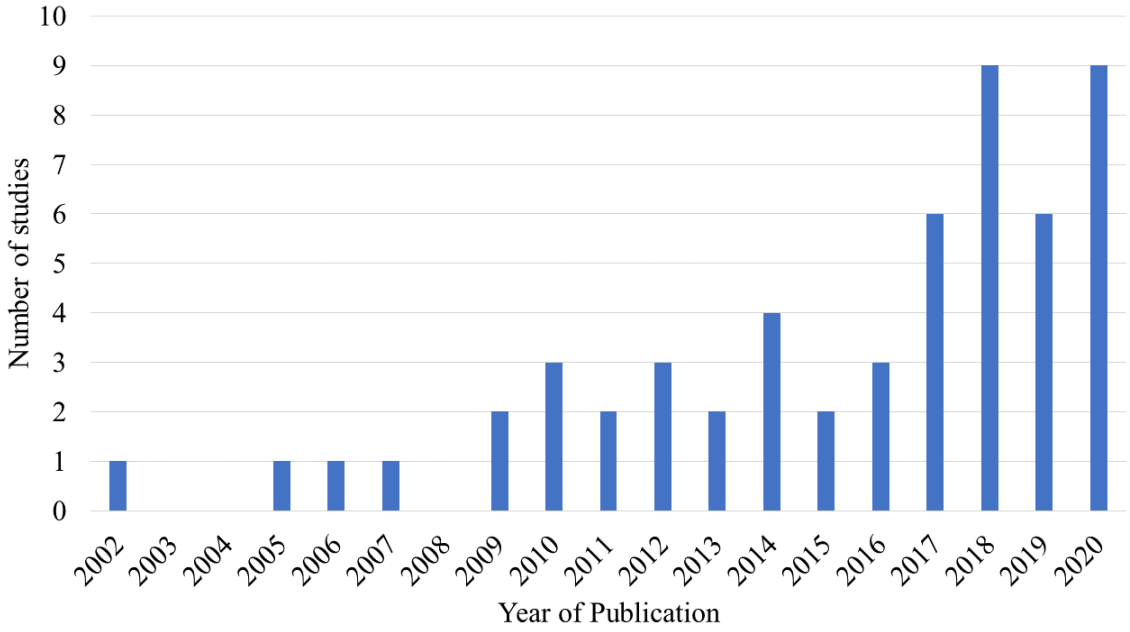


Figure 3.1: Temporal view of publications

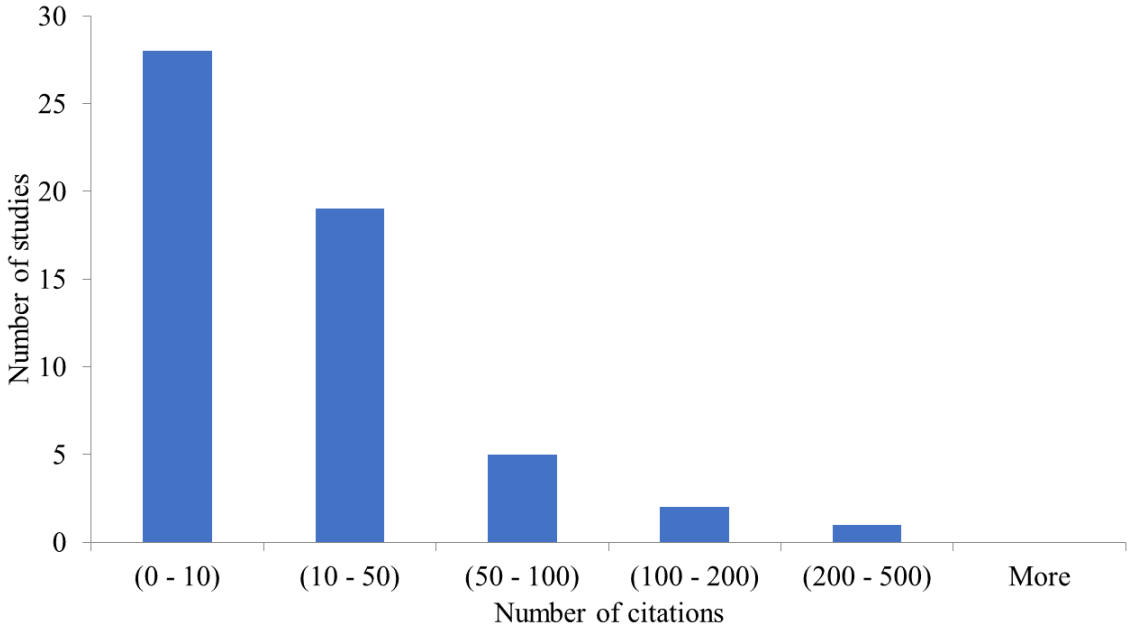


Figure 3.2: Citation distribution of paper repository

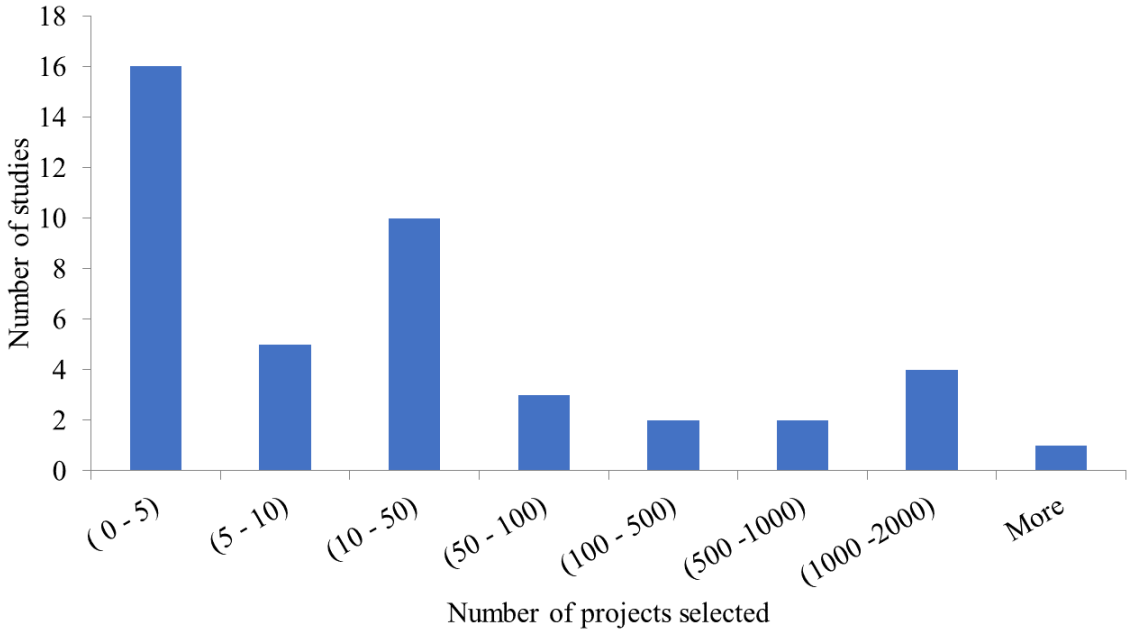


Figure 3.3: Number of projects selected by each literature

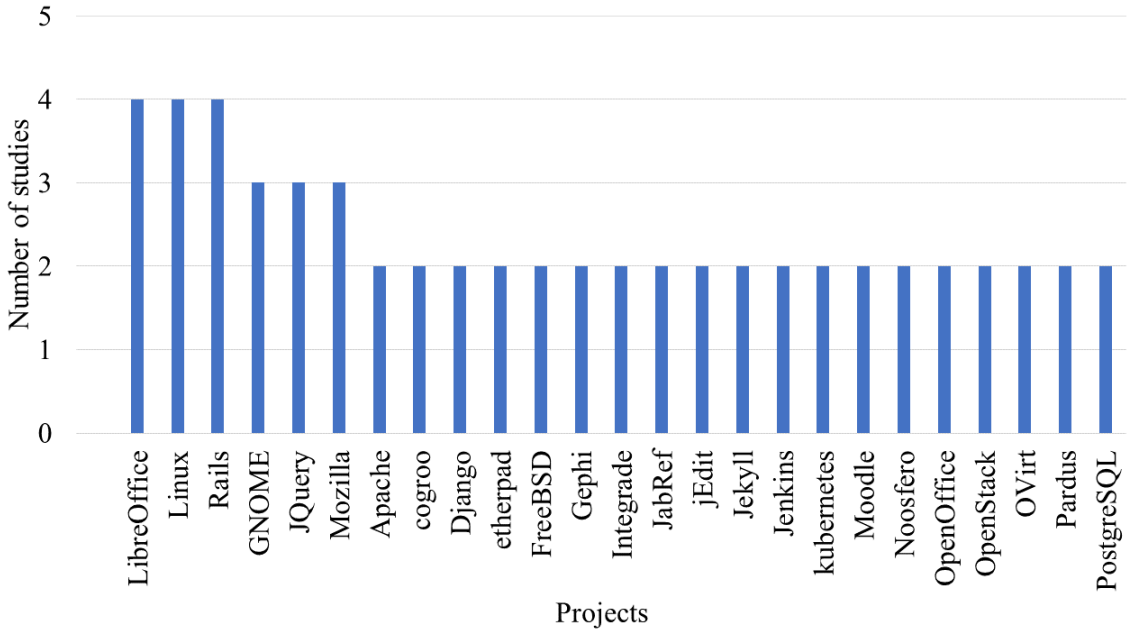


Figure 3.4: Projects selected by multiple literature

Identifier	Year	Reference	Identifier	Year	Reference
[PS1]	2002	Nakakoji et al. [32]	[PS29]	2017	Zhou et al. [59]
[PS2]	2005	Nakakoji et al. [31]	[PS30]	2017	Cheng et al. [10]
[PS3]	2006	Herraiz et al. [20]	[PS31]	2017	Coelho and Valente [12]
[PS4]	2007	Jensen and Scacchi [21]	[PS32]	2018	Coelho et al. [13]
[PS5]	2009	Nurolahzade et al. [33]	[PS33]	2018	Bayati [4]
[PS6]	2009	Shibuya and Tamai [36]	[PS34]	2018	Middleton et al. [29]
[PS7]	2010	Oezbek et al. [34]	[PS35]	2018	Calefato et al. [7]
[PS8]	2010	Terry et al. [47]	[PS36]	2018	Balali et al. [3]
[PS9]	2010	Ko and Chilana [25]	[PS37]	2018	Dias et al. [15]
[PS10]	2011	Sinha et al. [38]	[PS38]	2018	Valiev et al. [51]
[PS11]	2011	Jergensen et al. [22]	[PS39]	2018	German et al. [18]
[PS12]	2012	Dabbish et al. [14]	[PS40]	2018	Steinmacher et al. [42]
[PS13]	2012	Wagstrom et al. [52]	[PS41]	2019	Cheng and Guo [11]
[PS14]	2012	Canfora et al. [9]	[PS42]	2019	Zhou et al. [60]
[PS15]	2013	McDonald and Goggins [28]	[PS43]	2019	Tan [44]
[PS16]	2013	Lee et al. [27]	[PS44]	2019	Müller [30]
[PS17]	2014	Fagerholm et al. [16]	[PS45]	2019	Steinmacher et al. [41]
[PS18]	2014	Tsay et al. [49]	[PS46]	2019	Avelino et al. [1]
[PS19]	2014	Bosu and Carver [5]	[PS47]	2020	Wang et al. [54]
[PS20]	2014	Tsay et al. [50]	[PS48]	2020	Subramanian [43]
[PS21]	2015	Steinmacher et al. [39]	[PS49]	2020	Canedo et al. [8]
[PS22]	2015	Foucault et al. [17]	[PS50]	2020	Trinkenreich et al. [48]
[PS23]	2016	Steinmacher et al. [40]	[PS51]	2020	Tan et al. [45]
[PS24]	2016	Zhu et al. [61]	[PS52]	2020	Wang [53]
[PS25]	2016	Sarma et al. [35]	[PS53]	2020	Silva et al. [37]
[PS26]	2017	Joblin et al. [23]	[PS54]	2020	Wessel [55]
[PS27]	2017	Lee and Carver [26]	[PS55]	2020	Tan et al. [46]
[PS28]	2017	Hata et al. [19]			

Table 3.1: Primary studies

3.2.1 The onion model

To understand the "natural production evolution" of OSS development, Nakakoji et al. conducted a case study in 2002 [32]. In this study, they proposed a model to describe the general structure of OSS community, which looks like a onion with eight layers (See Figure 3.5). Each layer represent a role, and they argue that the role of contributors will evolve from outside to the inner circle. There are eight roles, in the order from outside to inside layer, they are passive user, reader, bug reporter, bug fixer, peripheral developer, active developer,

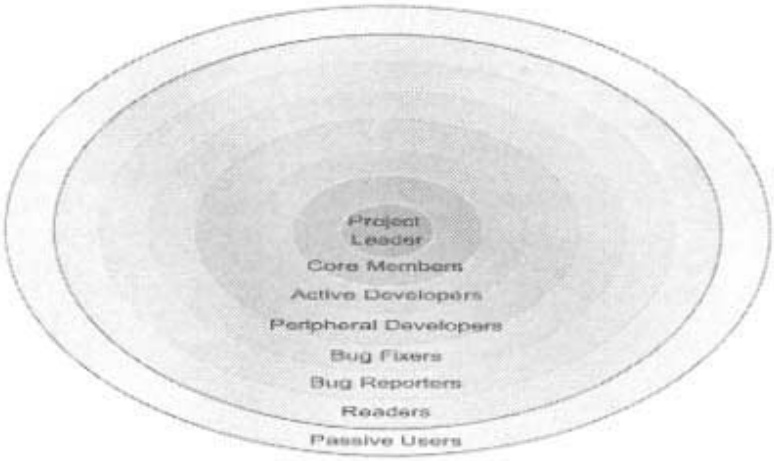


Figure 3.5: The onion model. Adapted from "Evolution patterns of open-source software systems and communities." by Kumiyo Nakakoji, Yasuhiro Yamamoto, Yoshiyuki Nishinaka, Kouichi Kishida, and Yunwen Ye, 2002, In Proceedings of the International Workshop on Principles of Software Evolution (IWPSE '02).

core member, and project leader. The definitions of these roles are presented in Table 3.2. They did not specify the method they used to construct this general framework.

