NAIST-IS-DD1561008

Doctoral Dissertation

A Human Capital Index for Open Source Software Development

Saya Onoue

March 15, 2018

Graduate School of Information Science
Nara Institute of Science and Technology

A Doctoral Dissertation
submitted to Graduate School of Information Science,
Nara Institute of Science and Technology
in partial fulfillment of the requirements for the degree of
Doctor of ENGINEERING

Saya Onoue

Thesis Committee:

Professor Kenichi Matsumoto	(Supervisor)
Professor Keiichi Yasumoto	(Co-supervisor)
Professor Akito Monden	(Okayama University)
Assistant Professor Hideaki Hata	(Co-supervisor)
Assistant Professor Raula Gaikovina Kula	(Co-supervisor)

A Human Capital Index for Open Source Software Development*

Saya Onoue

Abstract

Software development is a very human-intensive activity. This dissertation proposes a framework for Open Source Software (OSS) that represents human factors as Human Capital. This Human Capital framework is a benchmark to measure the reliability and sustainability of OSS projects.

To propose a framework for OSS Human Capital, first a systematic mapping study was carried out that classified 78 studies into four dimensions: (1) capacity for skill attainment, (2) deployment for a workforce, (3) development for access to learning, and (4) know-how for knowledge sharing. The key outcome is a set of 13 indicators and 12 metrics for constructing a Human Capital Index (HCI).

The dimension of deployment and know-how are studied through the population structure of 90 OSS projects, using a demographic approach. This dissertation proposes software population pyramids, which represent the deployment and know-how dimensions of Human Capital. The second study investigates the characteristics of contributors' activities in OSS development for capacity.

The final study is the construction and evaluation of the HCI framework. This empirical study evaluates the HCI of OSS projects and clarifies the metrics that affect OSS Human Capital. Furthermore, the HCI dimension plot is useful for comparing human activity in OSS projects. This dissertation provides an evidence-based comprehensive framework to help practitioners understand the Human Capital in their projects.

Keywords:

Open Source Software, Human Capital, Software Development Community, Software Ecosystem, Demography

*Doctoral Dissertation, Graduate School of Information Science, Nara Institute of Science and Technology, NAIST-IS-DD1561008, March 15, 2018.

オープンソースソフトウェア開発における人的資本指標の提案*

尾上　紗野

内容梗概

人的資本とは人間が持つ知能や技能などの能力を資本として捉えた概念であり，オープンソースソフトウェア (OSS) 開発は人的資本に頼った活動であるといえる．本論文では，OSS 開発の人的資本の指標を示すフレームワークを提案する．このフレームワークは，OSS プロジェクトの信頼性と継続性を計測することが可能である．OSS 開発の人的資本の枠組みを提案するために，本論文では初めに 78 編の研究を (1) Capacity (2) Deployment (3) Development (4) Know-how の 4 つの要素に分類し，その結果をもとに 13 の尺度を提案し，12 の指標を定義した．本論文では，はじめに人口学的アプローチを用いた Deployment と Know-how の指標の提案を行った．ここでは提案したソフトウェア人口ピラミッドによって 90 の OSS プロジェクトの人口構造を明らかにし，貢献者の獲得・維持にかかる指標と，貢献者の多様性にかかる指標を提案した．次に，Capacity の指標を提案するため，OSS 開発における貢献者の活動の特徴を調査し，貢献者の自己学習にかかる指標を提案した．最後に，提案した OSS プロジェクトの人的資本指標を評価するため，1,418 の OSS プロジェクトの人的資本指標を算出した．人的資本指標の推移から，プロジェクトを Growing，Growing after shrinking，Shrinking after growing，Shrinking の 4 つに分類し，各プロジェクトの人的資本指標の推移分析を通して人的資本に特に影響を与える指標を明らかにした．

キーワード

オープンソースソフトウェア，人的資本，ソフトウェア開発コミュニティ，ソフトウェアエコシステム，人口統計学

*奈良先端科学技術大学院大学 情報科学研究科 博士論文, NAIST-IS-DD1561008, 2018 年 3 月 15 日.

List of Major Publications

Journal paper

1. S. Onoue, H. Hata, A. Monden, and K. Matsumoto, "Investigating and Projecting Population Structures in Open Source Software Projects: A Case Study of Projects in Github," IEICE Transactions on Information and Systems, volume E99-D, number 5, pages 1304–1315, May 2016.

2. 尾上紗野, 畑秀明, 松本健一, "GitHub 上の活動履歴分析による開発者分類," 情報処理学会論文誌, volume 56, number 2, pages 715–719, February 2015.

International Conference

1. S. Onoue, H. Hata, and K. Matsumoto, "A Study of the Characteristics of Developers Activities in Github," In Proc. of 5th International Workshop on Empirical Software Engineering in Practice (IWESEP 2013), pages 7-12, December 2013.

2. S. Onoue, H. Hata, and K. Matsumoto, "Software Population Pyramids: the Current and the Future of Oss Development Communities," In Proc. of 8th International Symposium on Empirical Software Engineering and Measurement (ESEM 2014), number 34, 4 pages, September 2014.

Domestic Conference

1. 尾上紗野, 畑秀明, 松本健一, "原型分析による活動履歴からの OSS 貢献者プロファイリング," 第 22 回ソフトウェア工学の基礎ワークショップ (FOSE2015), pages 41–46, 2015 年 11 月.

2. 尾上紗野, 畑秀明, 門田暁人, 松本健一, "人口ピラミッドによる OSS プロジェクト貢献者の流動性分析," 第 21 回ソフトウェア工学の基礎ワークショップ (FOSE2014), pages 63–68, 2014 年 12 月.

List of Other Publications

International Conference

1. N. Lertwittayatrai, R. G Kula, S. Onoue, H. Hata, A. Rungsawang, P. Leelaprute, and K. Matsumoto, "Extracting Insights from the Topology of the Javascript Package Ecosystem," In Proc. of 24th Asia-Pacific Software Engineering Conference (APSEC2017), pages 298–307, December 2017.

2. K. Nakasai, H. Hata, S. Onoue, and K. Matsumoto, "Analysis of Donations in Eclipse Project," In Proc. of 8th IEEE International Workshop on Empirical Software Engineering in Practice (IWESEP 2017), pages 18-22, March 2017.

3. H. Hata, T. Todo, S. Onoue and K. Mstasumoto, "Characteristics of Sustainable OSS Projects: A Theoretical and Empirical Study" 8th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE 2015), pages 15-21, May 2015.

Domestic Conference

1. 中才恵太朗, 尾上紗野, 畑秀明, 松本健一, "オープンソースソフトウェアにおける寄付の分析," 情報処理学会研究報告, volume 2016-SE-194, number 5, pages 1-6, 2016 年 11 月.

2. 瀧本恵介, 門田暁人, 尾上紗野, 畑秀明, 亀井靖高, "原型分析を用いたソフトウェアバグ分析," 電子情報通信学会技術研究報告, ソフトウェアサイエンス, volume IEICE-116, number 277, pages 91–96, 2016 年 10 月.

3. 坂口英司, 伊原彰紀, 尾上紗野, 畑秀明, 松本健一, "複数のオープンソースプロジェクトに参加する開発者による貢献の分析", 情報処理学会研究報告, 第 92 回グループウェアとネットワークサービス研究会, 第 9 回セキュリティ心理学とトラスト研究会, volume. 2014-GN-92 No.15, volume 2014-SPT-9 No.15, pages 1-4, 2014 年 5 月.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1. Background

Software development is a very human-intensive activity. Like most intangible assets, software development is said to lack physical substance (unlike physical assets such as machinery and buildings) and is therefore usually more difficult to evaluate. Thus, evaluating software development in terms of the actual individuals is not practical because it is more related to skills and knowledge that are created, retained and lost in software development.

By leveraging the concepts of the economic viewpoint of "Human Capital" various dimensions of Human Capital can be closely analyzed and empirically measured [97]. Human Capital can be explained as the capability of individuals to provide solutions (e.g., skills and knowledge). These "skills, knowledge, and similar attributes affect particular human capabilities to do productive work" which in turn can be improved through on-the-job training, formal education and programmed learning. Such capital resides with, and is utilized by individuals. Many previous studies have focused on human factors, including productivity, communication structure, or knowledge loss in software engineering. Such studies approach OSS projects as an ecosystem on their own (citation needed). However, Crowston and Howison concluded that assessing the health of an OSS project is not an easy task [26]. They have not studied OSS health comprehensively and did not discuss the relation of each perspective. In this thesis, I construct an evaluation of OSS projects that combines several perspectives comprehensively.

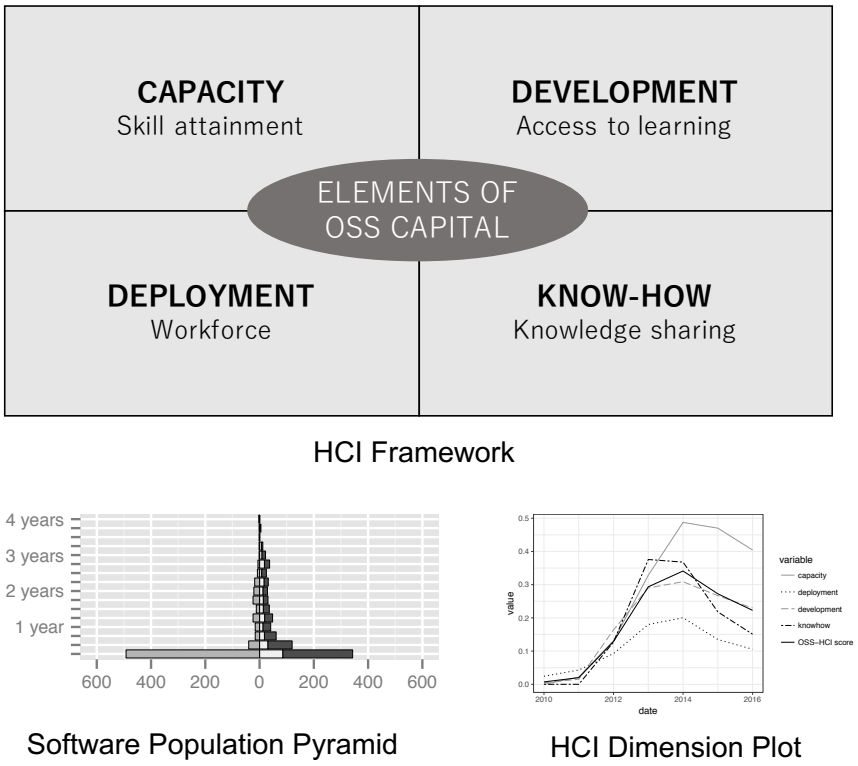


Figure 1.1: Summary of Results

1.2. Main Results

Figure 1.1 shows a summary of the results. This dissertation presents synthetic perspectives to measure Human Capital in OSS. The term, "Human Capital" was proposed by Gray Becker in 1964 [9], who discussed dimensions relevant to knowledge, habits, and social and personality attributes, regardless of the organization.

In this thesis, I discuss an OSS project community and propose four dimensions of such a community as well as a Human Capital Index (HCI) to measure and evaluate human activities. The contributions and results of this thesis are as follows:

1.2.1 Systematic Mapping of the Literature

A key outcome of this mapping study is a set of indicators for constructing a HCI. Four dimensions: capacity, deployment, development, and know-how in HCI are applied to software engineering. A systematic mapping study from five top journal articles and four top international conferences published between 2013 to 2017 are undertaken and 78 studies are identified regarding human aspects of software engineering, which are classified into the four dimensions: capital for skill attainment (capacity), deployment for a workforce, development for access to learning, and know-how for knowledge sharing.

For the dimension of capacity, topics related to (a) *personality*, (b) *success*, and (c) *performance* are discussed. Deployment, the most complex topic, is discussed in relation to (a) *OSS* (b) *community*, (c) *communication* and (d) *productivity*. For a discussion of the dimension of development, (a) *user* and (b) *OSS* are discussed. Finally, the dimension of know-how is discussed as related to practices, especially with GitHub-related research such as pull requests.

In terms of theories, 77% of papers do not report any theories such psychology, game theory ad demography. Experimental research methods are the most common type of research, and case studies and surveys are applied theories popularly.

In terms of data, the results of this thesis show that the dimension of capacity is the least discussed with a low cover of only 3% of papers. Finally, 58% of papers analyze less than 10 projects with data that covers more than five years. Moreover, only half of the papers tend to use multiple sources of data, often combining code and other assets such as mailing lists (ML), bug tracking (BTS) and issue tracking systems (ITS).

1.2.2 Profiling Contributors Based on Activities

This thesis discusses the characteristics of contributors' activities in OSS development. To clarify the characteristics of contributors' activities, GitHub activities used by each contributor to represent the capacity dimensions of OSS Human Capital are explored. The goal is to indicate that active software projects have various kinds of contributors characterized by different types of development activities.