Nakakoji et al. collected data through survey, mailing list archives, and quantitative data when necessary. They did not provide a systematic method that can be used to identify these roles in other OSS projects, such as offering some factors or metrics. However, this "onion model" makes a solid contribution to the role classification method in OSS projects, and provide a basis for future researchers to discover proper factors that can be used to identify different roles.

Herraiz et al. conducted a study to investigate the difference of joining process between volunteers and hired developers [20]. In this study, they used the onion model provided by Ye et al. [58], which contains a more theoretical identification and description of the roles. However, not all roles are studied, they collected metrics like 1st message, 1st report, 1st fix, 1st commit to identify the role transition of developers, but ignored roles like passive users whose activity cannot be traced. The roles mentioned in this study are shown in Table 3.3. However, the roles with clear identification methods only include advanced user, bug

Roles	Definition
Passive user	Users who just use the system in the same way as most of us use commercial software
Reader	Active users of the system
Bug reporter	Contributors who discover and report bugs
Bug fixer	Contributors who fix the bug that is either discovered by themselves or reported by bug reporters
Peripheral developer	Contributors who contribute occasionally new functionality or features to the existing system
Active developer	Contributors who regularly contribute new features and fix bugs
Core member	Contributors who are responsible for guiding and coordinating the development of an OSS project
Project leader	Often the person who has initiated the project

Table 3.2: Definitions of roles in the study of Nakakoji et al.

reporter, bug fixer, and committer. Herraiz et al. mined CVS, mailing list and Bugzilla to collect statistics, then they identified roles based on these statistics. Herraiz et al. found that there is not a clear "common" joining pattern, and its behavior does not seem to comply with the predictions of the onion model. Also, onion model is followed only by the volunteer developers but not for hired developers.

Roles	Definition
Plain user	occasionally visit the project's web site, and maybe reads mailing lists.
Advanced user	occasionally send some messages to mailing lists.
Bug reporter	report bugs
Bug fixer	fix bugs
Core developer	collaborates in the project with a high level activity
Committer	Contributors who have write access to code repository of the project through VCS

Table 3.3: Definitions of roles in the study of Herraiz et al.

In 2009, Shibuya and Tamai conducted an study to investigate the process of participation in OSS communities. In this study, they only studied a subset of roles from the onion model. The roles they studied are bug reporters and committers. Bug reporters are developers who submit bug reports, while committers are developers who have write-access to the code

repository. The identification criteria for these roles are presented in Table 3.4. Shibuya and Tamai extracted qualitative (publicly available documents) and quantitative (bug tracking system and revision control systems) data of three OSS projects and conducted an analysis. They found that the number of active developers does not change significantly when the total number of committers increases for the selected OSS projects and most part of the development is executed by core members. They also found difficulties faced by newcomers, such as selection of a suitable task, and they identified patterns of contributor activities. Moreover, they presented two types of paths to become a member with write access, which will be further discussed in Section 3.4.

Roles	Definition
Active developers	Contributors who submit at least one contribution in a specified month
Bug reporters	Contributors who submit bug reports
Committers	Contributors who have write-access to code repository

Table 3.4: Definitions of roles in the study of Shibuya and Tamai

In 2010, Ko and Chilana investigated Mozilla contributors to investigate power users' activities [25]. In this study, they separated contributors based on onion model. The contributors are separated into four categories: core developers, active developers, reporters, and users, the definitions of these roles are presented in Table 3.5. Ko and Chilana collected data mainly by analyzing Mozilla Bugzilla bug report repository. Through the analysis, they found that the primary value of open bug reporting is in recruiting talented reporters, and not in deriving value from the masses. One limitation of this study is that the classification of contributors was static, so contributors may be placed in the wrong group. Thus, they suggest a future direction that understanding contributions over time.

After the onion model has been proposed, researchers have carried out experiments to verify if the onion model is correct under different settings. Oezbek et al. conducted a social network analysis to check if the onion model is valid with respect to the participation in OSS project mailing-list traffic [34]. They collected data from the mailing list archive to

Roles	Definition
Core developers	Release drivers, super reviewers, module owners, peers
Active developers	Contributors who had been assigned any reports
Reporter	Contributors who had reported at least one bug
User	The remaining contributors

Table 3.5: Definitions of roles in the study of Ko and Chilana

construct the network, and the roles are classified based on network properties possessed by different nodes. Three roles are involved in this study, they are core developer, co-developer, and periphery. The definitions of roles are presented in Table 3.6.

Based on their visual observation and quantitative analysis of the network, they claimed that the onion model could be misleading as core appears to have qualitatively different roles as well. Although they found the core developers could be qualitatively different, they did not discover the precise identification criteria for these roles. Apart from that, the transition of individual mailing-list participants towards ever higher participation is qualitatively discontinuous, which also contradict the smooth transition from layer to layer suggested by the onion model.

Roles	Definition
Core	Developers who have sent at least one e-mail to the list in at least k calendar months, with k=8
Co-developer	Developers who strongly oriented to the core but also share some links between each other
Peripheral participants	Developers who either only connected to the project core or not at all

Table 3.6: Definitions of roles in the study of Oezbek et al.

Herraiz et al.'s work [20] found that the joining process does not seem to comply with the predictions of the onion model within single project, but whether this will hold true under the context of ecosystem is not studied. In 2011, Jergensen et al. conducted a study to examine whether the onion model of joining and progressing still holds true in large project ecosystems and how the model might change in such settings [22].

In this study, Jergensen et al. selected six projects from GNOME ecosystem as subjects. They mined mailing list archives, bug tracking system, and source code repositories to collect data. They used an approach similar to Herraiz et al., they identified developers' roles based on their first contribution to mailing lists, project bug tracker, and project source code. In that case, they studied a subset of onion model roles which are users, bug reporters, bug fixers, and committers.

Through analyzing the role transition of contributors, they found little support for the traditional onion model within a single project, but it is supported slightly more often when considering multiple projects. They also found that tenure in a project or the ecosystem did not have a high impact on the centrality when both number of contributions and the eigenvector centality are considered. They discovered four progression paths, these role transition patterns will be further illustrated in Section 3.4.

Sinha et al. studied the phenomenon of the induction of external developers as code committers in 2011 [38]. The roles studied in this paper include core committer, non-core committer, bug submitter, and bug committer. The identification criteria for these roles are presented in Table 3.7. The roles they studied are subset of the onion model, but with slightly different definitions. After mining data from code repositories and bug-tracking systems, they classify the committers based on the criteria and analyze their activities. They found that developers establish trust and credibility in a project by contributing to the project in a non-committer role, and employing the organization of a developer is another factor that influences trust.

In 2014, Bosu and Carver proposed an novel core identification method called Core Identification using K-means (CIK) [5]. CIK will classify developers into two groups, core and peripheral, based on 6 SNA centrality measures. They aim to identify how OSS developers' reputation affects the outcome of their code review requests, with the assumption that core developers have a better reputation.

Roles	Definition
Core committer	User who has committed code to the project code repository before the first release
Non-core committer	User who did not commit code to the project code repository before the first release
Bug submitter	User who has created a bug report in bug-tracking system, but has not committed code to the code repository
Bug committer	Person who has submitted a code patch in the bug-tracking system but who has not committed code

Table 3.7: Definitions of roles in the study of Sinha et al.

Different from Oezbek et al.'s work ,Bosu and Carver construct the social network with code review interactions instead of mailing list. The SNA centrality measures they used include betweeness, closeness, eigenvector, PageRank, and eccentricity. They use K-means clustering algorithm to combine these different measures into a new measure and detect the core developers.

With analysis regards the first feed back interval, review interval, acceptance rate, and number of patches per review request, Bosu and Carver found that core developers enjoy quicker first feed back intervals, shorter review intervals, and higher code acceptance rates. They did not observe any qualitative differences among core developers in this study, which implies the criteria for further classification of core developers may not be relevant to core developers' code review activity.

In 2017, Joblin et al. conducted a comparison of count-based operationalizations and network-based operationalizations. Count-based role classification relies on simple counts of individual developers' activities as mentioned in Section 3.2.2. In this paper, Joblin et al. collected these metrics from both the mailing list archive and version control system. They used these operationalizations to classify developers into core and peripherals based on the standard 80th percentile threshold, since it is widely used and it has been justified in other research.

They identified three count-based operationalizations and proposed five network-based operationalizations. The three count-based operationalizations are based on three metrics: commit count, lines of code (LOC) count, and mail count. As for the network-based operationalizations, they are degree centrality, eigenvector centrality, hierarchy, role stability, and core-peripheral block model. Each of these five operationalizations reflects one type of characteristics that should distinguish core and peripheral developers.

To evaluate the count-based operationalizations and network-based operationalizations, they also conducted a survey to collect developers' perception of roles. After the evaluation, they found that count-based operationalizations of developer roles are outperformed by network-based operationalizations. In addition, the network perspective can offer valuable insights regarding developer roles which are concealed by non-technical operationalizations.

One-Time Contributors (OTCs) are on the very fringe of the peripheral developers. To provide a better understand of OTCs, Lee and Carver investigated activities of core contributors, peripheral contributors, and OTCs. In this study, they classified contributors based on their code contribution, the roles, and their definitions are shown in Table 3.8. Lee and Carver mined projects' repositories for information such as code-review data, inline comments, patch data, request details, and contributors' data. Through analysis, they found that OTCs represent a distinct group of contributors compared to core and peripheral developers. The results confirmed that OTCs do face unique barriers and stronger barriers than other peripheral contributors do, and their patches are also different from other peripheral developers. It implies that peripheral developers can be further divided into more groups of contributors with qualitative differences, and the criteria for this classification could be related to the barriers they face and the patches they commit.

To make a project sustainable, it is necessary to attract and retain developers, especially project leaders and core developers. This is also true under the context of ecosystems. Cheng et al. conducted a case study on the GNOME ecosystem to reveal the factors that influence

Roles	Definition
OTC	Contributors who have one code patch marked with a status of 'MERGED'
Peripheral	Contributors who contribute less than 2% of the total code in the repository but not OTCs
Core	Contributors who contribute 2% or more of the total code in the repository

Table 3.8: Definitions of roles in the study of Lee and Carver

developers' chances to evolve into project leaders and core developers [10].

In this study, Cheng et al. used the widely accepted role classification model proposed by Xu et al. [57]. This model contains four roles: active users, project leaders, core developers, and co-developers. Cheng et al. decided to only consider project leaders, core developers, and co-developers, since their focus is on the evolution of the developers. However, they did not adopt all the calculation methods given by Xu et al., the reasons are that project leaders and core developers are not listed in GNOME dataset, and there are different ways to identify core developers. The definitions of these roles are presented in Table 3.9. Apparently, these two categories of core developers are identified using level of contribution based method similar to Lee and Carver [26] and network analysis based method similar to Oezbek et al. [34].

Cheng et al. used the development data of GNOME to identify developers' roles and extract indicators for developers' subjective willingness and project environment. After analysis, they were able to discover that different sets of indicators that influence different role transitions. Moreover, they suggested to investigate the evolution of developer roles in other larger OSS ecosystems, such as GitHub, to validate the generalizability of the results.

In 2018, Coelho et al. conducted a survey to reveal core developers' motivations for joining an open source project [13]. In this study, they identified core developers based on the number of commits. They first define the core team as developers who produce 80% of

Roles	Definition
Project leader	Initiator of a project.
Developmental core developer	Top developers whose cumulative contribution is just more than or equal to 80% work of the project.
Collaborative core developer	Developers in the center of the cooperation network of developers in the project.
Co-developer	Developers who have no significant contributions

Table 3.9: Definitions of roles in the study of Cheng et al.

the overall amount of commits in a project, then exclude developers who have less than 5% of total number of commits. After core developer identification, they surveyed the core developers and analyzed the response. They revealed the motivations, the most common project characteristics and practices that most helped engagement and barriers faced by for the core developer. They also compared their results with other studies focused on other roles, such as One-Time Code Contributors (OTC), casual contributors and newcomers.

In 2018, Calefato et al. aimed to study developers' personality in various contexts, for example, different roles [7]. They studied two roles, core developers and peripheral developers. Core-peripheral is a simplified version of the onion model, there have been studies using different criteria to distinguish core developer and peripheral developers [5] [23] [26]. In this study, core developers are the project members with commit access to the repositories, while peripheral developers are the commit authors without access to the repository. Calefato et al. identified the role of developers based on data extracted from project code repositories. Unlike GitHub, There is no visible role labels of participants regarding the subjects for this study, the Apache Software Foundation projects. Therefore, they collect commit metadata from Git repositories for information such as committer id to find the integrator who have commit access. The results show that developers' traits do not vary with their roles, membership, and extent of contribution. The roles refer to two groups of developers at the same time period, while membership refers to comparison of the same developers before and after obtaining membership. They also found that developers' personality evolves over time as

more conscientious, agreeable and neurotic, and the openness and agreeableness traits are antecedents of successfully becoming a project contributor.

In 2019, Müller conducted a study to build a better understanding of OSS communities, their dynamics over time, key players and dependencies on them [30]. The role involved in this study is the top contributor, which are the contributors who contribute the majority of the project. In this study, they use lines of code (LoC) to measure developers' contribution. This study is still on going when this paper is published. Based on the preliminary results, they confirmed the assumption that most code in open source projects is created by just a small number of active contributors. They realized the limitation of using LoC as measurement and mentioned other measurement methods as future improvements. They also presented other improvements such as evaluating communities by their lifetime, comparing active and inactive projects, and so on.

Avelino et al. conducted an empirical investigation of projects' abandonment and survival in 2019 [1]. They aimed to provide empirical evidence on the frequency of project abandonment and survival, the differences between abandoned and surviving projects, and the motivation and difficulties faced when assuming an abandoned project.

In this study, they studied the core developers but give the another name which is Truck Factor (TF) developers. TF is the minimal number of developers that the project depends on for its maintenance and evolution, and the departure of influential TF developers is TF developers detachment (TFDD). To identify the TF developers, they used the algorithm proposed by Avelino et al. [2]. The TF developers are computed based on the degree of authorship (DOA), and the developers with the highest DOA of at least 50% of the system's files. In this study, the metrics needed for TF calculation were extracted from GitHub projects' repositories. This algorithm is also used by Canedo et al. [8] to identify core developers.

Through the analysis of TFDD in the OSS projects, they found that TFDDs indeed happen in open source projects, and that projects can survive such conditions by attracting new core contributors. Moreover, the results of the survey revealed the motivation that led these developers to take over the studied projects after the project faced a TFDD, and revealed the principal enablers and barriers faced by these developers during this process.

By comparing the factors that attract new contributors in Coelho et al.'s work [12] with this paper's results about attracting new TF developers, they found that new TF developers are motivated by their own use and need to save projects in contrast to new, casual or one-time contributors. As future directions, they suggested the design of tools to assess the risks faced by an open source project, in case it is abandoned by its TF developers, and develop recommenders of TF developers for a system.

As the effect of gender diversity in OSS communities has gained increasing attention, Canedo et al. conducted a research to investigate the gender diversity and work practices of core developers contribution to OSS communities. In this study, they mined software repositories and identified the core developers using Truck Factor (TF) algorithm, then identified their gender and compare their contributions. Moreover, they conducted a survey with women core developers to understand their perceptions about gender bias.

To identify the core developers, they also used the same TF algorithm proposed by by Avelino et al. in 2016 [2]. This study is not classified as the same category as Avelino et al.'s work, since in this study they define core developers as "developers that significantly contribute to the development of a system", and this study's focus is not the maintenance of OSS projects. However, the core developers and maintainers are highly coincident, and I will elaborate on this in Section 4.1.

Through identifying the women core developers in OSS communities and conducting a survey, Canedo et al. found that there are more significant underrepresented women core developers

than women developers. However, there are no statistically significant gender-related differences in the work practices of core developers. And women core developers believe that gender diversity is important for OSS communities.

3.2.2 Participative system

To better understand the nature of collaboration in OSS development, Nakakoji et al. proposed a framework (See Figure 3.6) to describe a participative system [31]. A participative system refers to an organic socio-technical system "in which the social and technical infrastructures interconnecting users and artifacts" through supporting collaboration both in designing and using artifacts and in framing individual and collective goals. In this framework, Nakakoji et al. classified OSS projects participants into three types based on their type of engagement, the definitions of these roles are shown in Table 3.10. Although a precise definition is given, they did not present how they use this definition and project's data to identify different roles in their case study on GIMP. This case study also tested their hypothesis about the transition patterns based on this role classification, which will be illustrated in Section 3.4.

Roles	Definition
Passive participants	Only use objects of design hosted by a participative system
Active participants	Contribute to objects of design in a participative system
Power participants	Influence both the objects of design and the participative system itself

Table 3.10: Definitions of roles in the study of Nakakoji et al.

3.2.3 Modeling project organizations

et al.'s work [20] shows that the onion model is not working well when depicting the joining process, and Jensen and Scacchi pointed out that the onion model also fails to draw out

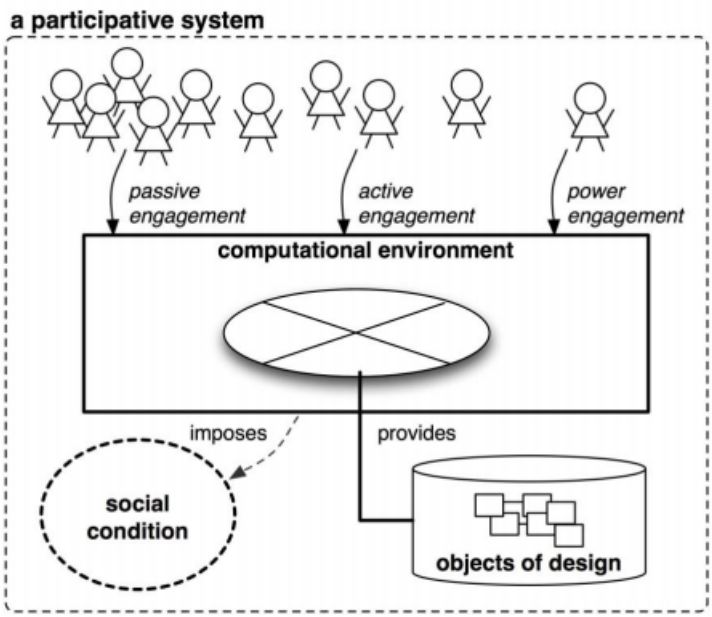


Figure 3.6: A framework to describe participative system . Adapted from "Understanding the nature of collaboration in open-source software development," by C. Jensen and W. Scacchi, 2005, 12th Asia-Pacific Software Engineering Conference (APSEC'05), pp. 8 pp.-

the presence of multiple tracks of project career advancement through different role-sets. Jensen and Scacchi conducted an empirical analysis of the role migration and project career advancement process through comparative case studied in 2007, to make the socio-technical process explicit. In this study, Jensen and Scacchi investigated three OSS project organizations (Mozilla.org, Apache.org, and NetBeans.org), and they present the role-sets, role migration and advancement process in these projects. The role-sets are identified based on participant interviews and web site mining. The detailed identification for each role is not presented, while the responsibility and how to obtain some of the roles are illustrated in the study. The role transition mentioned in this paper will be further illustrated in Section 3.4.

3.2.4 Development maturity and specialized roles

Another study on roles in ecosystem was conducted by Wagstrom et al [52]. in 2012. In this study, they examine a subset of communities found in GitHub and identify a variety of

roles. In this study, Wagstrom et al. collected metrics like watch, issues, PR, and marked as member from the project repositories, and retrieved relevant users' information like account information, the users they are following, their followers, and the repositories they own. Then based on this information, they identified a set of roles.

The roles they found can be divided into two classes: Development maturity and Specialized roles. Development maturity roles include lurkers, issues, independent, aspiring, external contributors, and internal collaborators; specialized roles include prodder, project steward, code warriors, nomad warriors, and project rockstars. The definitions of these roles are presentated in Table 3.11.

Besides identifying these roles, they also found that these roles often persist across sub-communities in the GitHub ecosystem. However, GitHub hosts more than 100 million projects, so a validation of this role classification on a larger scale may reveal other properties of these roles and discover other roles.

The concept of project rockstars is adopted from Dabbish et al.'s work in 2012, and this concept is also adopted by Lee et al. The comparison of rockstars in these three literature is shown in Table 3.12. Wagstrom et al. provided finer identification criteria compared with Dabbish et al.'s work, their criteria include the level of contribution based on the number of commits and range of influence based on the number of followers. In the study of Dabbish et al., they interviewed users of GitHub and identified a group of developers referred as "coding rockstars". In this study, they found 4 key features of visible feedback driving a rich set of inferences around commitment, work quality, community significance and personal relevance. In the study of Lee et al., they used GitHub's API to collect developers' actions include commit, follow as data. They sorted developers based on their number of followers and number of actions performed and identified four rockstars. They found that these rockstars' actions have a greater influence on their followers, and the type of actions affect their followers differently. Their influence on followers may depend on a project's age, and

their followers' activity will increase when they increase the activity on a project. Moreover, their followers use them as guide to projects to work on.

3.2.5 Active contributors and supporting contributors

In 2019, Cheng and Guo proposed another role classification by analyzing OSS contributors' activities. In this study, they first extracted six activities of contributors (knowledge sharing, code contribution, issue coordination, progress control, code tweaking, and issue reporting), then they identify two groups of contributors with hierarchical clustering analysis. These two groups are active contributors and supporting contributors, and they further analyzed the clusters of each role and found four clusters in the active contributors group and five clusters for supporting contributors. Clusters in the active contributors are named as intense code contributors, coordinators, core developers, and all-rounders. Clusters in supporting contributors are named as engaged issue reporters, occasional issue reporters, progress controllers, issue fixers, and rare contributors. The definitions of these roles are presented in Table 3.13. This role classification is based on the type of contribution as they use six activities which represent different types of contributions in hierarchical clustering analysis, so the contributors are merged into the same cluster when they are similar in the term of type of contribution. This study also includes an analysis of role dynamics and discussion about role change, the role transition mentioned in this paper will be further illustrated in Section 3.4.

3.2.6 Project centric and community centric roles

Previous studies mainly focus on the roles within a project, even when ecosystem is considered. However, as the open source community drastically matured, non project-centric roles emerge and become important to OSS communities. In 2020, Trinkenreich et al. conducted

Roles	Definition
Development maturity roles	These roles track the progress of an individual as they become socialized into the community to become full contributors
Specialized roles	These roles include roles that a contributor can take depending on their commitment and interest
Lurkers	Individuals who have only taken action to monitor a project or issues related to a project.
Issues	Individuals who have been active on the project issue tracker, either by filing new issues or commenting on existing issues or pull requests
Independent	Individuals who have forked the project source code repository using the GitHub infrastructure but have not issued any pull requests
Aspiring	Individuals who have created pull requests that have been closed but have never had their pull requests merged
External contributors	Individuals who have created pull requests that were later merged into the project source code, but are not official contributors to the project or members of the organization that own the project
Internal collaborators	Individuals who are marked as contributors to the project or are members of the organization that owns the project and have source code in the main project repository
Prodder	Individuals who identify and take on long standing issues or issues that have idle for a long time. They are top 20% individuals in terms of the number of issues they have prodded, subject to a floor than an individual must have been involved on at least 1% of the issues in a project
Project stewards	Individuals who primarily focus on managing the project. They are the top 20% of individuals working on a project both in terms of number of issues closed and number of pull requests closed
Code Warriors	Individuals who have frequent and consistent commits to a project. They are the top 20% of individuals working on a project in terms of both the frequency of their commits and also the standard deviation of the time between their commits
Nomad Coders	Individuals who have contributed only minor code changes and then have either move onto the next projects or individuals who are participating in one project, but make minor contributions to another project
Project Rockstars	Individuals who have a high visibility in their project and are significant contributors to their project. They are in the top 20% in terms of the number of commits to a project and the number of people in the project who follow them

Table 3.11: Definitions of roles in the study of Wagstrom et al.

Study	Definition
[PS12]	Developers with thousands of followers
[PS13]	Developers in the top 20% in terms of the number of commits to a project and the number of people in the project who follow them
[PS16]	Developers whose number of followers and actions were significantly different from other developers

Table 3.12: Comparison of definitions of rockstars

Roles	Definition
Active contributors	Major group with high dendrogram height
Supporting contributors	Major group with low dendrogram height
Intense code contributors	Contributors who develop a certain functionality of the software within a short time period.
Coordinators	Contributors who focus mainly on Knowledge Sharing, Issue Coordination, and Issue Reporting activities.
Core developer	Contributors who perform actively in Code Contribution, Code Tweaking, Progress Control, and Knowledge Sharing
All-rounders	Contributors who have medium level of contribution in all dimensions
Engaged issue reporters	Contributors who focus on Issue Reporting, report more issues
Occasional issue reporter	Contributors who focus on Issue Reporting, report fewer issues
Progress controllers	Contributors who mostly engage in the Progress Control activity
Issue fixers	Contributors who focus on making small tweaks to the code or fixing bugs.
Rare contributor	Contributors who only participate in a minuscule amount of activities

Table 3.13: Definitions of roles in the study of Cheng and Guo

a study to investigate the roles and activities that are part of the current OSS landscape and the different career pathways in OSS [48]. They found another set of roles named community-centric roles.