In previous studies, using the Myers-Briggs type indicator (MBTI), Sfetsos et al. investigated the impact of personality types on pair programming. Their results showed that pairs with heterogeneous personalities and temperaments had better communication, pair performance, and pair collaboration-viability [98]. Salleh et al. studied the effects of personality on pair programming using the five-factor model, which characterizes personality in terms of five broad personality traits: openness to experience, conscientiousness, extroversion, agreeableness, and neuroticism [94]. Compared to these personality-based analyses which look at how differences in individuals are rejected in their activities, this chapter investigates how contributors could be characterized based on differences in the patterns of activities extracted from project activity logs.

Contributors were categorized based on measures such as whether they preferred communication by coding or comments, or whether they are specialists or generalists. This result indicates that active software projects have various kinds of contributors characterized by different types of development activities.

1.2.3 Measuring Community Structure

This thesis explores the community structure of OSS projects using a demographic approach. Software population pyramids are proposed that represent the deployment and know-how dimensions of Human Capital. The goal is to understand the community structure of OSS projects based on coding contributors and non-cording contributors.

There are many studies that focus on the social aspects of software development. Bird et al. reported some studies about social collaborations in OSS [13–15]. They reported that developers play a significant social role in email lists [14]. Similarly, Bird at. el. analyzed email addresses in open source software projects to examine the community structure among developers [15]. Bogdan noted the success of an OSS project depends to a large extent on its social aspects [109].

In this study, eight cases from four types of projects are investigated and defined in CCR and NCR. Also, a population in a software community is predicted and then a future population prediction method using a cohort component method in demography to a prediction of OSS future communities are given.

There are four types of population structures in OSS development communities in terms of experiences and contributions. This thesis predicted the future population accurately using a cohort component population projection method which predicts a future population using a survival rate calculated from past populations.

Other authors have also been able to predict a future population of OSS projects. Rastogi et al. presented a framework that characterizes the stability of the community in software maintenance projects using community participation patterns. They modeled the community participation patterns of contributors and forecast future behavior to help plan and support informed decision-making [88]. Loyala et al. proposed a methodology that adapts Lotka-Volterra-based biological models used for describing host-parasite interactions to understand how the population of OSS contributors evolves over time [64].

On the other hand, there are some studies focused on the activity periods of contributors. Steinmacher et al., for example, found that 20% of new contributors become long-term contributors [101]. Zhou and Mockus studied long-term contributors (LTC), and analyzed the behavior of individual participants in Gnome and Mozilla [125]. They reported that future LTCs tend to be more active and show more community-oriented attitudes than do other joiners during their first month.

To the best of my knowledge, this is the first study that applies demography to the field of OSS research. This approach addresses OSS-related problems based on demography will hopefully bring new insights since studying population is novel in OSS research.

1.2.4 OSS Human Capital Index (OSS-HCI)

A HCI is constructed based on a systematic mapping of the literature as shown in Section 1.2.1. The HCI indicators are based on the results of the mapping study. The goal is to determine whether or not the defined indexes provide meaningful and interesting insights into the relationship between the OSS HCI.

Ye et al. examined the structure of Free and Open Source Software (F/OSS) communities and the co-evolution of F/OSS systems and communities. They reported that F/OSS systems and communities generally co-evolve, and they co-

evolve differently depending on the goal of the system and the structure of the community [121]. This study also mentions product evolution and community activities through analyzing the Human Capital in OSS projects. This study can help to understand the co-evolution of OSS systems and communities.

Pinto et al. analyzed the activities of casual contributors [84]. Casual contributors are contributors who do not want to become active members. They describe casual contributors as fostering diversity and collaboration.

This thesis used an empirical study to measure the HCI of OSS projects to classify 1,418 OSS projects. As a result, projects were classified into (a) Growing, (b) Growing after Shrinking, (c) Shrinking after Growing and (d) Shrinking. 42% projects can be categorized as Growing, and 42% as Growing after Shrinking projects. On the other hand, more Growing after Shrinking projects were found compared to Shrinking after Growing projects. A project that might once have been Shrinking will find it hard to be Growing, but even Growing projects can be Shrinking.

The HCI dimension plot can investigate the development of a project. A Growing project, for example, increases in each dimension every year. On the other hand, even if development increases, if a project cannot keep its existing contributors, a project will be Shrinking. This study also used regression analysis to show which metrics affect the OSS-HCI score. As a result, we found that capacity and know-how effect to OSS-HCI score most.

1.3. Overview of this Dissertation

The rest of this dissertation is organized as follows. A mapping study of human factor studies in software engineering is presented in Chapter 2. In this chapter, related papers are identified and classified into the four dimensions of Human Capital.

Chapter 3 presents a study of the characteristics of contributors' activities in OSS development. To clarify the characteristics of contributors' activities, we used the GitHub activity from each contributor to represent the capacity dimensions of Human Capital.

Chapter 4 analyzes the community structure of OSS projects using a demo-

graphic approach and software population pyramids that represent the deployment and know-how dimensions of Human Capital are proposed.

Chapter 5 constructs a HCI. The HCI indicators are based on the results of the mapping study in which the set of indictors are introduced and the metrics are defined for the HCL.

Finally, Chapter 6 concludes this dissertation.

Chapter 2

Human Capital in Software Engineering: A Systematic Mapping Study

2.1. Overview

In this chapter, related papers are identified and classified into four dimensions of Human Capital. A key outcome of this mapping study is a set of indicators for constructing the Human Capital Index (HCI) shown in Chapter 5. First, however, the four dimensions of capacity, deployment, development, and know-how are introduced in HCI. These dimensions are applied to software engineering. Chapter 3 discusses the dimensions of capacity, while Chapter 4 explores the deployment and know-how dimensions. The following research questions guide our study:

RQ_1: *What are the kinds of topics related to the Human Factors discussed in SE?*

RQ_2: *What are the Human Factor theories being adapted in SE?*

RQ_3: *Where are the Human Factor origins of the data in SE?*

To answer these research questions a systematic mapping study is carried out from five top journal articles and four top international conferences published between 2013 to 2017.

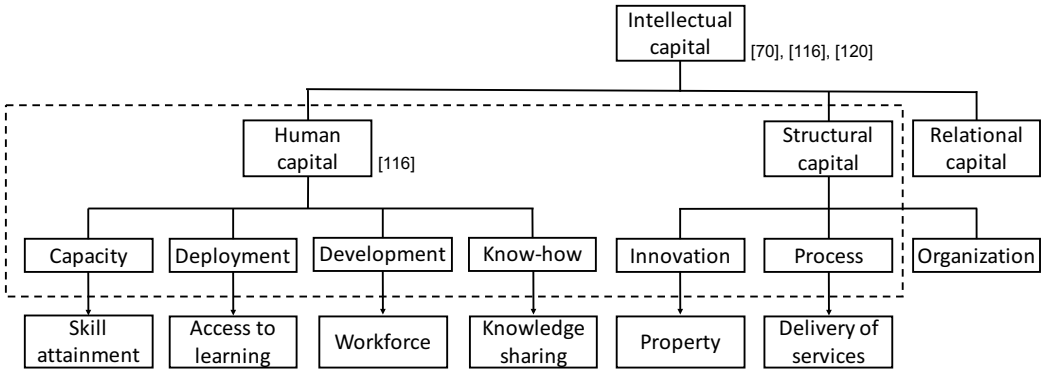


Figure 2.1: Human Capital vs. Structural Capital

2.2. Background

The concept of the Global Human Capital Index (GHCI)[1] as defined by the World Economic Forum[2] is applied to the analysis of an OSS community. This index is termed, OSS Human Capital. The OSS Human Capital proposed in this dissertation is aimed at providing a synthetic assessment of an OSS's Human Capital.

2.2.1 Intellectual Capital in Software Engineering

Intellectual capital is defined as "the sum of all knowledge firms utilize for competitive advantage" [70, 116, 120]. Intellectual capital in economic terminology is the intangible value of a business, covering its people (Human Capital), the value inherent in its relationships (Relational capital), and everything that is left when the employees go home (Structural capital).

Wohlin et. al. states that intellectual capital is the umbrella by which Human Capital is paired with social and organization capital [116]. In other words, software development is hugely dependent on people [30].

In this thesis, the original concepts are revisited, and a more in-depth analysis of Human Capital in terms of its four dimensions is provided. Within structural capital, dimensions of innovation and process are also examined. Complementary

[1] http://reports.weforum.org/global-human-capital-report-2017/
[2] https://www.weforum.org

Table 2.1: Definition of the Human Capital Dimensions for SE

Dimensions	GHCI Definition	Our Definition
Capacity	Measures formal educational attainment.	Measures spontaneous learning attainment.
Deployment	Measures how many people are able to participate actively in the workforce.	Measures how many contributors are able to participate actively in the community.
Development	Knowledge of application and accumulation among the adult population.	Knowledge of application and accumulation among the community of contributors.
Know-how	Breadth and depth of specialized skills used in the workforce.	Breadth and depth of specialized skills (i.e., work practices) used in the community.

to the economic definitions, innovation is used to refer to versions that control silos that contain artifacts such as source code, licenses and anything subject to copyright. Process capital is then defined as process- related artifacts such as bugs and issue-reporting systems (i.e., BTS and ITS), communication and collaboration mechanisms such as mailing lists (i.e., ML) and code review and other tools used during software development.

2.2.2 Human Capital in Software Engineering

There are four thematic dimensions that form the proposed GHCI— *Capacity*, *Deployment*, *Development* and *Know-how*. One of the goals of this thesis is to quantify and understand how human-related studies relate to these dimensions so these dimensions are translated into the SE context.

Capacity A more educated population is better prepared to adapt to new technologies, innovate and compete on a global level. Thus, the Global Human Index (GHI) states that capacity is based on formal education such as primary,

secondary and tertiary levels of attainment. Education includes spontaneous action [63]. Almost all of OSS development field excludes issues of education and relies on the spontaneous learning of contributors. In our definition, we conjecture that the capacity dimension is a measurement of the spontaneous learning attainment by contributors in the community.

Deployment Beyond formal learning, Human Capital is enhanced in the workplace through learning-by-doing, tacit knowledge, exchange with colleagues and formal on-the-job learning. Thus, for GHI, the Deployment dimension measures how many people are able to participate actively in the workforce as well as how successfully particular segments of the population are able to contribute. A country's labor force participation (i.e., employment rates) is the broadest measure of the share of its people participating in the labor market. In our definition, the deployment dimension is understanding community structure or social interaction and a measurement of how many contributors are able to participate actively in a community.

Development The Development dimension concerns the formal education of the next-generation workforce and continued upskilling and reskilling of the current workforce. In our definition, the development dimension is knowing applications and accumulation of knowledge among the community of contributors, through understanding the growth of end-users or the learning curve of new contributors in the community.

Know-how Know-how concerns the breadth and depth of specialized skill use at work. In GHI, the economic complexity is a measure of the degree of sophistication of a country's "productive knowledge" as can be empirically observed in the quality of its export products. In addition, the Index measures the current level availability of high- and mid-skilled opportunities and, in parallel, employer's perceptions of the ease or difficulty of filling vacancies. In the definition, used in this thesis, the know-how dimensions are about breadth and depth of specialized skill (i.e., work practices) use in the community, for example, the proportion of core or peripheral contributors' performance and knowledge loss caused by contributors' leaving the community.

2.3. Method

2.3.1 Research Questions

The study analyzes current trends that arise from the following research questions:

RQ_1: *What are the kinds of topics related to the Human Factors discussed in SE?*
The study investigates what topics and common terminology are often used to describe the different aspects of Human Capital.

RQ_2: *What are the Human Factor theories being adapted in SE?*
This thesis conjectures that there are different kinds of theories being proposed and adapted for Human Capital. Such evidence can improve the understanding of how researchers design their experiments and interpret Human Capital.

RQ_3: *Where are the Human Factor origins of the data in SE?*
This thesis seeks to understand the types of assets (i.e., structural capital) used by researchers to uncover evidence of Human Capital. In detail, data characteristics such as data collection sizes, duration and diversity are extracted.

2.3.2 Review Method

In this section, the different steps executed in our mapping study are explained. A systematic mapping study is a type of systematic literature review. A systematic literature review is a repeatable method for identifying relevant studies to answer specific research questions [66]. In particular, a systematic mapping study is designed to give an overview of a research area through classification and counting contributions in relation to the categories of that classification [55, 81, 82]. Kitchenham et al. [57] contrasted the different characteristics of the process of systematic literature reviews and mapping studies. A systematic mapping study follows the same principled process as systematic literature reviews, though there are different criteria for inclusions/exclusions and quality [115]. Systematic mapping studies do not have strict roles compared to systematic literature reviews,

however, various types of papers should be reviewed for understanding target topic area widely.

First, the method for systematic mapping study is introduced based on the following characteristics [55].

- (C_1) a defined search strategy
- (C_2) a defined search string, based on a list of synonyms combined by ANDs and ORs
- (C_3) a broad collection of search sources
- (C_4) a strict documentation of the search
- (C_5) quantitative and qualitative papers should be analyzed separately
- (C_6) explicit inclusion and exclusion criteria
- (C_7) paper selection should be checked by two researchers

Figure 2.2 presents an overview of the mapping study design, which follows (C_1) a defined search strategy. In detail, the method is comprised of two parts, the initial phase and the refinement phase. A total of five steps are followed in this method. Note that Steps 4 and 5 follow two branches, dividing the papers into method (A-4 and A-5) and survey and meta studies (B-4 and B-5).

Initial Phase

Step 1: Collection of Papers

Table 2.2 shows all 10 search sources and their impact factor (IF) or conference ranking for our mapping study. A conference ranking is referred to as the CORE Conference Ranking[3]. To ensure a high quality of papers and to understand the state–of–the–art in the field, this thesis searched for papers in the top journals and five conferences from the software engineering domain. To reduce its selection bias, a range of digital resources, represented as (C_4) were selected from a broad

[3]http://www.core.edu.au

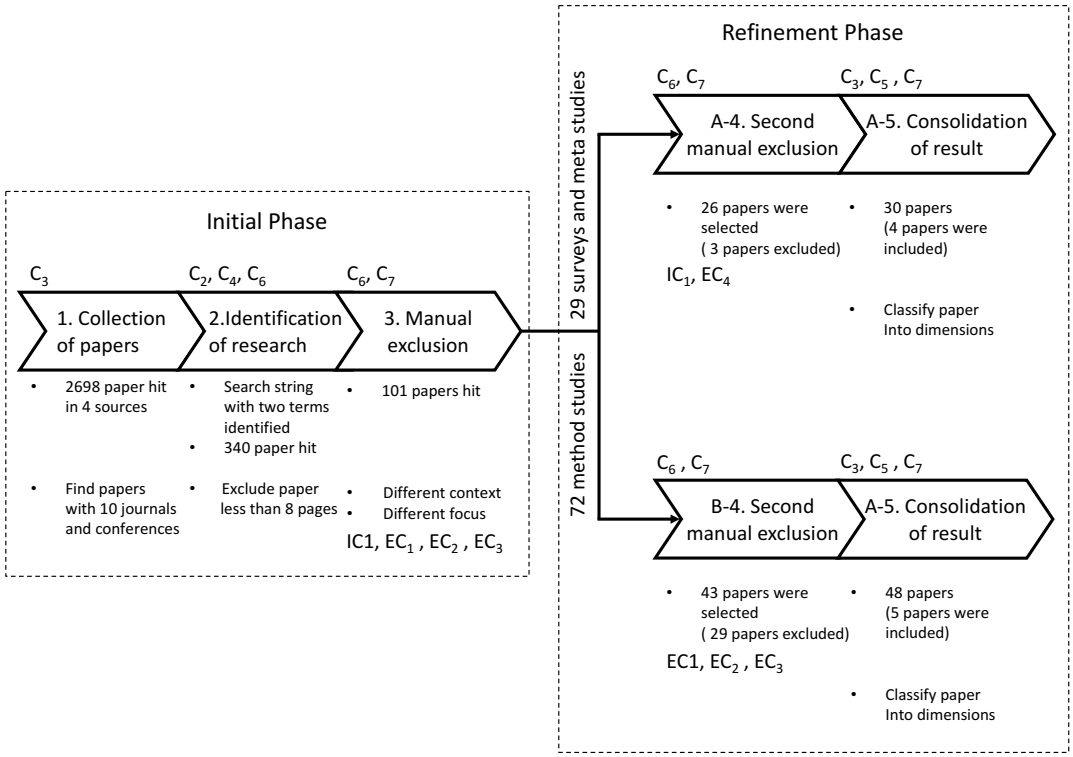


Figure 2.2: Overview of the Mapping Study

selection of search sources.; ACM Digital Library[4], IEEE Xplore[5], Science Direct[6], Springerlink[7].

As shown in Figure 2.2, 2,698 papers were extracted from the four search sources which map to the Top ten publication venues for Software Engineering. Additionally, only technical papers were included, hence filtering out short papers, editorials, tutorials, panels, poster sessions and prefaces and opinions (i.e., papers were automatically filtered out that were shorter than eight pages). Since the intention is to understand the current trends of Human Capital research, papers were collected that were published in the last five years (i.e., 2013 $\sim$ 2017).

Step 2: Identification of Research

[4] `https://dl.acm.org/`

[5] `http://ieeexplore.ieee.org/Xplore/home.jsp`

[6] `http://www.sciencedirect.com/`

[7] `https://link.springer.com/`

Table 2.2: Targeted SE Journals and Conferences

Journal	(TSE) IEEE Transaction on Software Engineering	IF: 2.63
	(ASEJ) Automated Software Engineering Journal	IF: 3.27
	(EMSE) Empirical Software Engineering	IF: 3.28
	(TOSEM) ACM Transactions on Software Engineering and Methodology	IF: 2.87
	(IST) Information and Software Technology	IF: 2.69
Conference	(ICSE) International Conference On Software Engineering	Rank: A*
	(MSR) Working Conference on Mining Software Repositories	Rank: A
	(ICSME) International Conference on Software Maintenance and Evolution	Rank: A
	(ESEC/FSE) Join Meeting of European Software Engineering Conference and Symposium on Foundation of Software Engineering	Rank: A*

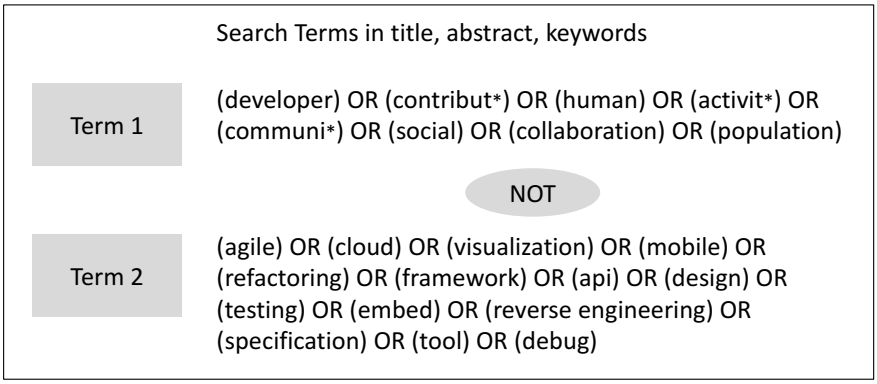


Figure 2.3: Defined Search String

To provide a comprehensive picture of recent research related to Human Capital, (C_2) a defined search string, was used to identify the research area. In this step, the first automated (C_6) explicit inclusion and exclusion criteria were conducted based on the search results.

Figure 2.3 shows the two terms (Term 1 and Term 2) used in the search string. To understand Human-related areas in SE, Term 1 was formulated to include *developer* OR *human* OR *social* OR *collaboration* OR *population*. To capture synonyms and other extensions, as shown in Table 2.3, the keywords *contribu**, *activit** and *communi** were stemmed and used to expand the search space. Since, this thesis is interested in more generic Human-related research from the SE domain, our search string also includes an exclusion list (i.e., Term 2). Words in Term 2 were excluded to avoid papers that are specific to a particular research

Table 2.3: Synonyms of Keywords Used in the Search.

Base Term	Synonyms
contribut*	contributor, contribution
activit*	activity, activities
communi*	community, communities, communication

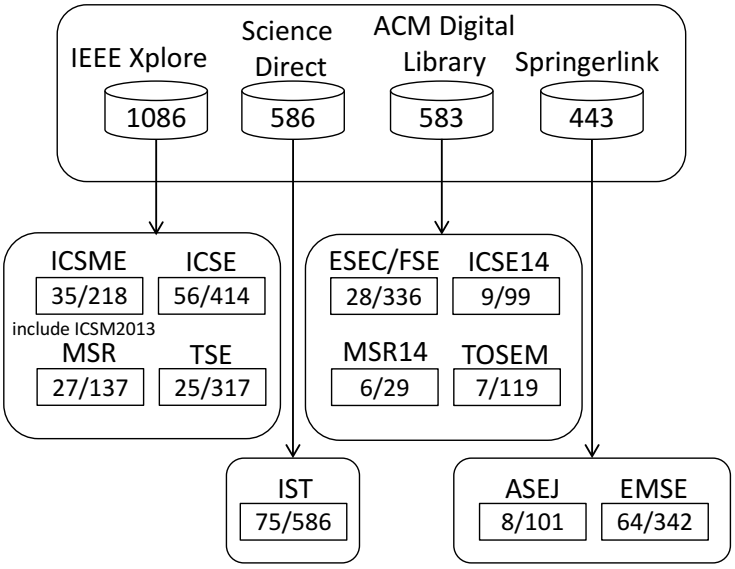


Figure 2.4: A Strict Documentation of the Search (C_4)

topic in SE. Hence, specific topics such as agile, cloud and so on were excluded. The search string was applied to the title, abstract and keywords sections of the papers.

As shown in Figure 2.2, 340 papers were collected after our automatic search execution. For ICSE and MSR conferences, special editions of ICSE14 and MSR14 were only published in the ACM Digital Library and not the Xplore. Figure 2.4 shows the details of the remaining papers from the original 2,698 papers collected in Step 1.

Step 3: Manual exclusion

To complete the initial phase, Step 3 involves the manual exclusions of the collect 340 papers. This step involves an (C_6) explicit inclusion and exclusion criteria

to remove papers that have a different context and focus of this dissertation's research area.

For the manual exclusion, only our inclusion and exclusion criteria were applied to the abstract of each paper.

Inclusion criteria Only a single inclusion criterion is defined, namely, that (IC_1)- the articles should focus on software developers or contributors.

Exclusion criteria Three exclusion criteria were defined that cover the dataset, purpose and the evaluation of the study:

- (EC_1) the analyzed subjects do not use contributors' activity data
- (EC_2) the purpose is not to collect activity data
- (EC_3) the purpose is not an evaluation of a proposed method

To reduce bias following (C_7), the paper selection was checked by two researchers. As a result of Step 3, the collected 340 papers were reduced to 101 papers.

Refinement Phase

The refinement phase includes a separation between (C_5) quantitative and qualitative papers. As a result, the two steps are divided into two branches (A and B).

Step 4: Second manual exclusion Similar to Step 3, the same inclusion and exclusion criteria are used (i.e., IC_1, EC_1, EC_2 and EC_3) for method papers in Step B-4. However, Step 4 includes a full manual reading of all contents of the paper. Finally, two criteria have been added to Step B-4:

- (EC_4) the paper is not product-focused
- (EC_5) the paper does not propose a human recommendation technique

For the survey papers (i.e., Step 4-A), extra criteria were added for exclusion based on the content (EC_6) of the paper, which is out of the research scope. In addition, another researcher was added (i.e., making the total number of reviewers three people). As a result of Step 3, the initial 101 papers could be reduced to 69 papers (i.e., 26 surveys and 43 method papers).

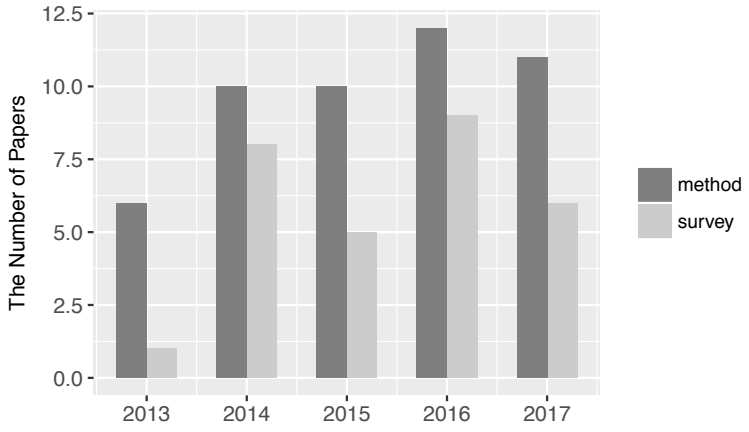


Figure 2.5: The Number of Papers in this Field every Year

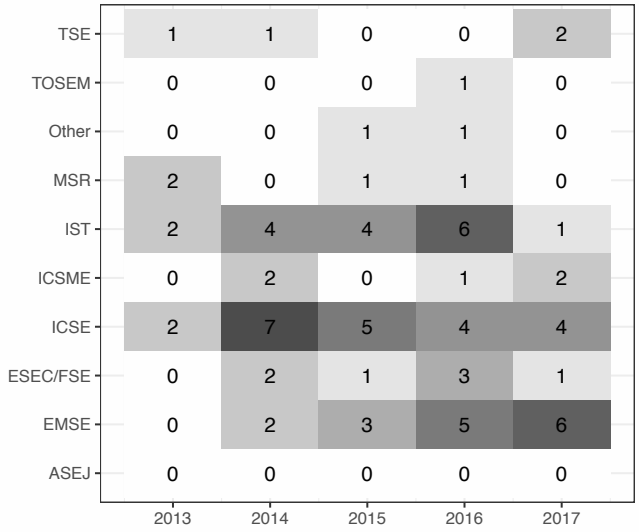


Figure 2.6: The Number of Papers Published every Year

Step 5: Consolidation of results In this step, three steps are performed. As shown in Figure 2.2, after manual discussions, first, some papers (i.e., four survey and five method papers) were included that were excluded in the initial phase, bringing our consolidated papers to 78 in total. Figures 2.5 and 2.6 show some of the trends of the Human-related research.