In order to investigate the "behind the scene" roles, Trinkenreich et al. recruited experienced and well-recognized OSS contributors for interviews. With the data collected by interviews,

they discovered the prevalence of community-centric roles. They classified roles into two groups: project-centric roles and community-centric roles. Project-centric roles include OSS coder, OSS project manager and OSS system admin. Community-centric roles include OSS community founder, community manager, and OSS community manager can be further divided into OSS strategist, OSS mentor, OSS treasurer, OSS writer, OSS advocate, and OSS license manager. The definitions of these roles are shown in Table 3.14.

Trinkenreich et al. found that people can build a career in OSS through different roles and activities, and people's career pathways are fluid, moving between project and community-centric roles. This study is based on interviews instead of data collected from projects, so they suggested to conduct an in-depth analysis of community-centric roles, to understand what projects currently do to foster them, and what are the needs for these roles to flourish. They did not propose measurements for the community-centric roles, so they also suggested researchers to work on proposing ways to identify and measure the evolution of stakeholders performing nontechnical roles in OSS.

3.3 Role identification methods

Previous research gave different definitions and identification criteria to different roles, roles under the same name could have different identification methods while roles under different names could share similar identification methods. In this section, I categorize the role identification methods used by the primary studies into six categories: membership based, count based, network based, turnover based, usage based, and development process based.

Roles	Definition
Community-centric roles	The roles related to the creation and management of the community
Project-centric roles	The roles related to the project deliverable
OSS community founder	Contributors who create a new product or new project
OSS advocate	Contributors who develop plans to bring new contributors, increasing contributions to the OSS project, and making the community inclusive, welcoming and safe
OSS mentor	Contributors who train newcomers
OSS strategist	Contributors who foster the adoption of OSS technology or improve its processes and improve transparency in organizations or communities
OSS treasurer	Contributors who lead strategical budgetary decisions for the OSS project or foundation
OSS writer	Contributors who contribute to documentation
OSS license manager	Contributors who oversee the compatibility and compliance of software licenses
OSS coder	Contributors who perform activities related to developing new code, maintaining existing code, and writing tests
OSS project manager	Contributors who perform management-centric activities such as managing budget, and perform product-centric activities such as managing releases and project deliverable.
OSS system admin	Contributors who support the base operational systems, selects, configures, connects, and fine-tune the subsystems that are components of a robust and efficient larger part

Table 3.14: Definitions of roles in the study of Trinkenreich et al.

3.3.1 Membership based

This category represents the role classification methods based on whether the developer is recognized as a project member. Such membership can be observed by different ways and usually related to the privileges to manage the project. There are five studies under this category, these studies are shown in Table 3.15. The roles investigated and identification criteria used by these studies are listed in Table 3.16. Although the roles studied by these literature are similar, the methods used to identify these roles are quite different, range from scraping visible roles on web pages to inferring write permission from event logs. The

definitions and the methods they used are presented in this section.

Study	Year	Reference
[PS29]	2017	Zhou et al. [59]
[PS31]	2017	Coelho and Valente[12]
[PS34]	2018	Middleton et al. [29]
[PS35]	2018	Calefato et al. [7]
[PS37]	2018	Dias et al. [15]
[PS38]	2018	Valiev et al. [51]
[PS43]	2019	Tan [44]
[PS47]	2020	Wang et al. [53]
[PS52]	2020	Wang [54]
[PS55]	2020	Tan et al. [45]

Table 3.15: Literature relevant to membership based role identification methods

Roles	Identification criteria	Data source
Insider, outsider	Visible role	GitHub interface
Core, peripheral	Committer	Git repository
Internal member, external member	Role flag and public profile	GitHub pull request, public profile
Elite, nonelite	Events related to write permission	GitHub project events
Maintainer	Repository owner	GitHub
Maintainer	Contributors' label	MAINTAINERS files of Linux kernel
Maintainer	Visible role	PyPI

Table 3.16: Roles, identification criteria, and data source used in membership based methods

Insider and outsider. Middleton et al. studied project communities on GitHub to discover which forms of software contribution characterize developers who join the project team, and they proposed two roles, insiders and outsiders, to define membership on GitHub [29]. The insiders are contributors who obtain the ability to commit changes to the project repository, while outsiders are the contributors without such write access. To identify the insiders, Middleton et al. collected the developers' roles on GitHub interface, and consider the roles that indicate "write access", such as "Collaborator" or "Member", as insiders. Although it is easy to collect the role information when it is visible, there are other challenges. One is that the available roles have changed during their study, and they had to find the correspondence

between roles manually. Another challenge is that GitHub provides the option to change the default visibility of project roles. To minimize the possible influence caused by invisible roles, they consulted the commit history of projects on GHTorrent where they can also identify insiders, since there are actions that can only be performed by project members.

Core and peripheral. As mentioned in section 3.2.1, Calefato et al. classified developers into core and pheripheral developers based on commit access.

Internal member and external member. Unlike the previous two membership-based role classifications, the classification used by Dias et al. is not based on "write access" to repositories [15]. In this paper, Dias et al. investigated the activity of internal and external developers. Internal developers are contributors that work for the company that open-sourced the project, while external developers are contributors that do not work for that company. They also utilized the features provided by GitHub to collect data, but instead of using visible roles on GitHub interface, they identify role of developer based on a flag "site_admin". This flag indicates the role of the site administrator with permissions to manage high-level application and VM settings, all users and organization account settings, and repository data. They consulted GitHub representatives and confirmed that this flag is only true for GitHub employees. They also analyzed the public profiles of the top 10 contributors to avoid false negatives (a paid that does not have its site_admin flag enabled). After the role identification, the analysis of contributors showed that both internal and external developers are rather active when it comes to submitting pull requests. Many externals play important roles in the studied projects, but internal developers still play the central roles in the projects. The differences between internal and external developers are that the majority of the external developers are casual contributors, external developers' contributions range from documentation to complex code.

Elite developer and nonelite developer. In 2020, Wang et al. investigated a group of developers named elite developers [53]. In this study, they intend to analyze elite developers'

activities in a comprehensive way. They defined elite developers as developers who own administrative privileges in the project and identified elite developers by developers' write permission for repository. They collected data from the GitHubArchive public data dump on Google Cloud and GitHub API. To identify the elite developers, they analyzed the event data to determine if a developer has write permission, since the list of developers' permissions is only accessible for repository owners. By analyzing the activities of elite developers, Wang et al. found that elite developers participate in a variety of activities, and elite developers tend to put more effort into supportive and communicative activities and less effort into coding as the project grows. As for the impact of elite developers' activities on project outcomes, they found that elite developers' efforts in nontechnical activities are negatively correlated with the project's outcomes in terms of productivity and quality in general, except bug fix rate.

Later, in a follow-up study, Wang presented a fresh approach to investigate developers' public activities to advance their understanding and further support the OSS community [54]. The role studied in this research is still elite developers and have the same identification criteria as previous work. In this study, Wang designed a tool that can be used to collect, model, and analyze elite developers' online contributing activities. The automation approach proposed can leverage GitHub's event log and apply a set of statistic methods to provide analytic support to elite developers.

Maintainer. In 2017, Zhou et al. used qualitative methods to understand maintainer behavior and to design suitable measures for maintainers' work, then use maintainers' activity data to quantify the growth of the system, the growth of workloads, work distribution and scalability of maintainers' work [59]. Zhou et al. choose Linux kernel as the subject. In this study, the maintainers as part of the core group, and some of them do not write code. Zhou et al. identified maintainers based on contributors' label, only individuals explicitly labeled as "maintainer" in MAINTAINERS files were considered as maintainer. Through a

mixed method of qualitative and quantitative analysis, they found that there are systematic differences among modules, and most of the modules have not grown appreciably over the last decade. For maintainers, the effort per maintainer does not increase, but the distribution of work is highly unbalanced, suggesting that a few maintainers may experience an increasing workload. They also discovered that assigning multiple maintainers to a file yields only a power of 1/2 increase in productivity.

Since only individuals explicitly labeled as "maintainer" were considered, there could be unrecognized maintainers. Moreover, their approach did not consider all the ways maintainers' effort, such as participation in discussion. There are also other limitations, such as only considering the mainline repository, and the uniqueness of Linux kernel limits the generalization. In that case, another research that using the similar method but has a more precise maintainer identification and using a set of representative OSS projects could provide better insights into maintainers' behavior.

Tan applied the identification method used by Zhou et al. In 2019 [44]. In Zhou et al.'s study, they found that assigning multiple maintainers to a file yields only a power of 1/2 increase in productivity, but they did not provide solutions to adjust the workflow to adapt the project. In Tan's study, they evaluated a new workflow: the multiple-committer model (MCM) that was applied by a subsystem of the Linux kernel to confront the heavy workload of the maintainers.They found that the model works well on the i915 subsystem and appears to effectively reduce the pressure in the subsystem. And they obtained 3 dimensions of factors, a system can determine if it meets these factors and then decide regarding the adoption. Later, in a follow-up study, Tan et al. also used this identification method and continued to investigate the model, MCM, to relieve the burden on maintainers [45]. Apart from that, they reviewed the online documents related to the MCM and conducted interviews to understand which factors are important for the implementation of the MCM. They also proposed an approach to identify potential committers using a collaboration network.

Finally, they conducted interviews to validate the generality of the results.

Understanding the reasons that lead to failure will help maintainers prevent failures happen. Coelho and Valente investigated to reveal the reasons and strategies that could be effective in 2017 [12]. Coelho and Valente first send the survey to maintainers on GitHub to gather information about the reasons behind failures. In this study, the maintainers they choose are the repositories' owners or projects' principal contributors if the repositories are owned by organizations. They discover a set of nine reasons for the failure based on analysis, the top reasons include lack of time, lack of interest, project is completed, usurped by competitor, project is obsolete, project is based on outdated technologies, low maintainability, and conflicts among developers. They also collected projects' information about the practises applied and show that some maintenance practices have an important association with a project's failure or success. Moreover, they collect data from the issues of failed projects, and reveal the principal strategies developers have tried to overcome the failure of the studied projects. Based on their work, researchers can work on defining and validating ""maturity models"" for open source projects, and minimize the risks of adopting the projects. They also provide future work like conducting investigation on proactive strategies to avoid the failure of projects, such as maintainer recommendation system.

Code reuse brings many benefits to modern software development, which will create complex networks of interdependencies of projects. Such a network of projects is named ecosystem. Valiev et al. studied ecosystem-level factors affecting the sustainability of open-source Python projects in 2018 [51]. In this study, Valiev et al. collected a panel data set of Python PyPI packages and modeled the factors that explain projects becoming dormant. Then they interviewed package maintainers to triangulate the model results and refine the discovered effects. Valiev et al. define maintainers as the same group of developers as core contributors, and they are highly active and have the deepest knowledge of the code base. They did not disclose the method they used to identify maintainers. However, based on PyPI documen-

tation, maintainer is a collaborator role available for a project on PyPI, so they are likely to identify the maintainer based on that.

3.3.2 Count based

The studies in this section classified developers based on quantitative measures of contributors' contribution, including the number of commits, lines of code (LoC), and so on. There are ten studies which used count based role identification methods, these studies are shown in Table 3.17. In the study of Coelho and Valente, they identify maintainers as project owners or principal developers. They did not specify how they determine a developer is principal developer or not, however, they are likely to use the code contribution as measure to identify principal developer. The identification criteria and data source used by these studies are listed in Table 3.18.

Study	Year	Reference
[PS3]	2006	Herraiz et al. [20]
[PS6]	2009	Shibuya and Tamai [36]
[PS13]	2012	Wagstrom et al. [52]
[PS30]	2017	Cheng et al. [10]
[PS27]	2017	Lee and Carver [26]
[PS31]	2017	Coelho and Valente [12]
[PS32]	2018	Coelho et al. [13]
[PS44]	2019	Müller [30]
[PS41]	2019	Cheng and Guo [11]
[PS49]	2020	Canedo et al. [8]

Table 3.17: Literature relevant to count based role identification methods

3.3.3 Network based

Different from classifying developers' roles based on their membership or quantitative measures of their contribution, another approach is conducting social network analysis. By

Study	Identification criteria	Data source
[PS27]	Percentage of code contribution	Project repositories
[PS31]	Principal developers	GitHub repositories
[PS32]	Number of commits	GitHub project repositories
[PS44]	Lines of code	Git repositories
[PS49]	Truck Factor	GitHub project repositories
[PS3]	Level of activity	CVS, mailing list, and Bugzilla
[PS6]	Frequency of contribution	Bug tracking system and revision control systems
[PS13]	Number of issues, commits and followers	Project repositories and users related
[PS30]	Percentage of code contribution	Development data of GNOME
[PS41]	6 factors	GitHub project repositories

Table 3.18: Identification criteria, and data source used in count based methods

constructing a network of developers with their direct or indirect interactions, researchers can observe the hierarchy of contributors, and identify roles of developers based on their characteristics revealed by the network. Although the studies under this category all use network analysis methods, they utilized different measures and different data source.

The studies in this section all classify developers based on the onion model. There are three studies under this category, these studies are shown in Table 3.19. The roles investigated and identification criteria used by these studies are listed in Table 3.20.

Study	Year	Reference
[PS7]	2010	Oezbek et al. [34]
[PS19]	2014	Bosu and Carver [5]
[PS26]	2017	Joblin et al. [23]
[PS30]	2017	Cheng et al. [10]

Table 3.19: Literature relevant to network based role identification methods

3.3.4 Turnover based

Turnover is the phenomenon of continuous influx and retreat of human resources in a team. Foucault et al. conducted a research to study developer's turnover in OSS. In this study,

Study	Identification criteria	Data source
[PS7]	Social network analysis	Mailing list
[PS19]	Social network analysis using K-means clustering algorithm based on centrality measures	Code review
[PS26]	Five network-based operationalizations and the standard 80th percentile threshold	Email, VCS, code review
[PS30]	Social network analysis	development data of GNOME

Table 3.20: Identification criteria, and data source used in network based methods

they classified developers based on external and internal turnover. The external turnover refers to the movements of developers in and out of a project, while internal turnover refers to the movements of developers inside a project. They defined three roles under the context of both internal turnover and external turnover, these roles are newcomer, leaver, and stayer. The definitions of these roles are given in Table 3.22.

In this study, they collect information from centralized VCS to identify authors, measure quality of project's modules and modulate project. They proposed and investigated metrics to measure turnover in OSS projects. Based on the results, they found that high turnover lead to success, but external turnover has a negative impact on the quality of the modules. And they also showed turnover patterns which reveals that the projects act differently regarding turnover.

Onboarding

Onboarding is a process that helps newcomers become integrated members of their organizations [16]. There are two roles involved in this process: newcomer and mentor. Newcomers are the developers who are new to a project and want to contribute, while mentors are experienced developers who support newcomers to get through the onboarding process and start to contribute. I put this as a subsection of Turnover based category, since the major

participant of this process, newcomer, has the same definition as external newcomer defined in turnover.

Onboarding is an important process for OSS projects to be sustainable, since the contributors, especially peripheral contributors, are constantly changing. There are ten studies under this category, these studies are shown in Table 3.21. Researchers have studied the barriers faced by newcomers and mentors in onboarding process [39] [3] [41], proposed assistance tools [9] [40] [35] for newcomers. There are also a number of studies that provide suggestions and practises for projects to attract and assist newcomers [16] [37] [46] [36].

Study	Year	Reference	Research focus
[PS6]	2009	Shibuya and Tamai [36]	Project characteristics
[PS14]	2012	Canfora et al [9]	Assistance tool
[PS17]	2014	Fagerholm et al. [16]	Mentoring, project characteristics
[PS21]	2015	Steinmacher et al. [39]	Barriers
[PS23]	2016	Steinmacher et al. [40]	Assistance tool
[PS25]	2016	Sarma et al. [35]	Assistance tool
[PS36]	2018	Balali et al. [3]	Barriers
[PS45]	2019	Steinmacher et al. [41]	Barriers
[PS51]	2020	Tan et al. [46]	Onboarding practise
[PS53]	2020	Silva et al. [37]	Onboard and motivate students

Table 3.21: Literature relevant to onboarding based role classification

Barriers. To assist newcomers onboard, it is necessary to identify the barriers they meet in the onboarding process and find suitable practises to overcome these barriers [39]. Steinmacher et al. conducted an empirical study to address this issue in 2015. In this study, Steinmacher et al. identified the the barriers from multiple sources including systematic literature review, open question responses, students' feedback and semi-structured interviews. They defined a model composed of 58 barriers including 13 social barriers. Thus provide a solid basis for practises and tools that can be used to overcome these barriers.

In 2018, Balali et al. identified 44 barriers faced by newcomers and mentors in OSS projects [3]. Different from newcomers' barriers identified by Steinmacher et al. [39], they 19 barriers

that affect mentors and 16 newcomers' barriers that have not been previously identified. They collected data by interviewing selected mentors. Through analysis, they found some barriers affect only newcomers or only mentors, other barriers affect both newcomers and mentors. Moreover, most of the barriers identified relate to personal and interpersonal issues. They also uncovered strategies to help newcomers overcome barriers and find gap in how to help newcomers deal with social barriers.They also discovered some gender-specific challenges and identified factors that influence the onboarding and retention of women contributors in OSS community.

In 2019, Steinmacher et al. reported a two phase study which consists of designing a model of 58 barriers [39] and proposing a portal [40]. This paper is a summary of their previous work, and they provided future directions such as using the model to plan further qualitative and quantitative studies to investigate specific barriers.

Tools and guidance. In 2012, Canfora et al. proposed an assistant tool named Yoda. Yoda is an approach to identify mentors by relying on historical data from the mailing list and versioning systems, and then recommend them when a newcomer joins the project. This paper defines five factors, inspired by ArnetMiner, that can be used to identify mentors, and these factors will be computed based on the communication network constructed with extracted data. These factors capture the interactions between this newcomer and possible mentors, the difference of newcomer and mentors in terms of experience in this project, and mentors' technical activity. After identifying candidate mentors with these factors, Canfora et al. adapted an approach proposed for bug triaging to find an appropriate mentor based on textual analysis of newcomer's request. The performance of Yoda is evaluated through an empirical study, developers in five OSS projects are selected as participants for the survey. The results show that Yoda can provide more precise recommendations compared to the recommendation approach relying on top committers.

After identifying the barriers faced by newcomers, Steinmacher et al. proposed a portal

named FLOSScoach to support newcomers in 2016. This portal is based on the model of barriers they defined in previous work, and it provides information and strategies to assist newcomers to overcome these barriers.

Apart from presenting the design of this portal, Steinmacher et al. also performed a study to evaluate the portal. This study analyzed diaries, self-efficacy questionnaire, and a questionnaire based on Technology Acceptance Model. The results indicate that FLOSScoach is capable of guiding newcomers and lowering barriers related to the orientation and contribution process, but it was not effective in lowering technical barriers.

After building the web portal FLOSScoach, Sarma et al proposed another system called BugExchange in 2016. BugExchange is a system that curates tasks from OSS projects to help train newcomers. Through mining project repositories, BugExchange can collect tasks, then it will analyze tasks to identify the skills required and complexity for each task. With such information, BugExchange will recommend tasks to newcomers and provide a network of near-peer mentors.

Sarma et al. conjecture that BugExchange may reduce newcomer dropouts and foster more casual contributors, but evaluation of the system is not performed. They adapted the same evaluation procedure used for FLOSScoach, but also planned to collect more information from other relevant people like instructors and observe the use of this system. They also planned to incorporate BugExchange into FLOSScoach.

Suggestions for projects. To examine how mentoring and project characteristics influence the effectiveness and efficiency of the onboarding process, Fagerholm et al. conducted a study in 2014. The influence of mentoring is analyzed by comparing the performance of developers who are onboarded by the collaboration program with developers who joined the same projects by nature process. Fagerholm et al. collected developers' activity in terms of speed of contribution and total number of commits, and the results indicate that the

mentoring has had a positive influence on the effectiveness and efficiency of the onboarding process. As for project characteristics, they selected five characteristics that could influence the onboarding process. The characteristics include commits, contributors, appeal, lifetime, and peripheral contribution. To analyze the influence of these characteristics, they measured four projects' characteristics and compared them also in terms of speed of contribution and total number of commits. The results show that if the project which has higher numbers on these characteristics, their contributors also have higher speed values and higher level of contribution. Therefore, they claimed that higher numbers in size, appeal, and lifetime are related to an improvement in the performance of new developers. Similar to Fagerhom et al.'s work, Shibuya and Tamai conducted a study to find the main projects' attributes where successful newcomers contribute to them. In this study, they take developers with more followers as successful and popular developers, and collect these developers' activities from GHTorrent. By analyzing these data, they extracted a list of project characteristics that newcomers contribute to them for the very first of their joining period in GitHub. The list consist of 15 attributes and are sorted based on their importance, the importance of each attribute is computed using Random Forest.

To support newcomers onboarding, GitHub encourages projects to apply labels such as good first issue (GFI) to tag issues suitable for newcomers. But, many newcomers still fail when they attempt to solve these issues. In 2020, Tan et al. conducted a preliminary study on this mechanism from its application status, effect, problems, and best practices. To collect data, Tan et al. first constructed a dataset with GHTorrent. They analyzed 9,368 GFIs from 816 popular GitHub projects and found GFIs need more days to be solved, and almost half of GFIs are not solved by newcomers. After that, they conducted email survey to analyze factors and problems related to the effectiveness of GFIs, and summarized strategies to identify appropriate GFIs.

Students are a special group of contributors in the OSS community, and previous studies

have only focused on software developers in general. Researches that foster the participation of students not only can increase the OSS workforce, but it also benefit students, since online contributions are considered when making hiring decisions. Silva et al. conducted a study on analyzing what motivates students to participate in OSS and how to onboard them. In this study, Silva et al. took Google Summer of Code (GSoC) program as subject, and developed an engagement theory that explains how to onboard students and how students become motivated to participate. This study is composed of three phases. In the first phase, they build the onboarding theory based on the analysis of applications. And in the second phase, they build the motivation theory based on empirical data. Finally, they present the theory to student and analyze the perceptions. The students' perceptions indicate that the motivational theory broadened their understanding of GSoC and inspired them to engage in such programs. By using GSoC as subject, the generalizability of their theory could be limited. Future research can extend the results by using larger set of OSS projects as the subject, and it is also necessary to evaluate this theory with a larger sample size.

Roles	Definition
Newcomer	Developers who joined the team of a module in the period P_2
Leavers	Developers who left the team of a module within the period of P_1
Stayers	Developers who contribute to a module in both P_1 and P_2

Table 3.22: Definitions of roles in the study of Foucault et al.

Quasi contributor

In 2018, Steinmacher et al. investigated how and why quasi-contributors fail [42]. Quasi-contributors are external developers who did not succeed in getting their contributions accepted to an OSS project. Based on this definition, quasi-contributors are a subset of external newcomers. In this study, they identified quasi-contributors as newcomers who had no "accepted contribution" to that specific project, while the "accepted contribution" are any changes that passed the pull-request cycle and merged to the project code base. I find

no role similar to quasi-contributor in the existing role classification models. Another role involved in this study is integrator, they are developers who merge pull requests.

In this study, Steinmacher et al. analyzed pull requests of selected popular OSS projects on GitHub and conducted two surveys with quasi-contributors and integrators separately. They found that quasi-contributors are common, and the most common reason for nonacceptance was "mismatch between developer's and team's vision/opinion". Moreover, one-third of the developers disagreed with their nonacceptance and declared the nonacceptance demotivated or prevented them from placing another PR.

First time contributor

In 2020, Subramanian investigated the characteristics of the first pull request made to an OSS project by developers to understand the first time contributor [43]. In this study, they mined the first pull request using GitHub API, and collected the user's first contribution in GitHub. First time contributor is similar to the newcomer, but the first time contributions studied in this paper are not in terms of specific projects but a community (GitHub). This paper revealed the top 15 languages/file types used and the size of contributions in first PR. Based on the results, they suggested moderators to prioritize tasks based on the top languages and file types, and the first time contributors should not be discouraged to take up big tasks.

3.3.5 Event based

Event based identification methods refers to methods that identify contributors based on specific event they participate, such as code review, bug report and bug patch.

Review

To ensure the quality of code, developers' submission to projects usually need to be reviewed by other contributors. This category include identification methods that identify reviewers based on the data related to review event. There are three studies under this category, these studies are shown in Table 3.23. The identification criteria and data source used by these studies are listed in Table 3.24.

Study	Year	Reference
[PS5]	2009	Nurolahzade et al.[33]
[PS24]	2016	Zhu et al. [61]
[PS39]	2018	German et al. [18]

Table 3.23: Literature relevant to review based role identification methods

Study	Identification criteria	Data source
[PS5]	Reviews in bug reports	Mozilla Bugzilla database
[PS24]	Code review comment	GitHub code contribution data of four projects
[PS39]	Code review	OpenStack code review

Table 3.24: Identification criteria, and data source used in review based methods

In 2009, Nurolahzade et al. examined the development process of Mozilla foundation and highlighted how different parties involved affect and steer the process [33]. Apart from the developers who submit the patch, they found module owners and peer developers are involved in the patch review process. Module owners are the developers who have authority of a module, a change to the code need to be approved by module owner, while peer developers are involved by conducting peer reviews.

Nurolahzade et al. mined Mozilla Bugzilla database to collect bug reports, patches and relevant developers. The methods they used to identify the module owners, module owner peers and peer developers are not disclosed. By analyzing the events, they found that most development and peer reviews in Firefox come from a group of developers who make up the core developer group, and the assigned developers are primarily responsible for bug reports

and are supported by peer developers and module owners. They also found that peers decrease module owners' workload by providing ideas before a patch is developed, reviewing patches before module owners, and finding and reporting back errors. They also discovered a pattern that module owners are concerned about long-term maintainability, while peers are interested in functionality and usability, which benefits module owners as they can use the feedback to make a better decision.

Fairness of code review is important to the quality of software products, as it can impact the productivity and motivation of participants. German et al. proposed a framework that describes how fairness affects modern code review and evaluates the role of fairness in the code review process. There are two roles involved in the code review process, they are authors of the code and reviewers. In this study, German et al. invited authors and reviewers to participate in the survey, they did not present the process they identify these participants. Their results presented evidence that fairness is an issue, and they presented a set of guidelines to address unfairness.

In 2016, Zhu et al. compared patch-based and pull-request-based tools in terms of code contribution effectiveness [61], and such comparison may lead to strategies and practices making the code contribution more satisfying and efficient from both contributors' and maintainers' perspectives. The roles studied in this paper are contributor and maintainer. In this study, Zhu et al. synthesized the results of published papers and investigated four GitHub projects. They first mined project repositories for information such as issues, review, and pull request, then they borrowed metrics for analyzing contribution practice, and conducted two sub-studies to address the internal and external validity. The contributors and maintainers are identified based on data such as the submitter and operator of pull requests. They found that modern tools, such as PR systems, have a lower processing time and attract more participation, and these improvements are at least partially attributed to the advanced features of PR systems.