Figure 2.5 shows the number of papers identified within the years 2013–2017 to each research type (survey and method). This suggests that method papers show a gradual increase year by year. Figure 2.6 shows a heat map of the number of each paper's publications in each conference or journal by year. This suggests that many Human Capital papers are submitted to IST, ICSE and FSE.

Next, three researchers classified all sorted papers into the four Human Capital dimensions. To classify, after reading the title and abstract, a consensus was reached by reviewers on which dimension best suits the paper. The classification is non-exclusive. From these papers, key indicators were then derived that are consistent during the classification process. Below our classification rationales are listed and borrowed from Table 2.1:

- **Capacity** - Papers should discuss the formal attainments of contributors
- **Deployment** - Papers should discuss contributor participation.
- **Development** - Papers should discuss known application and accumulation among contributors.
- **Know-how** - Papers should discuss the breadth and depth of specialized contributor knowledge use in a software project.

Table 2.4 shows the results of the classifications of selected papers into the four dimensions. Results indicate that much research has been carried out on the deployment and know-how dimensions.

To address each of the research questions, the following approach was applied to analyze the consolidated papers. Each research question identifies characteristics within the four dimensions, with the end goal a mapping from each viewpoint.

Approach to answer RQ_1. Topics are extracted from the title, abstract, and keywords in the primary study and co-occurrences analyzed to answer the first

Table 2.4: Summary of the Consolidated Papers (i.e., method and survey) by Dimension

Dimension	method	survey	# of papers
Capacity	S09 [3], S16 [62], S18 [10], S26 [85], S39 [117], S74 [5]	S47 [100], S65 [60]	8
Deployment	S01 [37], S02 [78], S08 [76], S12 [118], S14 [43], S17 [42], S19 [22], S27 [50], S29 [79], S34 [75], S42 [35], S43 [27], S44 [68], S50 [29], S53 [47], S59 [58], S66 [53], S67 [113], S69 [36], S70 [49], S71 [95], S76 [110], S78 [106]	S07 [96], S24 [23], S28 [32], S36 [46] S38 [33], S45 [93], S48 [71], S63 [103],	31
Development	S22 [102], S31 [99], S33 [34], S49 [54], S55 [18], S72 [122],	S03 [6], S11 [114], S46 [87], S56 [59], S57 [17], S68 [1],	12
Know-how	S05 [39], S10 [31], S15 [48], S20 [12], S21 [21], S23 [86], S30 [65], S32 [107], S35 [108], S40 [92], S50 [29], S54 [61], S60 [24], S64 [4], S73 [8], S75 [11], S74 [5]	S04 [16], S06 [112], S13 [72], S25 [7], S37 [51], S41 [111], S51 [69], S52 [83], S58 [123], S61 [40], S62 [41], S77 [77],	29
Total			78 (two repetitions)

*Underlines show repetitions.

research question. In this research, an N-gram Weighting Scheme tool is used[8]. This tool uses enhanced suffix array [2] to enumerate valid N-grams. The output after applying the N-gram IDF tool to the pre-processed data is an N-gram dictionary, which is a list of all valid N-gram key terms. Similar to Terdchanakul et al., [105], we use the N-gram technique to formulate the most frequent keywords used in our corpus (i.e., title, abstract, keywords). Results will be a generated network of topics, with the edges representing the co-occurrence score. These generated keywords will provide insights into the common topics discussed in each Human Capital dimension.

Approach to answer RQ_2. The different types of theories to answer research question two are analyzed. In this analysis, it is important to investigate how much theory is used for the four dimensions. It is also important to find out what kinds of theories are proposed as well as the types of research design used in the studies.

Approach to answer RQ_3. The kinds of data sources are analyzed to answer the last research question. In this analysis, the types of data (code, assets) and their origins (i.e., GitHub, Jazz, Bug Tracking System (BTS) or Mailing List (ML)) for the four Human Capital dimensions are found. I then take a closer look at the collected dataset, analyzing the size of the projects, the period of data collection, whether or not the projects are open source or closed and if the study used single or multiple data sources.

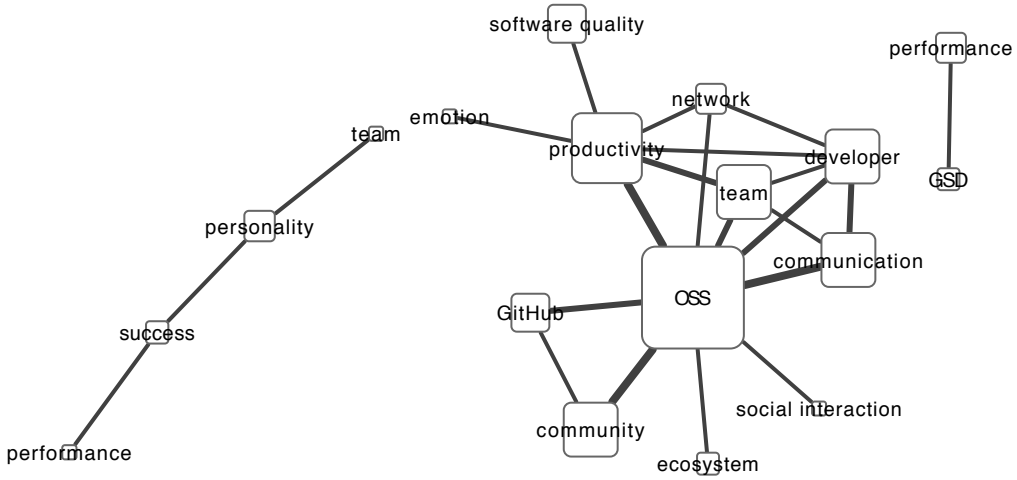
2.4. Results

In this section, the results of the mapping study in terms of the topics (i.e., RQ_1), theory (i.e., RQ_2) and data (i.e., RQ_3) in Human Capital are discussed.

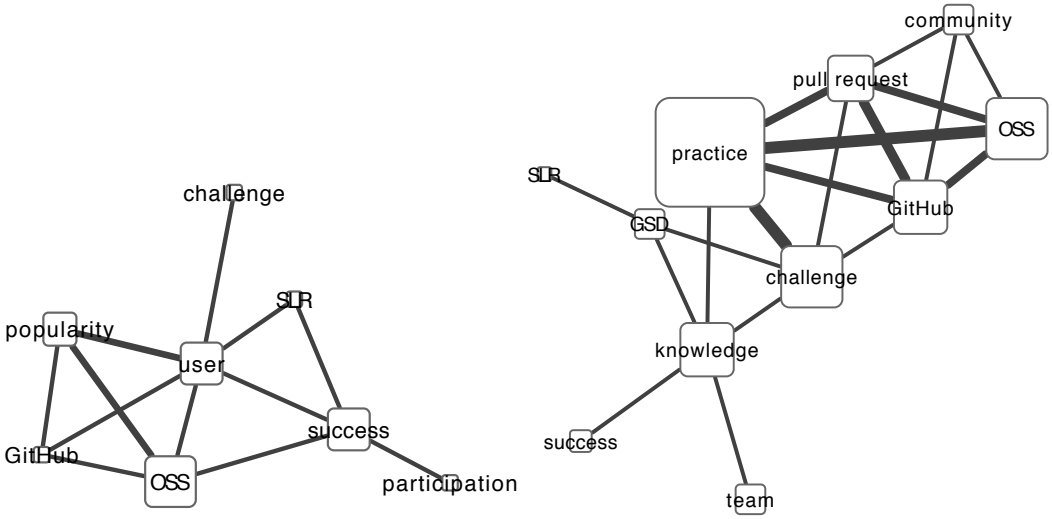
2.4.1 Topics Discussed in Human Capital

Figure 2.7 shows the difference in the network of topics generated for each dimension of the consolidated papers. The size of the network is influenced by either the number of consolidated papers within the dimension and the diversity

[8]https://github.com/iwnsew/ngweight



(a) capacity (b) deployment



(c) development (d) know-how

Figure 2.7: Generated Network of Topics for each Dimension (a) capacity, (b) deployment, (c) development and (d) know-how.

Table 2.5: Top 3 Topics of Papers (by highest co-occurrence score). The Common Topics Are Highlighted by Dimension.

Dimension	Top 3 topics discussed (by ↔ score)
Capacity	personality ↔ team
	personality ↔ success
	performance ↔ success
Deployment	OSS ↔ community
	OSS ↔ communication
	OSS ↔ productivity
Development	popularity ↔ user
	popularity ↔ OSS
Deployment	practice ↔ challenge
	practice ↔ OSS
	pull request ↔ GitHub

of topics used in the field.

Complementary, Table 2.5 shows the main topics (i.e., top 3) topics for the consolidated papers. For capacity, the topics relate to *personality, team, success and performance*. This result indicates that capacity is a more a team-based factor that is linked to *success* and *performance*. In terms of deployment, the networks show that the topics have complex interactions, indicating that these common topics are discussed across the consolidated papers. The four common topics discussed are *OSS, community, communication and productivity*. Finally, much of the know-how topics are strongly related to *practices*. Also, the GitHub-related research is popular, with links to pull-requests. Overall, the extracted topic keywords correspond to the definitions of Human Capital in SE (see Table 2.1), thus providing a validation to this work.

2.4.2 Theories Used in Human Capital

Out of the 78 consolidated papers, only 18 papers described a theory other than grounded theory in their study. This leaves 77% of the papers not reporting any theory. From Table 2.6, most papers' theories were used for the capacity

Table 2.6: Coverage of Selected Papers with a Theory

Dimension	Coverage of Papers	Paper id
Capacity	75%	S09, S16, S18, S26, S39, S65
Deployment	16%	S01,S14, S34, S67, S71
Development	0%	
Know-how	24%	S06, S23, S32, S40, S52, S54, S58

Table 2.7: The Types of Theory Proposed by Our Selected Papers

Proposed Theory	Paper id
Psychology/Psycholinguistics	S01, S06, S09, S16, S18, S39, S54, S65, S71
Game Theory	S14, S67
Group Dynamics	S52, S67
Organization Theory	S52, S71
Demography	S34
Food Web (Ecology)	S23
Financial Risk Management	S40
Information Field Theory	S26
Knowledge-based theory of the firm	S58
Signalling Theory	S32

*Underlines show repetitions.

dimension. Interestingly, none of the papers classified from the development dimension reported a theory usage.

Complementary, Table 2.7 shows that most (i.e., 9) papers had adapted theory from the psychology/psycholinguistics field. For instance, there were several papers that borrowed concepts from (i.e., S01 - Creation of model for team leader role with personality types and gender classification, S06 - The success factors in GSD by questionnaire to expert and S09 - Experiment of personality factors and group processes) various aspects of the human psychology.

Figure 2.8 shows the types of research methods used in complementary to each dimension. In this result, the experimental research method is the most common

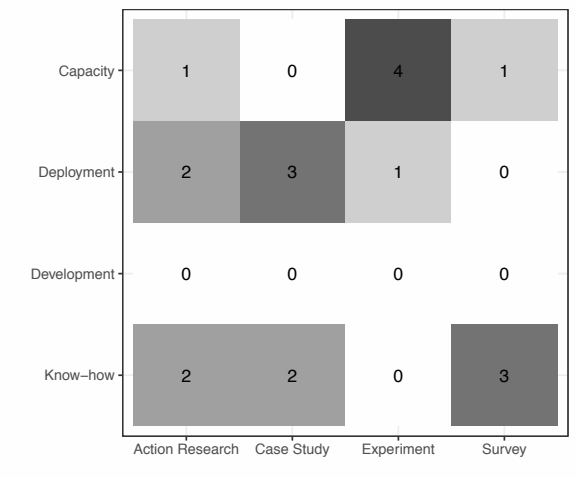


Figure 2.8: Heatmap Showing the Research Methods Used per Dimension for Papers with a Theory.

type of research method among the consolidated papers, and it is also used in the capacity papers. In addition, in action research, case study and surveys are popularly used with their theories. Action research represents research methods that research either initiated to solve an immediate problem or to improve the way issues are addressed and problems solved [104].

2.4.3 Sources of Human Capital

Table 2.8 shows a breakdown of the data origins for the 36 consolidated papers (i.e., excluded survey papers). In this result, human capacity dimension papers have the least source of data (i.e., coverage of 3%). From the table, it can be seen that researchers used assets such as mailing lists (ML), bug tracking (BTS) and issue tracking systems (ITS) to complement their code in their datasets (i.e., deployment = seven papers used code, while 17 papers used assets, development = two papers used code, while five papers used assets, know-how = six papers used code, while 12 papers used assets). Additionally, most data originate from not only code but also from software assets. Note that in Table 2.8, *others* refers to asset sources such as closed company data, stack overflow, code review, web documents, gerrit code review and Ruby API systems.