Bug report and bug fix

Developers not only make code contributions to add new features or modules to OSS projects, they also report bugs and submit bug patches to improve the code quality and make OSS projects sustainable. As mentioned in the previous sections about the onion model, bug reporter and bug fixer can be identified by investigating bug tracking system such as Bugzilla. They are also called bug submitter and bug committer in the study of Sinha et al. [38], or issue reporter and issue fixer in the study of Cheng and Guo [11]. Such distinct event makes it easy to identify and separate them from other contributors.

In 2014, Tsay et al. conducted another study to investigate how developers in open work environments evaluate and discuss pull requests [50]. The third party audience member identified in this study are similar to the bug reporter, however, third party audience members only report problems but do not actually contribute code. These third party developers hold a stake in particular code contributions, often needing a particular change for their own usage. They will apply pressure to core members to influence their evaluation decisions. In this study, they collected data from a sample of extended discussions around pull requests and interviews, and they found three types of participants are involved. Apart from third party audience members, the participants also include submitters and project core members. The submitters are developers who submit the pull requests, and core members are identified based on their commit access.

There is another group of developers that contribute to the bug report and fix of OSS projects, but they participate the bug bounty programs instead. An bug bounty programs is a reward program offered by an organization to external parties, authorizing them to perform security assessments on the organization's assets. Considering the reward offered, bug bounty contributors and other bug reporters could have a qualitative difference.

To understand the characteristics of bug bounty program contributors, Hata et al. conducted

a study in 2017 [19]. In this study, they analyzed the history of bug bounty programs and contributors and conducted a survey. They further classified bug bounty contributors into non-project-specific bug bounty hunters and project-specific security contributors, based on whether they contribute to specific programs. This study provided insights to make bug bounty programs better and insights for further studies of new software development roles.

3.3.6 Other methods

There are also other identification methods used by only a few literature, and cannot be categorized into the types of role identification methods listed above. I also put the studies which did not specify the identification methods in this subsection.

Usage based

In 2010, after conducting an interview to understand how members of OSS community perceive usability issues, Terry et al. discovered a way to distinguish users into four groups. These groups are reference users, bleeding edge users, stable release users, and linux distribution users, and the first two groups are referred as core users. The definitions of these roles are presented in Table 3.25. In this study, they found that FOSS project members possess rather sophisticated notions of software usability, and uncovered a wide range of practices that ultimately work to improve software usability.

Unspecified methods

I found three studies that investigated developers' roles but did not present the identification criteria used. Two of the studies investigated maintainers while another study investigated lead and core developers.

Roles	Definition
Core user	Motivated users who closely follow and interact with the project.
Reference users	Users with close social ties to individual project members, with domain expertise and extensive experience using the software
Bleeding edge users	Users who use nightly builds of the software.
Stable release users	Users who use the software and provide feedback when it is released as a stable version
Linux distribution users	Users who only use software provided in their Linux distribution

Table 3.25: Definitions of roles in the study of Terry et al.

Zhou et al. investigated how open-source projects on GitHub differ with regard to forking inefficiencies [60]. They conducted interviews with active developers, including 15 maintainers and fork owners. However, they did not disclose the method they used to identify active developers or maintainers. Besides conducting interviews and literature analysis to identify potential context factors of forking inefficiencies, they also mined repository dumps on GHTorrent for information like forks, external commits, external pull requests and issues. Then they used multiple regression modeling to test the correlation between hypothesized context factors and outcome. This study revealed that there are significant inefficiencies in the collaborative development process of many communities, and many of these inefficiencies associate with common project characteristics and practices. Based on the analysis, Zhou et al. provided specific best practices for project maintainers to reduce forking-related inefficiencies.

Apart from reducing the workload of maintainers by using better workflow, software bots can also be helpful by automating routine tasks and interacting with human developers. However, software bots can be disruptive [6], especially for newcomers considering the social barriers they face [40]. Under this context, Wessel conducted a research to design and evaluate strategies to mitigate problems related to software bots [56].

In this study, there are three phases. The first phase includes analyzing pull requests, interviewing with contributors, maintainers, and bot developers. Similar to the work of Zhou et al. [60], this study also used projects on GitHub as subject, and it also did not present the method they used to identify maintainers. The next two stages include design strategies to support developers' interactions and transforming design strategies to bot prototypes. These two stages are not completed when they report their research. Although the focus of this study is not identify maintainers, presenting their strategy will increase the reliability of their study. This is especially true in GitHub, since different groups of maintainers could be considered as maintainers using different criteria. For example, if core developers are regarded as maintainers, then maintainers can be identified with the method used by middleton et al. [29] or network analysis used by Bosu and Carver [5].

In 2013, McDonald and Goggins conducted a study with lead and core developer to understand how OSS communities measure success [28]. In this study, they selected 10 lead and core members of three large OSS projects hosted on GitHub, and conducted semi-structured interviews. McDonald and Goggins did not disclose the definition or the methods they used to identify the lead and core developers, so I decided to take lead and core developer as a different role. Since this study aim to exam how OSS community leaders evaluate the success of their projects, this role could be related to a high level of participation and involved in project management, thus it could be a combination of core developers and project leaders (or OSS community managers).

McDonald and Goggins found that the number of contributors are most commonly cited as measures of success, and there is considerable value in making code submissions public through pull requests. They also discovered that visible activities on GitHub's interface are important indicators. Moreover, external metrics, like IRC channel, number of people visit API documentation, and number of downloads, are also important measures of success.

Tsay et al. conducted a study to understand how information in transparent OSS environ-

ments is used to evaluate contributions [49]. In this study, they studied the social connection between submitters and project managers, however, they did not disclose the method used to identify project managers. Since the dataset they used are a sample of pull requests, so the identification criteria they used are likely to be related to the push access to repositories. In that case, the project manager is similar to roles such as insider, core, internal member, and elite developers in membership based model or maintainer.

Tsay et al. found that project managers used information involving both good technical contribution practices for a pull request and the strength of the social connection between the submitter and project manager when evaluating pull requests. The results also revealed that pull requests with many comments were much less likely to be accepted, but this is moderated by the submitter's prior interaction in the project. In addition, well-established projects were more conservative in accepting pull requests.

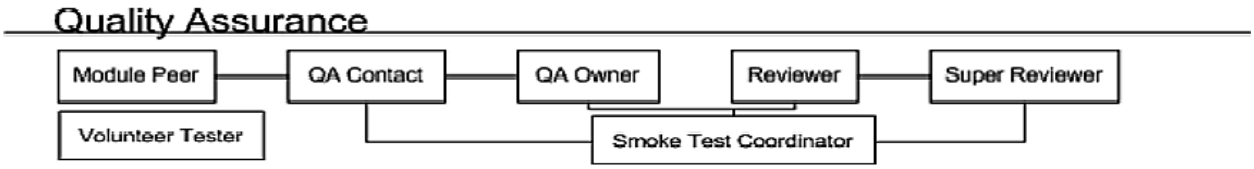
3.4 Role transition patterns

Apart from providing role classification models, some of the studies also investigated the role transition patterns. In this section, I will present the role transition patterns studied. Some patterns describe linear paths, such as onion model, type of engagement, development maturity roles and retiring path, however, developers may skip roles when migrate along the path. Other studies discovered transition paths that are more complex, such as the ones within project organizations, and the ones about project-centric and community-centric roles.

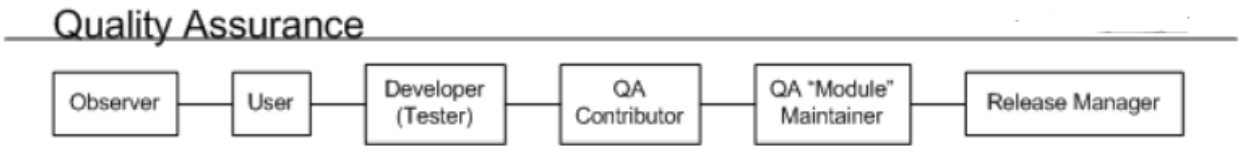
3.4.1 The onion model

In 2002, Nakakoji et al. proposed an onion model (Figure 3.5) to describe the general structure of OSS community [32]. They also studied the evolution of communities and

present a role transition path of developers. In this path, developers first are attracted to an OSS community as Passive Users, then gradually obtain understanding of the the system and recognition within the community. As they gain more understanding and recognition, their roles migrate from Passive Users to Readers, Bug Fixers and eventually Core Members. However, not all members want and will become Core members. This path describes a linear role transition pattern, and such pattern is later challenged by other researchers. In 2006, Herraiz et al. found that there is not a clear joining pattern, and its behavior does not seem to comply with the predictions of the onion model [20]. In this study, they discovered that the path given by onion model is followed only by the volunteer developers but not for hired developers. These two type of paths followed by different developers are also identified by Shibuya and Tamai in 2009 [36]. In 2010, Oezbek et al. conducted a study to investigate onion model [34]. They found that core member status may be qualitatively different, and the transition of individual mailing-list participants towards ever higher participation is qualitatively discontinuous. They did not propose any new transition pattern, but provided evidence that the existing transition pattern could be misleading. In 2011, Jergensen et al. studied the progression paths of developers and grouped the progression paths into five major categories based on their relationship to the socialization process in the software ecosystem [22]. These paths are social-technical path, accelerated path, technical-social path, technical only path, and source only path. In 2017, Cheng et al. investigated factors that will influence developers' role evolution [10]. In this study, they claimed that there are four evolutionary lines of developers. These evolutionary lines include from co-developers to core developers, from co-developers to project leaders, from core developers to project leaders, and from project leaders to core developers. They only studied the transition from core developers to project leaders.



(a) Mozilla.org



(b) NetBeans.org

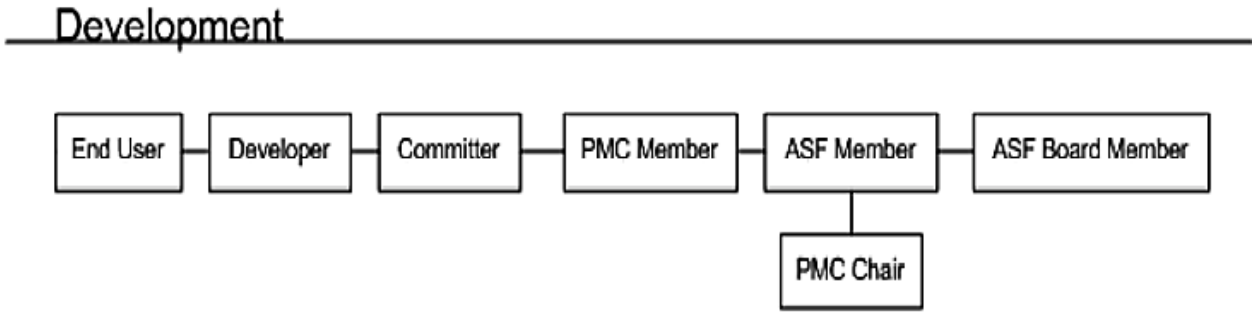
Figure 3.7: The quality assurance paths. Adapted from "Role Migration and Advancement Processes in OSSD Projects: A Comparative Case Study," by C. Jensen and W. Scacchi, 2007, 29th International Conference on Software Engineering (ICSE'07), pp. 364-374

3.4.2 Participative system

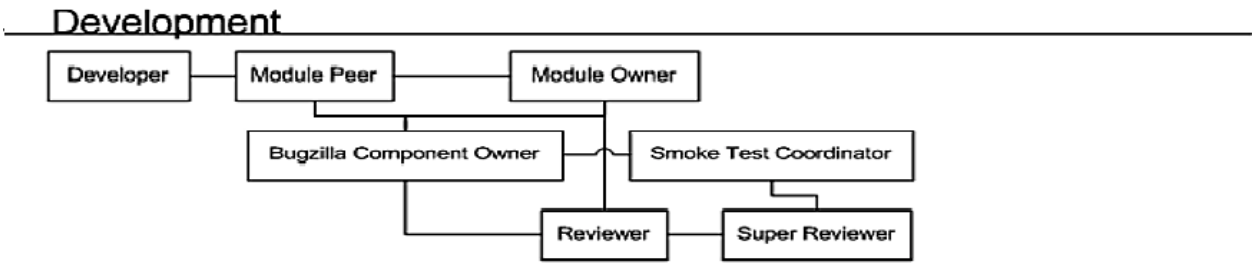
In 2005, Nakakoji et al. conducted a study that challenges the linear transition path of the onion model and argued that one's role change over time is a result of his/her changing type of engagement [31]. Such changes of types of engagement demonstrate a migration path of a participant engaging in a participative system. They proposed the transition path that individuals migrate along the spectrum of passive, active, and power engagement.

3.4.3 Within project organizations

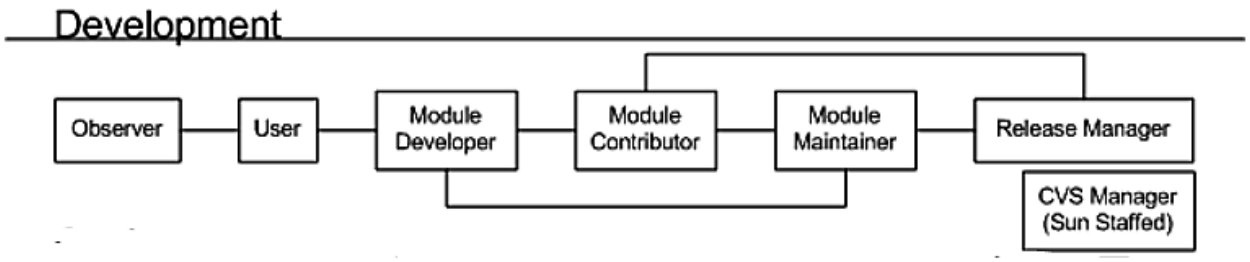
In 2007, Jensen and Scacchi investigated the role-sets, role migration and advancement process in three project organizations [21]. The role-sets covers various roles that participate different aspects of these three projects, including quality assurance (See Figure 3.7), development(See Figure 3.8), and so on.



(a) Apache.org



(b) Mozilla.org



(c) NetBeans.org

Figure 3.8: The development paths. Adapted from "Role Migration and Advancement Processes in OSSD Projects: A Comparative Case Study," by C. Jensen and W. Scacchi, 2007, 29th International Conference on Software Engineering (ICSE'07), pp. 364-374

3.4.4 Development maturity roles

In Wagstrom et al.'s work [52], the development maturity roles they proposed provide a finer grained method to follow a user through their participation in a project as they move from an interested lurker to a core project member. In this transition pattern, it is possible to skip roles in this progression, but each individual occupies only a single role at a time.

3.4.5 Retiring path

In 2019, Cheng and Guo studied the role dynamics of OSS developers [11]. They presented the heatmaps of role transition frequency which implies multiple role transition paths. They discovered that people usually began to engage in a project by assuming Issue Fixer and Engaged Issue Reporter roles. They also found a possible retiring path which developers transit from Progress Controller to Occasional Issue Reporter.

3.4.6 Project-centric and community-centric roles

In 2020, Trinkenreich et al [48]. studied the career pathways of successful OSS contributors, and presented the career pathways they discovered based on the interview (See Figure 3.9). They found that developers' career pathways are fluid, moving between project and community-centric roles.

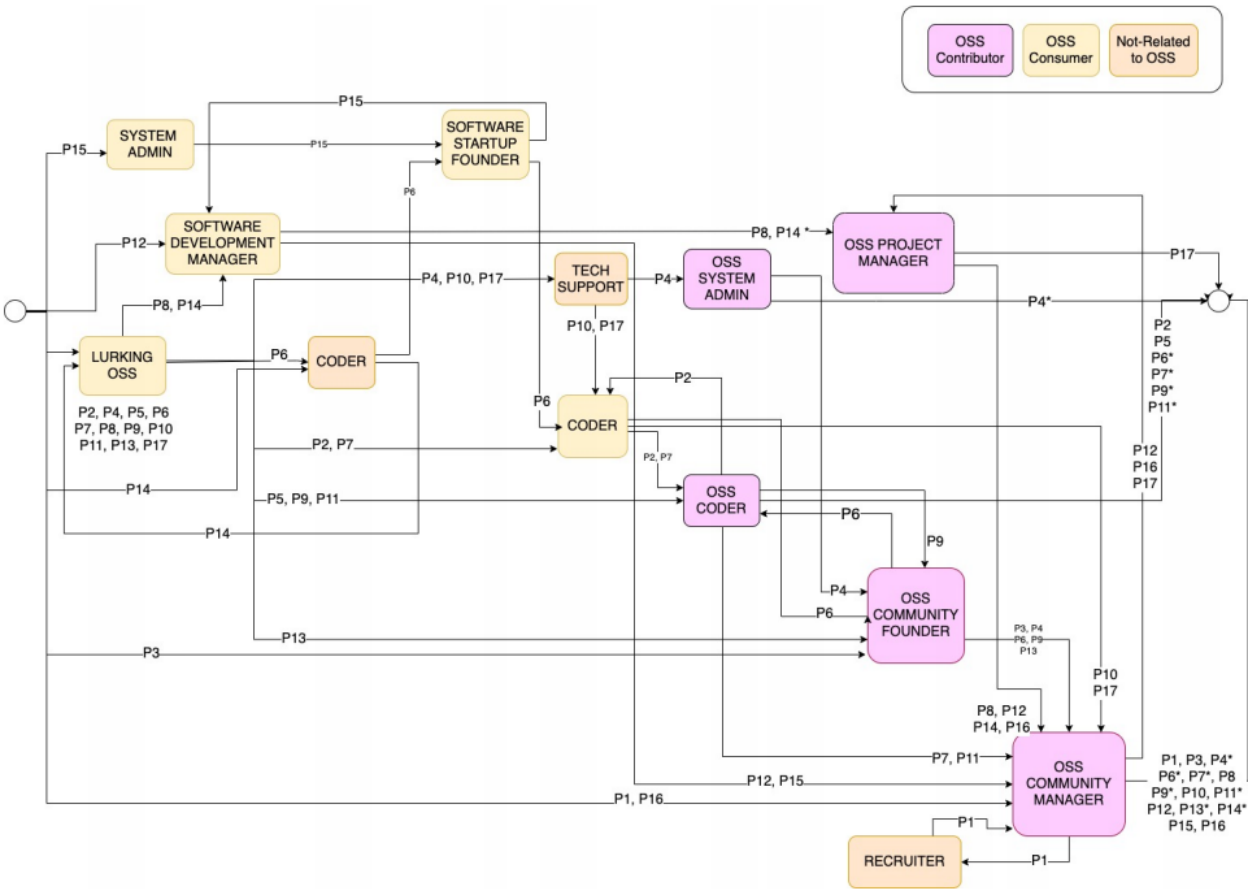


Figure 3.9: Overview of career pathways reported by interviewees. Adapted from "Hidden Figures: Roles and Pathways of Successful OSS Contributors." by Bianca Trinkenreich, Mariam Guizani, Igor Wiese, Anita Sarma, and Igor Steinmacher, 2020, Proc. ACM Hum.-Comput. Interact. 4, CSCW2, Article 180 (October 2020), 22 pages.

Year	Literature	Authors
2002	Evolution patterns of open-source software systems and communities [32]	Nakakoji et al.
2006	The processes of joining in global distributed software projects [20]	Herraiz et al.
2007	Role Migration and Advancement Processes in OSSD Projects: A Comparative Case Study [21]	Jensen and Scacchi
2009	Understanding the process of participating in open source communities [36]	Shibuya and Tamai
2010	How power users help and hinder open bug reporting [25]	Ko and Chilana
2010	The Onion Has Cancer: Some Social Network Analysis Visualizations of Open Source Project Communication. [34]	Oezbek et al.
2011	Entering the circle of trust: developer initiation as committers in open-source projects [38]	Sinha et al.
2011	The onion patch: migration in open source ecosystems [22]	Jergensen et al.
2012	Social coding in GitHub: transparency and collaboration in an open software repository [14]	Dabbish et al.
2012	Roles in a networked software development ecosystem: A case study in GitHub. [52]	Wagstrom et al.
2013	GitHub developers use rockstars to overcome overflow of news [27]	Lee et al.
2014	Impact of developer reputation on code review outcomes in OSS projects: an empirical investigation [5]	Bosu and Carver
2017	Classifying developers into core and peripheral: an empirical study on count and network metrics, etc. [23]	Joblin et al.
2017	Are one-time contributors different?: a comparison to core and periphery developers in FLOSS repositories [26]	Lee and Carver
2017	Developer Role Evolution in Open Source Software Ecosystem: An Explanatory Study on GNOME [10]	Cheng et al.
2018	Why we engage in FLOSS: answers from core developers [13]	Coelho et al.
2018	On developers' personality in large-scale distributed projects: the case of the apache ecosystem [7]	Calefato et al.
2019	Managing the open cathedral [30]	Müller
2019	Activity-based analysis of open source software contributors: roles and dynamics [11]	Cheng and Guo
2019	On the abandonment and survival of open source projects: An empirical investigation	Avelino et al.
2020	Hidden Figures: Roles and Pathways of Successful OSS Contributors. [48]	Trinkenreich et al.
2020	Work Practices and Perceptions from Women Core Developers in OSS Communities	Canedo et al.

Table 3.26: Literature studied role classification models

Chapter 4

Discussion

In this section, I review my results and provide some discussions about my findings.

4.1 Disclose the method used for role identification

Some studies did not provide the criteria used for role identification [60] [56], or the specific type of data they used for identification [12]. This makes their study hard to reproduce and also becomes a threat to validity. In the work of Zhou et al. [60], they interviewed maintainers of several popular OSS projects on GitHub, but did not provide the method they used to identify these maintainers. There could be different methods to identify maintainers of projects on GitHub, including methods based on visible roles or pull request events, so it will become a threat to validity of sample selection if no specific method is shown. Similarly, Wessel also did not disclose the method they used to identify maintainers [56]. In the study of Coelho and Valente, they defined maintainers as repositories' owner or projects' principal contributor, however, they did not present the method they used to distinguish principal contributors from the other contributors. As presented in the previous sections, there are

different ways to identify the core contributors, such as count-based and network-based, which could utilize different metrics of the project.

4.2 GitHub projects as study subject

Among all the primary studies, 21 studies collected data from projects hosted on GitHub, and the data sources include the dump on GHTorrent, GitHub user interface, and GitHub API. GitHub is chosen since a large quantity of different projects are hosted on GitHub and it provides a variety of metrics of projects. Considering the number of projects hosted on GitHub, it is easy to find different projects to obtain a representative sample. And since the researchers can apply the same data extraction procedure to all the projects, the cost of collecting a dataset of large size becomes affordable. However, as the type of information available could be different from platform to platform, such as assigned role of developers, I suggest future researchers not to limit their subjects in one platform. By gathering different type of information across various platforms, they may find more insights into characteristics of developers' roles.

4.3 The overlap and conflict of role identification

Among the role classification models presented in the previous sections, I find some similar roles in different models. The first set of similar roles are maintainer, core developer and code warrior, these roles are all related to a high level of contribution to the project. While another set of roles related to a low level of contribution are peripheral, rare contributor, nomad coder. Lurker, user, passive participant are similar roles that make no code contribution to the project, but they may contribute in another way, such as participating mailing list or issues of the project. There is also a set of roles that related to project progress management, they

are project leader, progress controller, OSS project manager, and OSS strategist. However, project leader is also defined as the initiator of the project in the studies of Nakakojio et al. and Cheng et al., which is identical to OSS community founder. Apart from that, there is a set of roles related to bug report, they are bug reporter, issues, bug submitter, and issue reporter. Internal member and internal collaborator are similar as well since they both include the contributors that work for the organization who own the project.While insider, core developer, and elite developer are similar as they can be identified based on write-access to the project.

Moreover, there are also roles that under the same name but have different identification criteria. One example is the maintainer, as shown in the previous section, maintainers can be identified based on TF algorithm, contributors' label or pull requests. There are other roles that also have different criteria, such as core, active contributor, and rockstar. Under diverse circumstances, the effectiveness of different identification criteria could be different as well. For researchers who intend to propose a finer model of OSS developers' roles, they need to take the effectiveness of different criteria into consideration.

Chapter 5

Conclusion

In this study, I identified 55 studies that investigated the roles of developers in the OSS community. I found six role classification models, and five major role identification methods: membership based, count based, network based, turnover based and event based. Moreover, I present the role transition patterns studies by these research. This study can be used as a baseline for future research on construction of a systematic and universal role classification model and corresponding role transition patterns of OSS developers. Finally, I provide a series of discussions about methods used by these studies and the roles studied.

Considering the role classification models studied, I found that the onion model has been further investigated and improved after emergence, such as studies which conducted verification of role transition or provided a finer definition of roles. However, other models have not receive much attention and been investigated by researchers under different conditions. It is also important to notice that no one model has used all types of role identification methods discovered in this paper, although they could have used more than one type of role classification methods. In that case, the current role classification models could be improved with further study in role identification methods. For future work, I suggest researchers to

conduct further research in the models other than the onion model and the effectiveness of different role identification methods, which could contribute to the construction of finer role classification models.

Bibliography

[1] G. Avelino, E. Constantinou, M. T. Valente, and A. Serebrenik. On the abandonment and survival of open source projects: An empirical investigation. In *2019 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, pages 1–12. ISSN: 1949-3789.

[2] Guilherme Avelino, Leonardo Passos, Andre Hora, and Marco Tulio Valente. A novel approach for estimating truck factors. pages 1–10.

[3] Sogol Balali, Igor Steinmacher, Umayal Annamalai, Anita Sarma, and Marco Aurelio Gerosa. Newcomers' barriers. . . is that all? an analysis of mentors' and newcomers' barriers in OSS projects. 27(3):679–714.

[4] Shahab Bayati. Understanding newcomers success in open source community. In *Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings*, ICSE '18, pages 224–225. Association for Computing Machinery.

[5] Amiangshu Bosu and Jeffrey C. Carver. Impact of developer reputation on code review outcomes in OSS projects: an empirical investigation. In *Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, ESEM '14, pages 1–10. Association for Computing Machinery.