Table 2.9 shows a detailed analysis of the data collected statistics (i.e., size,

Table 2.8: Summary of the Data Coverage by Dimension.

Dimension	Coverage	Type	Origin	Paper ID
Capacity	3%	Innovation + Process Capital	Jazz	S16
Deployment	53%	Innovation Capital	VCS	S12, S27, S42, S66, S69, S70, S76
		Innovation + Process Capital	GitHub	S02, S14, S34, S53, S71
		Process Capital	ML/Chat log	S12, S17, S19, S29, S42, S64, S66, S67, S69
			BTS/ITS	S08, S19, S29, S69
			Other	S19, S43, S02, S78
Development	11%	Innovation Capital	VCS	S31, S49
		Innovation + Process Capital	GitHub	S55
		Process Capital	ML/Chat log	S33
			BTS/ITS	S72
			Other	S22, S72
Know-how	33%	Innovation Capital	VCS	S15, S20, S21, S23, S40, S64, S75
		Innovation + Process Capital	GitHub	S05, S21, S30, S32, S35
			Jazz	S54
		Process Capital	ML/Chat log	S15, S64, S75
			BTS/ITS	S20, S23, S32, S64, S73
			Other	S10, S73, S75

*Underlines show repetitions.

Table 2.9: Summary of Data Collection of Sources

		Capacity	Deployment	Development	Know-how	Coverage (%)
Projects	S (> 10)	1	9	3	8	58%
	M ($11 \sim 100$)	0	7	0	1	22%
	L ($101 \sim$)	0	3	1	3	19%
Period	S ($< 1year$)	0	1	2	3	16%
	M ($< 5year$)	1	5	0	2	22%
	L ($5\ year \sim$)	0	8	2	3	36%
	**unspecified	0	5	0	4	25%
Origin	Company	1	2	1	2	16%
	OSS	0	17	3	10	83%
Source	Multi	0	7	1	7	42%
	Single	1	12	3	5	58%

period, data origin and sources) for our 36 method papers. Note that some papers were excluded from this analysis due to their difficulty of classification (i.e., S31, S33, S40, S75). 58% of papers analyzed less than 10 projects in their study. In fact, five of these papers only used a single project for analysis (i.e., S10 or S16). On the other hand, one paper (i.e., S21) used 58,092 projects in their study. Evidence then suggests that many of the papers collected data that ranged more than five years, with six papers having data ranging less than a year. Another interesting finding is that the papers tend to use OSS projects (83%) compared to closed company data. In regards to the sources, about half of the papers tend to use multiple sources of data, often combining code and other assets (i.e., as shown in Table 2.8) in their studies.

2.4.4 Human Capital Index (HCI)

In this section the implications of this study are discussed. A key outcome of this study is a set of indicators for the different dimensions of Human Capital. Hence, the set of indicators are introduced first. Next, in this section, the strengths and weaknesses of the study with threats to validity are discussed.

Table 2.10 describes the proposed indicators and the mapping to their respective inspired consolidated papers. Here, the rationale and definition of each indicator by dimension is explored.

Table 2.10: Proposed Human Capital Indicators Mapped to the Consolidated Papers (i.e., by method, experiment (exp.) and survey).

	Indicator	**Paper Id**		
		method	**exp.**	**survey**
Capacity	Individual Contribute activity profiling	S16	S09, S18, S26, S39, S74	S47, S65
Deployment	Community Structural Complexity	S14, S27, S34	S59	S07, S36,
	Core vs. Peripheral Developer workload equality	S76		
	Contributor Participation rates	S53, S78		S38, S63
	Productivity rates	S08, S12, S19, S27, S42, S43, S66, S71	S44	S24
	Community Social Interaction	S02, S08, S12, S17, S42, S19, S29, S66, S67, S69, S70, S71	S50,	S45, S48, S63
	Community Diversity		S01,	S28, S38
Development	End-user participation	S22, S33, S55, S72		S03, S46, S68, S57
	Developer Learning-curve rate	S31, S49		S11, S56
Know-how	Maturity of Work Practices (Documentation & execution of work practices)	S05, S10, S23, S30, S32, S35, S60, S64, S73, S75	S50, S74	S04, S06, S13, S25, S37, S41, S51, S52, S58, S61, S62, S77
	Core vs. Peripheral knowledge	S15, S20, S54		
	Knowledge loss rates	S40		
	Onboarding rates	S21		

*Underlines show repetitions.

Capacity

1. **Individual Contribute activity profiling** - This indicator is a rating for an individual score for a contributor. Examples could include a study about measuring team personality and climate (i.e., S19). This study measures neuroticism, extroversion, conscientiousness and all that of developers by experiment. Another study is related to personality profiles of developers (i.e., S16). This study analyzes message exchanges or developers' tasks using code and assets data source.

Deployment

1. **Community Structural Complexity** - This indicator rates the complexity of the community of contributors and how they actively participate. An example is studying about population structure in OSS projects (i.e., S34). In detail, this study creates a population pyramid by the contributors' activity period in OSS projects. Another example is a study about the impacts of organizational factors on software quality (i.e., S59), where authors conduct observations of an in-house software development project within a large telecommunications company.
2. **Core Developer vs. Peripheral Developer workload equality** - These rates are in regard to the work activities of developers in the community. Example studies include observation of the variation and specialization of workload in an ecosystem community to identify developers' activity types and comparing the number of files that developers modify (i.e., S76).
3. **Contributor Participation rates** - This indicator describes contributors participation in a particular activity such as code review (i.e., S53, S78). S53 investigates the phenomena of inactive code review contributors from activities such as pull requests, while S78 is a study about review participation in code review, introducing several metrics such as purpose, history, and prior activity of reviewers and patch authors.
4. **Productivity rates** - This indicator measures the productivity of contributors. For instance, study S24 evaluates developer performance for software-intensive products. This study interviews managers to understand from a

managerial perspective how they engage in software product development activities, and to evaluate performance in large organizations. On the other hand, study S44 is about sensing developers' emotions, progress and the use of biometric measures.

5. **Community Social Iteration** - The social interaction indicator is a measure of contributor collaboration and communication within the community. Examples include a study about communication in open source software development mailing lists (i.e., S17) and a study about identification of contributors' collaborations from different sources (i.e., S29) by analyzing source code co-changes.
6. **Community Diversity** - This indicator measures the diversity within the community. For example, S01 studies team leadership roles with personality types and gender classification. It uses experimental data to develop a model for software development team composition by keeping gender as a major effecting variable with personality. Furthermore, study S38 identifies barriers for female participation on stack overflow. It interviews female contributors about contribution barriers in online communities.

Development

1. **End-user participation** - This indicator is user centric, measuring from a user's perspective of skills development. For instance, there are some studies that involve discovering how end-user programmers and their communities use public repositories (i.e., S22). This study analyzes end-user programmer communities, the characteristics of artifacts in community repositories, and how authors evolve over time. More recently, there is work that investigates factors that impact the popularity of GitHub repositories (i.e., S55). This work analyzes stars awarded to GitHub projects and identifies the popularity growth of these repositories.
2. **Developer Learning-curve rate** - This indicator is developer centric, measuring from a developer's perspective of skills development. For example, study S31 investigates the effect of the Google summer of code. This study compares developers' activity in OSS projects to before and after participation the Google summer of code event. Another study conducts

a questionnaire to investigate the impressions, motivations, and barriers of one time code contributors to FLOSS projects (i.e., S56).

Know-how

1. **Maturity of Work Practice** - This indicator measures the degree of work practices used in contributors' software development processes. For instance, study S05 is concerned with a pull-based software development model. It explores how pull-based software development works by analyzing pull requests and comments history. On the other hand, study S10 explores the prior beliefs of developers at Microsoft, confirming beliefs to actual empirical data. It is a survey to understand a priori opinions on issues such as cost, quality, and interval related to the project.
2. **Core vs. Peripheral knowledge** - This indicator explores the knowledge of core vs. peripheral developers. For instance, study S15 classifies developers into core and peripheral by count and network metrics. This study measures metrics related to commits and emails within the project. Another example is a study about determining developers' expertise and roles (i.e., S20). This study analyzes bug tracker and source code repository to characterize developers.
3. **Knowledge loss rates** - This indicator investigates the loss of knowledge by a contributor leaving the community. For instance, study S40 quantifies and investigates how to mitigate turnover-induced knowledge loss. In detail, it quantifies the extent of abandoned source files using source code history and assesses knowledge loss by turnover.
4. **Onboarding rates** - This indicator measures the retention of contributors to a project. In this thesis, we identified a study that explores a precursor to joining a project. The study analyzes the technical factors of past experience and social factors of past connections to understand onboarding in software projects.

2.5. Summary

In this chapter, related papers were identified and classified into four dimensions of Human Capital. 78 studies regarding human aspects in software engineering were classified into four dimensions: capital for skill attainment, deployment for a workforce, development for access to learning, and know-how for knowledge sharing.

RQ_1: What are the kinds of topics related to the Human Factors discussed in SE?

For the capacity dimension, topics related to (a) *personality*, (b) *success*, and (c) *performance* were the most discussed. In terms of the deployment of the most complex topics, these were discussed in (a) *OSS* (b) *community*, (c) *communication* and (d) *productivity*. In terms of development, the most of topics were discussed (a) *user* and (b) *OSS*. Finally, know-how topics were related to *practices*, especially with GitHub related research such as pull requests.

RQ_2: What are the Human Factor theories being adapted in SE?

77% of papers did not report any theory. Also, the experimental research method was the most common type of research, with case studies and surveys popularly used with their theories.

RQ_3: Where are the Human Factor origins of the data in SE?

The results showed that papers in the capacity dimension have less sources of papers (i.e., coverage of 3%). We also show that 58% of papers analyze less than 10 projects with data ranging more than five years. Finally, half the of papers tend to use multiple sources of data, often combining code and other assets.

A key outcome of this mapping study is a set of indicators for constructing HCI.

Chapter 3

Profiling Contributors Based on Activities

3.1. Overview

In this chapter a study of the characteristics of contributors' activities in OSS development is presented. To clarify the characteristics of contributors' activities, the GitHub activity by each contributor is used to clarify the capacity dimensions of the Human Capital discussed in Chapter 2.

In this study, a case study of the OSS contributors is conducted. The goal is to indicate that active software projects have various kinds of developers characterized by different types of development activities.

19 contributors' activities from two projects were investigated. In detail, we analyze GitHub contributor events of (a) push, (b) pull request, (c) issues, (d) create, (e) delete, (f) pull request comment, (g) commit comment, and (h) issues comments.

3.2. Background

When trying to identify good contributors and understand how contributors set goals in life, the different types of contributors are an interesting topic for investigation. For example, the blog article "Are You a Good Programmer?,"[1] describes four types of good programmers.

This article suggests that the four types of programmers approach code in different ways based on their motivation. **The Philosopher**, driven by a need for safety and security, writes tightly controlled code. **The Inventor**, driven to explore, creates quirky and unique code. **The Conqueror**, driven to compete, looks for harder challenges. Finally, **The Problem Solver**, motivated to create value, tries to deliver the desired outcome.

Similarly, the article "The 6 Types of Software Engineers: Identification, Care and Feeding[2]," introduces the following *six* types of software engineers: **The Veteran**, **The Hotshot** (smart and young engineer), **The Great One** (always delivers on schedule, writes solid code, and so on), **The Teflon-gineer** (will do anything to reduce his work), **Offshore**, **The Maverick** (smart, creative, dependable, does not want to build on or maintain the existing codebase). Then the article "10 Types of Programmers You'll Encounter in the Field[3]:" presents 10 types: **Gandalf** (as adept at working magic as Gandalf), **The Martyr** (a workaholic), **Fanboy**, **Vince Neil**, **The Ninja** (the team's MVP, and no one knows it), **The Theoretician**, **The Code Cowboy**, **The Paratrooper**, **Mediocre Man**, and **The Evangelist**.

These lists of contributor types are neither complete nor comprehensive, nor do contributors have to fit these categories. The point is that there are many different types of contributors, with many different ways of making valuable contributions. This research studied types of contributors based on actual contributors' activities. We did not investigate the social structure in software projects, nor did we analyze the roles of contributors in such projects. Instead, we tried to

[1] `http://techiferous.com/2011/08/are-you-a-good-programmer/`

[2] `http://crankypm.com/2008/08/the-6-types-of-software-engineers-`identification-care-and-feeding/

[3] `http://www.techrepublic.com/blog/10-things/10-types-of-programmers-`youll-encounter-in-the-field/

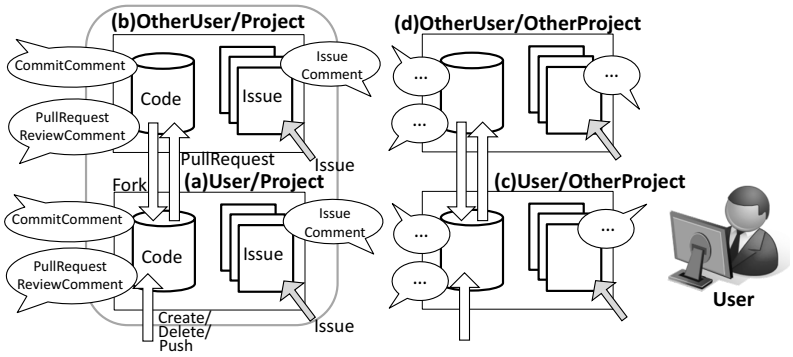


Figure 3.1: Contributor's Activities in GitHub.

characterize contributors based on their observed activities.

As case studies for this research, we chose two active software projects, node and jQuery. Both of these use GitHub[4], which is a widely-used hosting service for software development projects that used the Git revision control system. GitHub provides "social coding" services for contributors to collaborate with each other. GitHub provides APIs (GitHub API v3[5]) which allowed us to collect contributors' activity data.

Our study collected data from the two projects, which we then analyzed to investigate the characteristics of contributors' activities. We analyzed the frequencies of types of activities, such as code-related, comments-related, and issue handling. We also investigated the specialization of the projects and the rates of the activities. Our study showed that the top contributors in these projects include different kinds of contributors with different characteristics of development activities.

Figure 3.1 provides an overview of a contributor's activities in GitHub. A project is identified by the owner and the name of the project (`owner/project_name`). In GitHub, a contributor can work on various projects. Also, in our study, we consider (a) and (b) projects as target projects and (c) and (d) as other projects. Therefore, we classify contributor's activities in terms of target projects and other projects. This means that for a contributor `User` who is a contributor to one specific project `Project` , there are four categories of working projects:

[4] `https://github.com/`

[5] `http://developer.github.com/`

(a) Target project `TargetProject` belonging to the contributor `User`. This code repository may have been forked from another user's repository.

(b) Target project `TargetProject` belonging to another contributor`OtherUser`, who may be the owner of the project.

(c) Non-target projects `OtherProject` belonging to the contributor `User`. These code repositories have been forked from the original repositories.

(d) Non-target projects `OtherProject` belonging to another contributor `OtherUser`.

Each project has a code repository and an issue repository. Although a contributor can directly access their own code repositories, they cannot directly push their changes to code repositories belonging to another contributors' account. To work with code from another contributor's code repositories, a contributor can make cloned repositories in their own account. This operation is called `fork`ing. Similarly, a contributor can `create` or `delete` branches and tags in their own repositories. A contributor can also `push` changes from their local code repositories to the GitHub code repositories. As shown in Figure 3.1, when a contributor wants to apply changes from their own code repositories ((a) or (c)) to another contributor's code repositories ((b) or (d)), a `PullRequest` is sent to the other contributor's projects.

Issue repositories contain reports of issues including bugs and feature requests. As shown in Figure 3.1, a contributor can `open`, `close`, or `reopen` any issues in her issue repositories for (a), (b), (c) or (d). Contributors can also make comments on commits (`CommitComment`), pull requests (`PullRequestReviewComment`) and issues (`IssueComment`).

In our study, we consider (a) and (b) projects as **target projects** and (c) and (d) as **other projects**. Therefore, we classify contributor's activities in terms of target projects and other projects.

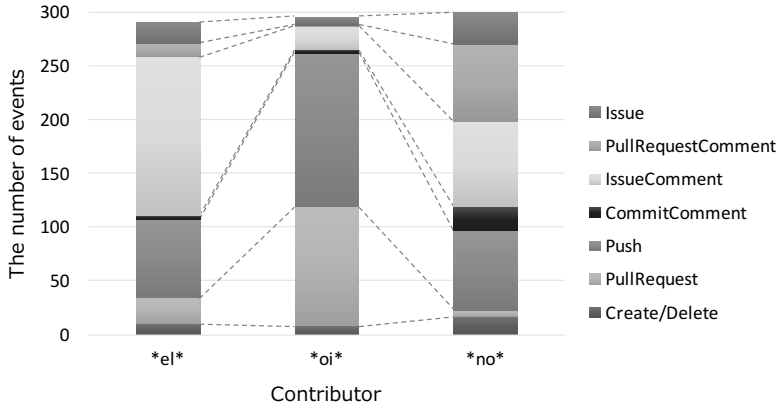


Figure 3.2: Distributions of Contributor's Activities for Three Typical Contributors in the node Project.

Table 3.1: Statistics of Target Projects (Aug. 15, 2013)

Project	Language	Commits	Stars	Forks	Contributors
node	JavaScript	8,974	23,984	4,572	447
jQuery	JavaScript	5,270	22,305	4,587	168

3.3. Method

3.3.1 Target Projects

To investigate various contributors' activities, in this study we selected two active projects, `node`[6] (joyent/node. Description: evented I/O for v8 javascript `http://nodejs.org/`) and `jQuery`[7] (jquery/jquery. Description: jQuery JavaScript Library `http://jquery.com/`). We chose these projects because they have many stars and forks, and they appear in the trending repositories. Table 3.1 presents statistics for these two projects. These projects also have many contributors who have participated in many GitHub activities, as explained in Section 3.2.

Table 3.2 shows the top contributors with more than 100 commits in these two projects, and how many commits each has made. Each project webpage

[6] `https://github.com/joyent/node`
[7] `https://github.com/jquery/jquery`

Table 3.2: Characteristics of Contributors' Activities. Partial Names Are Used, Because of Privacy Concerns.

	node		jQuery	
1	*ry*	2,941	*er*	1,591
2	*sa*	1,413	*me*	436
3	*no*	1,208	*za*	318
4	*is*	502	*wl*	297
5	*nd*	288	*im*	267
6	*oo*	177	*au*	266
7	*oi*	157	*ra*	248
8	*el*	119	*le*	200
9	*re*	114	*ib*	145
10	——		*rk*	117

shows such information. We only show partial names of contributors because of privacy concerns. In this study, we selected data about such top contributors from the two target projects for our analysis of contributors' activities. While GitHub identifies the contributors to projects based on the cumulative number of commits, we intended our study to clarify the differences in contributions between contributors.

3.3.2 Data Collection

Using the GitHub APIs, we collected data about the contributors' activities. There are various APIs for different kinds of GitHub data, such as Git revision control data, issues, repositories, and users. All API access is over HTTPS, and all data is received as JSON data structures. We collected data on Aug. 14, 2013 in two phases, first identifying contributors and second extracting activities.

Phase 1: Identifying Contributors. When we identified a repository, such as joyent/node or jquery/jquery, the repository-related API provided a list

of contributors[8] to that repository. This list included the contributors' names and number of contributions, which is the number of activities related to the project. For our study, we limited the contributors to contributors who have made more than 100 contributions because we wanted to investigate active contributors. In this phase, we identified 9 contributors for the node project and 10 contributors for the jQuery project. Table 3.2 shows all the contributors for both projects.

Phase 2: Extracting Activities. Using the names of the contributors, the activity-related API provided a list of each contributor's activity events[9]. GitHub has 18 different types of activity events. However, because some types of events seldom occurred in our data collection, we ignored them. In our study, we investigated these eight activity events: `CreateEvent`, `DeleteEvent`, `PushEvent`, `PullRequestEvent`, `CommitCommentEvent`, `IssueCommentEvent`, `PullRequestReviewComment`, and `IssueEvent`. Section 3.2 discussed the relationships between the events and the projects. For each event, the event lists include the date and touched repositories. The activity-related API limits the number of events to the most recent 300.

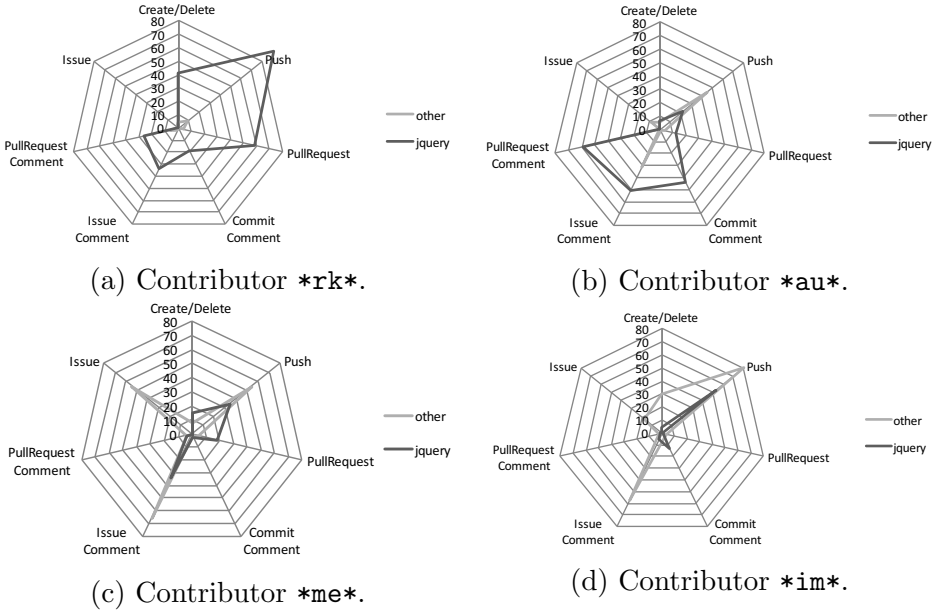
3.3.3 Threats to Validity

Construct Validity. The major threat to the construct validity is the limitation of available event data to the most recent 300 events in the activity-related GitHub API. In our study, we found that 300 events was not large enough for some contributors because these contributors produce 300 events or more within a few weeks, so the most recent 300 may not be representative activities for these contributors. As a result, our study may have only clarified the characteristics of activities in a short-term window. For more representative analysis, data obtained over a longer term is desirable.

External Validity. Our study is limited to two projects written in JavaScript using the GitHub environment, so our results cannot be generalized to other projects and development languages. In addition, these were open source software projects, and contributors' activities in industry software development may be different.

[8] `http://developer.github.com/v3/repos/#list-contributors`

[9] `http://developer.github.com/v3/activity/events/`



(a) Contributor `*rk*`.

(b) Contributor `*au*`.

(c) Contributor `*me*`.

(d) Contributor `*im*`.

Figure 3.3: Radar Charts of Contributor's Activities on the jQuery Project

3.4. Results

We investigated contributors' activities in terms of four areas, the type of activity, the specialization of contributions, contributions relative to the day of the week, and the frequency of activity.

3.4.1 Coding, Commenting, and Issue Handling

First, we investigated the frequencies of the extracted eight events in the 300 events for each contributor to identify their major focus of activities. For convenience, the `CreateEvent` and `DeleteEvent` counts were combined. Figure 3.2 presents a bar chart of the characteristic activities of three typical contributors from the node project. These three contributors were chosen because they represent a variety of approaches. In the bar chart, large areas of each bar represent high numbers of those types of activity events.

The events `Create`, `Delete`, `Push`, and `PullRequest` are related to coding, while the events `CommitComment`, `IssueComment`, and `PullRequestReviewComment` are related to commenting. As can be seen in Figure 3.2, the first contributor,

`*el*` has many `IssueComment` and `Push` events. Similarly, the second contributor, `*oi*`, has many `Push` and `PullRequest` events. The third contributor, `*no*`, has a relatively balanced mix of activities. Based on this categorization of events, we can identify the majority of the first contributor's activities as related to commenting, the second contributor's activities as related to coding, while the third contributor has a balance of activities including both coding and commenting activities. Similarly, for every contributor on both projects, we identified the majority focus of their activities. Table 3.3 summarizes these results.

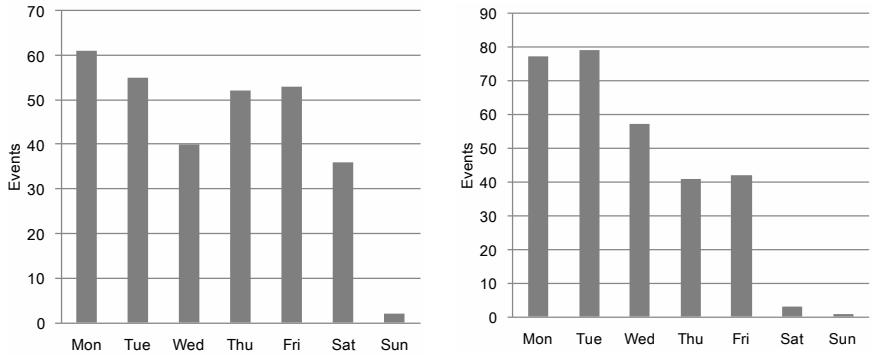
3.4.2 Expertise of Contribution

Second, we investigated how much contributors contribute to their own target projects and other projects. As explained in Section 3.2, we classified projects into target projects and other projects. Figure 3.3 shows the radar charts of the activity events for four typical contributors on the jQuery project. In these radar charts, we separately plotted the contributors' activities on their target projects and other projects. The blue lines represent the activities for the target project, the jQuery project, while the red lines represent activities for other projects. Each axis shows the number of events for a specific activity. These charts allow us to easily see the differences in characteristics of contributors' activities between their target project and other projects.