[6] Chris Brown and Chris Parnin. Sorry to bother you: Designing bots for effective recommendations. In *2019 IEEE/ACM 1st International Workshop on Bots in Software Engineering (BotSE)*, pages 54–58.

[7] Fabio Calefato, Giuseppe Iaffaldano, Filippo Lanubile, and Bogdan Vasilescu. On developers' personality in large-scale distributed projects: the case of the apache ecosystem. In *Proceedings of the 13th International Conference on Global Software Engineering*, ICGSE '18, pages 92–101. Association for Computing Machinery.

[8] Edna Dias Canedo, Rodrigo Bonifácio, Márcio Vinicius Okimoto, Alexander Serebrenik, Gustavo Pinto, and Eduardo Monteiro. Work practices and perceptions from women core developers in OSS communities. In *Proceedings of the 14th ACM / IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, ESEM '20, pages 1–11. Association for Computing Machinery.

[9] Gerardo Canfora, Massimiliano Di Penta, Rocco Oliveto, and Sebastiano Panichella. Who is going to mentor newcomers in open source projects? In *Proceedings of the ACM*

SIGSOFT 20th International Symposium on the Foundations of Software Engineering, FSE '12, pages 1–11. Association for Computing Machinery.

[10] Can Cheng, Bing Li, Zeng-Yang Li, Yu-Qi Zhao, and Feng-Ling Liao. Developer role evolution in open source software ecosystem: An explanatory study on GNOME. 32(2):396–414. Company: Springer Distributor: Springer Institution: Springer Label: Springer Number: 2 Publisher: Springer US.

[11] Jinghui Cheng and Jin L. C. Guo. Activity-based analysis of open source software contributors: roles and dynamics. In *Proceedings of the 12th International Workshop on Cooperative and Human Aspects of Software Engineering*, CHASE '19, pages 11–18. IEEE Press.

[12] Jailton Coelho and Marco Tulio Valente. Why modern open source projects fail. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, ESEC/FSE 2017, pages 186–196. Association for Computing Machinery.

[13] Jailton Coelho, Marco Tulio Valente, Luciana L. Silva, and André Hora. Why we engage in FLOSS: answers from core developers. In *Proceedings of the 11th International Workshop on Cooperative and Human Aspects of Software Engineering*, CHASE '18, pages 114–121. Association for Computing Machinery.

[14] Laura Dabbish, Colleen Stuart, Jason Tsay, and Jim Herbsleb. Social coding in GitHub: transparency and collaboration in an open software repository. In *Proceedings of the ACM 2012 conference on Computer Supported Cooperative Work*, CSCW '12, pages 1277–1286. Association for Computing Machinery.

[15] Luis Felipe Dias, Igor Steinmacher, and Gustavo Pinto. Who drives company-owned OSS projects: internal or external members? 24(1):16.

[16] Fabian Fagerholm, Alejandro S. Guinea, Jürgen Münch, and Jay Borenstein. The role of mentoring and project characteristics for onboarding in open source software projects. In *Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, ESEM '14, pages 1–10. Association for Computing Machinery.

[17] Matthieu Foucault, Marc Palyart, Xavier Blanc, Gail C. Murphy, and Jean-Rémy 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 2015, pages 829–841. Association for Computing Machinery.

[18] Daniel M. German, Gregorio Robles, Germán Poo-Caamaño, Xin Yang, Hajimu Iida, and Katsuro Inoue. "was my contribution fairly reviewed?": a framework to study the perception of fairness in modern code reviews. In *Proceedings of the 40th International Conference on Software Engineering*, ICSE '18, pages 523–534. Association for Computing Machinery.

[19] Hideaki Hata, Mingyu Guo, and M. Ali Babar. Understanding the heterogeneity of contributors in bug bounty programs. In *Proceedings of the 11th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, ESEM '17, pages 223–228. IEEE Press.

[20] Israel Herraiz, Gregorio Robles, Juan JosÉ Amor, Teófilo Romera, and Jesús M. González Barahona. The processes of joining in global distributed software projects. In *Proceedings of the 2006 international workshop on Global software development for the practitioner*, GSD '06, pages 27–33. Association for Computing Machinery.

[21] Chris Jensen and Walt Scacchi. Role migration and advancement processes in OSSD projects: A comparative case study. In *Proceedings of the 29th international conference on Software Engineering*, ICSE '07, pages 364–374. IEEE Computer Society.

[22] Corey Jergensen, Anita Sarma, and Patrick Wagstrom. The onion patch: migration in open source ecosystems. In *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering*, ESEC/FSE '11, pages 70–80. Association for Computing Machinery.

[23] Mitchell Joblin, Sven Apel, Claus Hunsen, and Wolfgang Mauerer. Classifying developers into core and peripheral: an empirical study on count and network metrics. In *Proceedings of the 39th International Conference on Software Engineering*, ICSE '17, pages 164–174. IEEE Press.

[24] Barbara Kitchenham and Pearl Brereton. A systematic review of systematic review process research in software engineering. 55(12):2049–2075.

[25] Andrew J. Ko and Parmit K. Chilana. How power users help and hinder open bug reporting. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '10, pages 1665–1674. Association for Computing Machinery.

[26] A. Lee and J. C. Carver. Are one-time contributors different? a comparison to core and periphery developers in FLOSS repositories. In *2017 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, pages 1–10.

[27] Michael J. Lee, Bruce Ferwerda, Junghong Choi, Jungpil Hahn, Jae Yun Moon, and Jinwoo Kim. GitHub developers use rockstars to overcome overflow of news. In *CHI '13 Extended Abstracts on Human Factors in Computing Systems*, CHI EA '13, pages 133–138. Association for Computing Machinery.

[28] Nora McDonald and Sean Goggins. Performance and participation in open source software on GitHub. In *CHI '13 Extended Abstracts on Human Factors in Computing Systems*, CHI EA '13, pages 139–144. Association for Computing Machinery.

[29] Justin Middleton, Emerson Murphy-Hill, Demetrius Green, Adam Meade, Roger Mayer, David White, and Steve McDonald. Which contributions predict whether developers are accepted into github teams. In *Proceedings of the 15th International Conference on Mining Software Repositories*, MSR '18, pages 403–413. Association for Computing Machinery.

[30] Matthias Müller. Managing the open cathedral. In *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ESEC/FSE 2019, pages 1176–1179. Association for Computing Machinery.

[31] K. Nakakoji, K. Yamada, and E. Giaccardi. Understanding the nature of collaboration in open-source software development. In *12th Asia-Pacific Software Engineering Conference (APSEC'05)*, pages 8 pp.–. ISSN: 1530-1362.

[32] Kumiyo Nakakoji, Yasuhiro Yamamoto, Yoshiyuki Nishinaka, Kouichi Kishida, and Yunwen Ye. Evolution patterns of open-source software systems and communities. In *Proceedings of the International Workshop on Principles of Software Evolution*, IWPSE '02, pages 76–85. Association for Computing Machinery.

[33] Mehrdad Nurolahzade, Seyed Mehdi Nasehi, Shahedul Huq Khandkar, and Shreya Rawal. The role of patch review in software evolution: an analysis of the mozilla firefox. In *Proceedings of the joint international and annual ERCIM workshops on Principles of software evolution (IWPSE) and software evolution (Evol) workshops*, IWPSE-Evol '09, pages 9–18. Association for Computing Machinery.

[34] Christopher Oezbek, Lutz Prechelt, and Florian Thiel. The onion has cancer: some social network analysis visualizations of open source project communication. In *Proceedings of the 3rd International Workshop on Emerging Trends in Free/Libre/Open Source Software Research and Development*, FLOSS '10, pages 5–10. Association for Computing Machinery.

[35] Anita Sarma, Marco Aurélio Gerosa, Igor Steinmacher, and Rafael Leano. Training the future workforce through task curation in an OSS ecosystem. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, FSE 2016, pages 932–935. Association for Computing Machinery.

[36] B. Shibuya and T. Tamai. Understanding the process of participating in open source communities. In *2009 ICSE Workshop on Emerging Trends in Free/Libre/Open Source Software Research and Development*, pages 1–6.

[37] Jefferson Silva, Igor Wiese, Daniel M. German, Christoph Treude, Marco Aurélio Gerosa, and Igor Steinmacher. A theory of the engagement in open source projects via summer of code programs. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ESEC/FSE 2020, pages 421–431. Association for Computing Machinery.

[38] Vibha Singhal Sinha, Senthil Mani, and Saurabh Sinha. Entering the circle of trust: developer initiation as committers in open-source projects. In *Proceedings of the 8th Working Conference on Mining Software Repositories*, MSR '11, pages 133–142. Association for Computing Machinery.

[39] Igor Steinmacher, Tayana Conte, Marco Aurélio Gerosa, and David Redmiles. Social barriers faced by newcomers placing their first contribution in open source software

projects. In *Proceedings of the 18th ACM Conference on Computer Supported Cooperative Work & Social Computing*, CSCW '15, pages 1379–1392. Association for Computing Machinery.

[40] Igor Steinmacher, Tayana Uchoa Conte, Christoph Treude, and Marco Aurélio Gerosa. Overcoming open source project entry barriers with a portal for newcomers. In *Proceedings of the 38th International Conference on Software Engineering*, ICSE '16, pages 273–284. Association for Computing Machinery.

[41] Igor Steinmacher, Marco Gerosa, Tayana U. Conte, and David F. Redmiles. Overcoming social barriers when contributing to open source software projects. 28(1):247–290.

[42] Igor Steinmacher, Gustavo Pinto, Igor Scaliante Wiese, and Marco A. Gerosa. Almost there: a study on quasi-contributors in open source software projects. In *Proceedings of the 40th International Conference on Software Engineering*, ICSE '18, pages 256–266. Association for Computing Machinery.

[43] Vikram N. Subramanian. An empirical study of the first contributions of developers to open source projects on GitHub. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: Companion Proceedings*, ICSE '20, pages 116–118. Association for Computing Machinery.

[44] Xin Tan. Reducing the workload of the linux kernel maintainers: multiple-committer model. In *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ESEC/FSE 2019, pages 1205–1207. Association for Computing Machinery.

[45] Xin Tan, Minghui Zhou, and Brian Fitzgerald. Scaling open source communities: an empirical study of the linux kernel. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*, ICSE '20, pages 1222–1234. Association for Computing Machinery.

[46] Xin Tan, Minghui Zhou, and Zeyu Sun. A first look at good first issues on GitHub. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ESEC/FSE 2020, pages 398–409. Association for Computing Machinery.

[47] Michael Terry, Matthew Kay, and Ben Lafreniere. Perceptions and practices of usability in the free/open source software (FoSS) community. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '10, pages 999–1008. Association for Computing Machinery.

[48] Bianca Trinkenreich, Mariam Guizani, Igor Wiese, Anita Sarma, and Igor Steinmacher. Hidden figures: Roles and pathways of successful OSS contributors. 4:180:1–180:22.

[49] Jason Tsay, Laura Dabbish, and James Herbsleb. Influence of social and technical factors for evaluating contribution in GitHub. In *Proceedings of the 36th International Conference on Software Engineering*, ICSE 2014, pages 356–366. Association for Computing Machinery.

[50] Jason Tsay, Laura Dabbish, and James Herbsleb. Let's talk about it: evaluating contributions through discussion in GitHub. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, FSE 2014, pages 144–154. Association for Computing Machinery.

[51] Marat Valiev, Bogdan Vasilescu, and James Herbsleb. Ecosystem-level determinants of sustained activity in open-source projects: a case study of the PyPI ecosystem. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ESEC/FSE 2018, pages 644–655. Association for Computing Machinery.

[52] Patrick Wagstrom, Corey Jergensen, and Anita Sarma. Roles in a networked software development ecosystem: A case study in GitHub. page 12.

[53] Zhendong Wang. Assisting the elite-driven open source development through activity data. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ESEC/FSE 2020, pages 1670–1673. Association for Computing Machinery.

[54] Zhendong Wang, Yang Feng, Yi Wang, James A. Jones, and David Redmiles. Unveiling elite developers' activities in open source projects. 29(3):16:1–16:35.

[55] Mairieli Wessel. Enhancing developers’ support on pull requests activities with software bots. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ESEC/FSE 2020, pages 1674–1677. Association for Computing Machinery.

[56] Mairieli Wessel. Leveraging software bots to enhance developers' collaboration in online programming communities. In *Conference Companion Publication of the 2020 on Computer Supported Cooperative Work and Social Computing*, CSCW '20 Companion, pages 183–188. Association for Computing Machinery.

[57] Jin Xu, Yongqin Gao, Scott Christley, and Gregory Madey. A topological analysis of the open source software development community. page 10.

[58] Y. Ye, K. Nakakoji, Y. Yamamoto, and K. Kishida. The co-evolution of systems and communities in free and open source software development.

[59] Minghui Zhou, Qingying Chen, Audris Mockus, and Fengguang Wu. On the scalability of linux kernel maintainers' work. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, ESEC/FSE 2017, pages 27–37. Association for Computing Machinery.

[60] Shurui Zhou, Bogdan Vasilescu, and Christian Kästner. What the fork: a study of inefficient and efficient forking practices in social coding. In *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ESEC/FSE 2019, pages 350–361. Association for Computing Machinery.

[61] Jiaxin Zhu, Minghui Zhou, and Audris Mockus. Effectiveness of code contribution: from patch-based to pull-request-based tools. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, FSE 2016, pages 871–882. Association for Computing Machinery.