For example, in Figure 3.3 (a), the contributor contributes mostly to their target project, particularly with code-related activities such as creates, deletes, pushes, and pull requests. In Figure 3.3 (b), the contributor also makes most contributions to their target project, but with comment-related activities such as commit comments, issue comments, and pull request comments. This contributor does not make so many code-related activities such as create, delete, or push. For other projects, this contributor also has some push and issue comment activities.

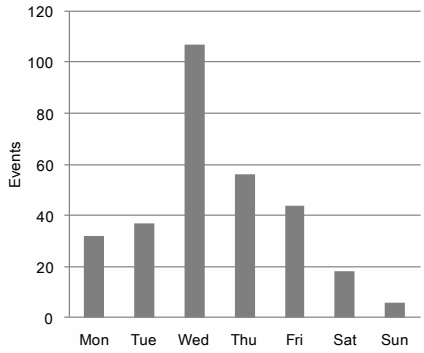
In Figure 3.3 (c), while the contributor makes code-related and issue comment contributions to their target project, most of their activities are on other projects with pushes, issues, and issue comments. In Figure 3.3 (d), the contributor shows similar characteristics, but contributions to their target project are less than other projects, and most of the activities are either pushes or issue comments.

This analysis reveals that top contributors contribute differently to target and

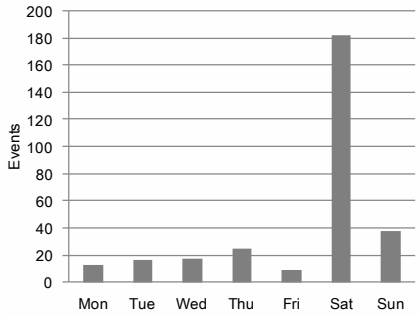


(a) Contributor `*sa*`.

(b) Contributor `*re*`.



(c) Contributor `*is*`.



(d) Contributor `*oi*`.

Figure 3.4: Frequency of Activities during Days of the Week in the Node Project

other projects. Some contributors contribute most of their development activities to their target projects, while others contribute most of their activities to other projects. In our analysis, the rest of the contributors showed similar patterns of activities.

3.4.3 Workdays

Figure 3.4 shows the number of contributors' activities divided across the days of the week. Charts of four typical contributors in the node project are shown. In Figure 3.4 (a), the contributor works Monday through Saturday. In Figure 3.4 (b), the contributor works Monday through Friday and does not work on weekends (Saturday and Sunday). In Figure 3.4 (c), the contributor mainly works on Wednesday, while in Figure 3.4 (d), the contributor works mostly on Saturday. The rest of the contributors have similar distributions. We found

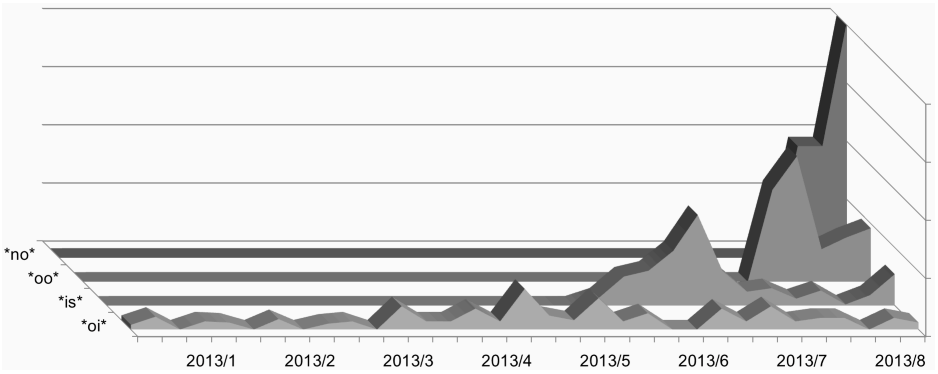


Figure 3.5: Distributions of 300 Activities during Weeks for Four Typical Contributors in the node Project

that some contributors work mostly on weekdays, while others work mainly on weekends. This analysis may allow us to distinguish professional contributors from volunteer contributors.

3.4.4 Frequencies of Activities

Figure 3.5 shows the frequencies of activity events over a range of months for four typical contributors. Because the activity-related GitHub API limits the number of events per contributor to 300, the date of the first recorded event for each contributor differed. For example, in this figure, the contributor `*no*` worked frequently in GitHub, with over 200 activities in a week. For comparison, the contributor `*oi*` worked only a little bit in GitHub each week, so that their 300 activities covers over a year. We found several patterns of work among the contributors, with some contributors producing 300 events in a few weeks, while others took months for 300 events. Table 3.3 summarizes these results in the Activities column.

3.4.5 Overall

Table 3.3 summarizes the analysis results in three key areas, type of activities, specialization, and frequency of activities for the activities of the top contributors with more than 100 contributions to the target projects. The majority column

shows the distribution of activities discussed in Section 3.4.1. If contributors have a mixture of coding, commenting, and issue handling activities, we identified that as *balanced*. He specialization column shows the contributions to the target and other projects as discussed in Section 3.4.2. We identify contributors as *specialists* if they work mainly on their target project, *others* if they work mainly on other projects, *both* if they work on both the target and other projects, and no contribution when the contributor has not contributed to their target project within their 300 activities. The frequency column shows the periods of time needed for each of the contributors to produce 300 activity events as described in Section 3.4.4.

As compared to Table 3.2, which only identified the top contributors and numbers of cumulative comments, Table 3.3 shows the various characteristics of contributions for different contributors. In this table, we can see that some contributors concentrate on coding and/or commenting, while others contribute with a mix of activities including coding, commenting, and issue handling. Top contributors may mainly work on their target projects, some mainly work on other projects, and some have not contributed to their target projects recently. In addition, the activity rates of contributors are very different, with some very active while others are much less active. In each of the two target projects, different contributors have different characteristics of development activities.

3.5. Discussion

This study examined the characteristics of contributors' activities by collecting and analyzing data from two active software projects in GitHub. This included code-related, comment-related, and issue handling activities. While GitHub provides a list of top contributors in each project based on the number of commits, we found that various contributors had different characteristics in their development activities.

In both projects in this study, the top contributors had a mixture of characteristics. For example, some contributors were balanced in doing coding, commenting, and issue handling, while others focused more on code or comments. Further, some contributors were specialists, doing most of their work on their

Table 3.3: Headings: Change Specialty to Specialization, and Activities to Frequency.

	node				
	Contributor	Commits	Majority	Specialty	Activities
1	*ry*	2,941	Balanced	Specialist	Years
2	*sa*	1,413	Code/comments	Both	Weeks
3	*no*	1,208	Balanced	Specialist	Days
4	*is*	502	Code/comments	Others	Months
5	*nd*	288	Code/comments	Both	Weeks
6	*oo*	177	Code/comments	Others	Months
7	*oi*	157	Code	Both	A year
8	*el*	119	Comments	No contribution	Months
9	*re*	114	Code	Specialist	Weeks
	jQuery				
	Contributor	Commits	Majority	Specialty	Activities
1	*er*	1,591	Code/comments	No contribution	Months
2	*me*	436	Balanced	Others	Months
3	*za*	318	Comments	No contribution	Months
4	*wl*	297	Balanced	No contribution	Days
5	*im*	267	Code	Others	Weeks
6	*au*	266	Comments	Specialist	Months
7	*ra*	248	Code	No contribution	Weeks
8	*le*	200	Balanced	No contribution	A year
9	*ib*	145	Code/comments	Specialist	Months
10	*rk*	117	Code	Specialist	Months

own target project, some did most of their work on other projects, and others mixed target and other project contributions. Finally, activity rates varied from days to a year between contributors.

While these results are specific to these projects and GitHub as a development environment, they suggest that active software projects need a mixture of contributors' characteristics, including generalists and specialists in activities such as coding, commenting, and issue handling, as well as contributors who focus on one project and transients who look in on multiple projects. Even the differences in activity levels, while complicating attempts at project management, are part of the open source software environment and should be expected.

3.6. Summary

In this chapter, a study of the characteristics of contributors' activities in OSS development is presented. To clarify the characteristics of contributors' activities, the GitHub activity by each contributor is used to represent the capacity dimensions of the Human Capital discussed in Chapter 2.

Developers were categorized based on measures such as whether they prefer communication by coding or comments, or whether they are specialists or generalists. This study indicates that active software projects have various kinds of developers characterized by different types of development activities.

Capacity in the OSS-HCI means whether contributors learned know-how as a contributor, in other projects before participating in the current the project. In this section, contributors' contributions such as comment, bug report and coding to other projects are clarified. Particularly, we focus on type of contributors' activities and apply to capacity in OSS-HCI.

Chapter 4

Measuring Community Structures

4.1. Overview

In this chapter, the community structure of OSS projects is analyzed using a demographic approach. Software population pyramids are proposed that involve a portion of the deployment and know-how dimensions of Human Capital that were discussed in Chapter 2. Later, these metrics will be used in Chapter 5 to building the HCI.

In this chapter, a case study on the software population pyramids was conducted. Also, future populations in OSS projects are predicted. The goal is to understand the community structure of OSS projects based on the coding contributors and noncoding contributors.

In this chapter, eight case studies from four types of projects were investigated for the two definitions of CCR and NCR. Also, the prediction of a population in a software community is proposed. Then, a future population prediction method using a cohort component method in demography to a prediction of OSS future communities is given.

4.2. Background

As of 2014, GitHub reported having over 3.4 million users and 16.7 million repositories[1]. Why does GitHub attract so many developers? Several studies have identified the essence of this success [19]. GitHub is a distributed version control system (DVCS) and a web-based hosting service for Git repositories. Brindescu et al. assessed the differences between the centralized version control system (CVCS) and DVCS [19]. They reported that developers prefer DVCS because of its useful features, such as the ability to commit locally, work offline while retaining full project history, and create merging branches cheaply. Muşlu et al. reported that developers moved from CVCS to DVCS because DVCS has the ability to work offline, to work incrementally, and to context switch and do exploratory coding efficiently [69]. GitHub has tapped into the opportunity to facilitate pull-based development by offering workflow support tools, such as code reviewing systems and integrated issue trackers. Gousios et al. reported the impacts of pull-based development based on mining repository data: fast development, transparency in project management, attracting contributions, crowd sourcing of code review, and democratizing development [39]. GitHub is also considered as a developers' social networking service, and it promotes software development through formal and informal collaboration, called social coding. Dabbish et al. examined the value of transparency and collaboration in OSS, reporting that developers form a rich set of social inferences, including inferring technical goals and vision and trying to identify projects with similar, and developers combine these inferences into effective strategies for coordinating work, advancing technical skills, and managing their reputations [28].

Although GitHub has many attractive features and many users and repositories, most projects are inactive and have very few commits. Based on a qualitative manual analysis of GitHub repositories, Kalliamvakou et al. reported that the majority of the projects are personal and inactive [52]. To survive and succeed, software development communities need to attract and retain contributors. Yamashita et al. proposed a pair of population metrics, namely, magnetism and

[1]Marisa Whitaker, "Former UC student establishes a celebrated website in GitHub that simplifies coding collaboration for millions of users," University of Cincinnati, April 2014, http://magazine.uc.edu/favorites/web-only/wanstrath.html.

stickiness [119]. Magnet projects are defined as those that attract a large proportion of new contributors, and sticky projects as those where a large proportion of the contributors will continue to make contributions. With the two values of magnetism and stickiness, OSS projects are classified into the following four categories: (1) Attractive projects have high magnet and high sticky values. These projects are successful both at attracting new contributors and at retaining existing ones. (2) Fluctuating projects have high magnet but low sticky values. These projects are successful at attracting new contributors but unsuccessful at retaining them. Therefore, the members of these OSS development communities fluctuate from year to year. (3) Stagnant projects have low magnet but high sticky values. In contrast to fluctuating projects, stagnant projects retain existing contributors but cannot attract new ones. (4) Terminal projects have low magnet and low sticky values. Based on this classification, Yamashita et al. empirically studied OSS project histories and identified at-risk projects.

The work of Yamashita et al. suggests that we should go further to analyze moving human resources of OSS projects in more details not only for the evaluation of project sustainability but also for providing actionable information to help practitioners monitor a project, know what is really working, improve efficiency, manage risk, anticipate changes, and evaluate past decisions [20]. For example, if one could know that a project is attracting new contributors but loosing experienced contributors, then it can be considered that the project is changing its direction originally intended by the experienced contributors. For a straightforward way to analyze such moving human resources, this paper focuses on the populations of development communities. And, to conduct software analytics in populations of OSS development communities, we apply an approach taken from demography. Demography is the scientific study of population. Demographers seek to uncover the levels and trends in a population's size and components [44]. Every population has a different composition: the number and proportion of males and females in each age group. This structure can have considerable impact on the population's current and future social and economic situation. Government policymakers and planners worldwide use population projections to gauge future demand for services and to forecast future demographic characteristics. We believe this perspective, that is, demography for actionable information, is also

important for OSS projects to manage sustainable development communities.

A population pyramid is a graphical illustration of the distribution of the various age groups in a population. Depending on the countries' conditions, the shape of population pyramids varies. Population pyramids are used to show the current status of a country's population and provide insights about political and social stability, as well as economic growth. Population projection is a powerful approach to discuss future populations [80]. In a previous study, we applied population pyramids to OSS development communities. We dubbed this approach software population pyramids [74]. In software population pyramids, contributors are grouped by their experiences in the communities. Extending the previous study [74], this paper investigates the characteristics of the population structures observed in OSS projects in GitHub by introducing demographic analysis. In addition, we project the future of population structures using the well-known cohort component method. We address the following research questions in this paper: What characteristics of population structures exist in OSS projects in GitHub, and can we project future population structures? The differences between this study and the Yamashita et al. study [119] can be summarized as follows:

- The study of Yamashita et al. is based on population migration metrics. Therefore, their research specialized in the migration and remaining of developers. In contrast, we introduced the demographic approach. Therefore, we can investigate population structures deeply and predict the future of development communities with a well-known method.

- Yamashita et al. considered developers to be authors of code changes, so they focused only on the commit and pull request activities. However, we are also interested in other contributors who send issues and comments. So we analyze other activities as well as commit and pull request activities, which makes it possible to understand development communities in detail.

- Our software population pyramids consist of various experience groups in a software development communities. Our method thus allows us to see long-term contributors, though the previous study did not distinguish between the experiences of individual developers.

4.3. Method

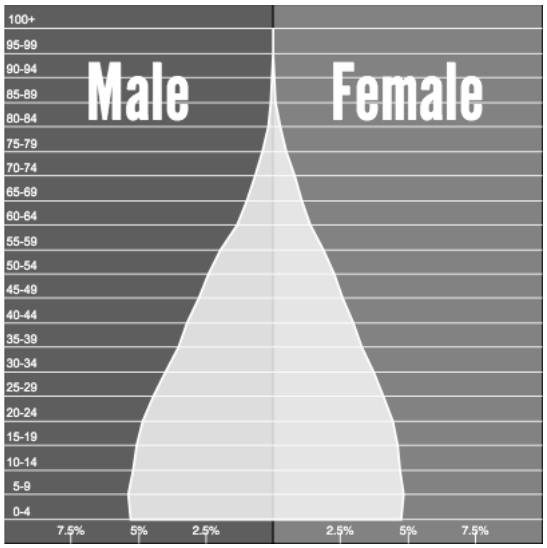
4.3.1 Population Structures

Age and sex are the most basic characteristics of a population. Every population has a different age and sex composition, and this population structure can have considerable impact on the population's current and future social and economic situation [44]. A population pyramid graphically displays a population's age and sex composition.

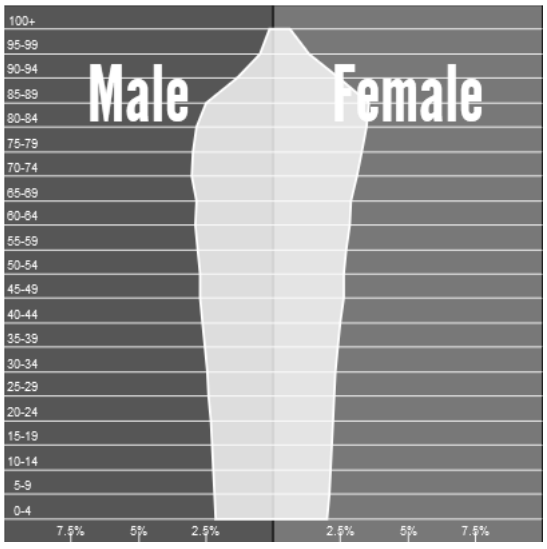
In a general population pyramid, the population is distributed along the horizontal axis, with males shown on the left and females on the right. The male and female populations are broken down into five-year age groups represented by horizontal bars along the vertical axis, with the youngest age groups at the bottom and the oldest at the top. The shape of the population pyramid gradually evolves over time, following trends in fertility, mortality, and international migration. We can understand the status of a country just by looking at population pyramids. Figure 4.1 (a) shows the population pyramid of India in 2010. This pyramid is large toward bottom, a form that is common in developing countries. Population pyramids are also useful for predicting the future composition of a population. Figure 4.1 (b) shows the projected population pyramid of Japan in 2050. It seems like a tower rather than a pyramid. This form is common in low birth rate and high longevity countries. Depending on the countries' status, the shape of population pyramids varies.

4.3.2 Software Population Pyramids

There are various contributors to the OSS project. There are, for example, bug reporters, commenters, reviewers, and coding contributors. All contributions and various contributors are important for OSS projects. For example, bug reporters assume an important role in improving the quality of OSS [89]. Also, developers can keep up motivation by getting some comments of thanks, admiration, or opinion. However, coding contributors, bug reporters, and commenters differ essentially. Contributors can comment or report bugs without a deep understanding of source code files, but coding contributors need to understand them. So,



(a) India in 2010



(b) Japan in 2050

Figure 4.1: General Population Pyramid
(https://www.populationpyramid.net)

coding contributors are considered to be required to have specific skills, unlike bug reporters or commenters. Therefore, in this study, we distinguish coding contributors with other, non-coding contributors.

It is often the case that core developers contribute to both coding and non-coding activities. We identify individuals as coding contributors if the contributors have experienced code-related activities at least once in his/her existing period. If a contributor only has non-coding activities in a given period, he/she is regarded as a non-coding contributor. Then if the contributor begins code-related activities later, he/she will be classified as a coding contributor. To clarify such transitions, we call such contributors "moved contributors". End users play an important role in maintaining contributors' motivation [124]. However, because contributions of end users are not recorded in software repositories in general, our study do not consider them.

We have proposed software population pyramids: population pyramids of software development communities [74]. Contributors are considered to be the constituent member of the communities, and the contribution periods are regarded as existing periods or lifetimes. A software population pyramid consists of two back-to-back bar graphs, with the population plotted on the X-axis and experience on the Y-axis. The bar graph on the right shows coding contributors, and the bar graph on the left shows non-coding contributors in a particular population in three-month experience groups. In a general population pyramid, the populations are broken down into five-year age groups. However, we should make software population pyramids with shorter periods because the five-year length is too long for OSS projects. In our previous study, we analyzed software population pyramids in one year length [74]. However, we found that many contributors leave projects within a year. In addition, less than three months is too short for many projects to obtain enough data to draw a population pyramid. So, in this study, we make software population pyramids with three months groups, and analyze population of contributors in OSS projects.

There are some differences between our software population pyramids and the general population pyramids.

- Whereas a population pyramid consists of bars for males and females, a software population pyramid consists of coding bars for contributors and

non-coding contributors.

- In a general population pyramid, people appear at birth and disappear when they die, but in a software population pyramid, contributors start their experiences when they enter and finish when they leave the development communities. In this study, we consider that a contributor left a project when he/she did not give any contribution on that project for more than three months. However, very few contributors might come back to the project after three-month (or more) interval. They disappear from the pyramids while they are inactive temporarily. In that case, we consider them as experienced contributors when they come back to the project.

- The height of general population pyramids are similar to each other, because maximal life-span of human is not so different in each country. However, software population pyramids have different heights, because OSS projects have different existing periods and people can leave freely.

- Because the parent-child relationships exist in population pyramids, there are correlations between the volume of the parent population and the population of children. However, software population pyramid do not exhibit such relationships. This can cause the pyramid to change dramatically.

4.3.3 Dataset

We analyze the GitHub dataset provided by Gousios [38] in MSR mining challenge 2014[2]. This dataset includes developers' activity histories for 90 OSS projects. Figures 4.2 and 4.3 show point diagrams that plot metrics of projects. Figure 4.2 shows the distribution of development periods and the number of contributors. From Figure 4.2, we can see that homebrew has many contributors and that the development period of rails is long. Figure 4.3 shows the distribution of the number of coding contributors and the number of non-coding contributors by project. From Figure 4.3, we see that homebrew and rails have many coding and non-coding contributors. From small to large-scale projects, this dataset includes various types of projects.

[2]http://2014.msrconf.org/challenge.php

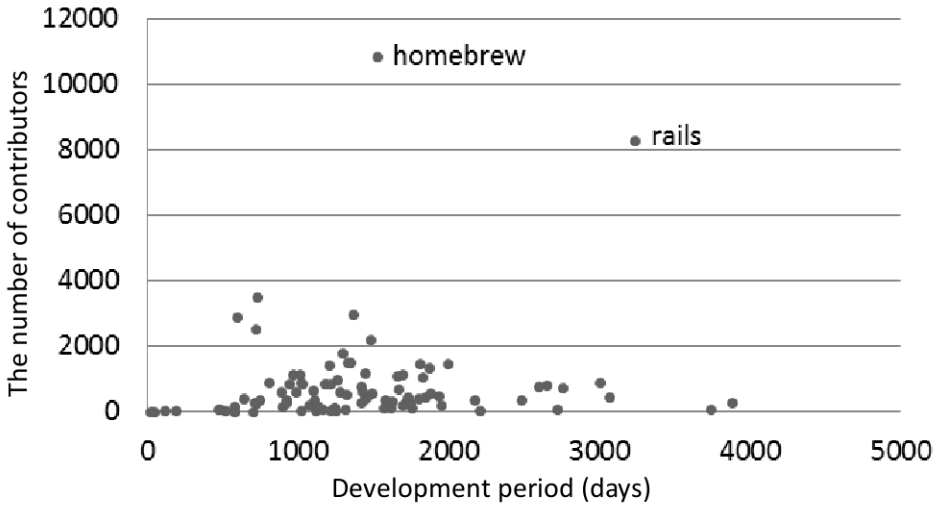


Figure 4.2: Distribution of Development Periods and the Number of Contributors.

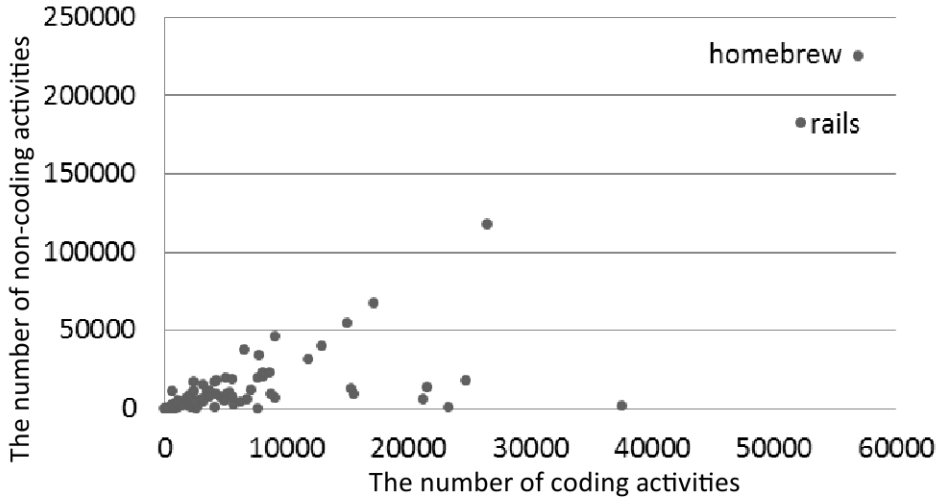


Figure 4.3: Distribution of the Number of Coding Contributors and the Number of non-coding Contributors.

Table 4.1: GitHub Development Activities

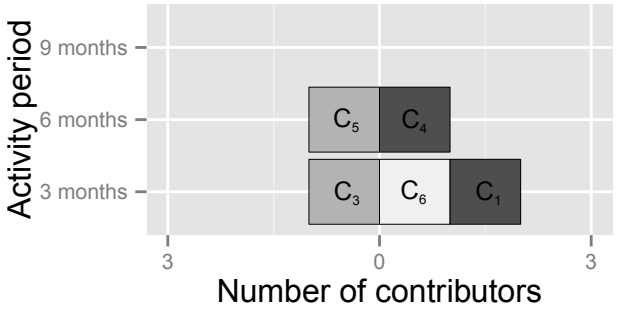
Development Activities	Overview	Separation
commits	Commit to the repository	coding
pull_requests	Request a commit to committers	
commit_comments	Comment against commit	non-coding
issues	An issue associated with a repository	
issue_comments	Comment against commit	
pull_request_comments	Comment against commit pull_request	
events	This is a read-only API to the GitHub events.	excluded in this study
followers	A follower to a user.	
forks	A copy of a repository	
org_members	Users that are members of an organization.	
repo_collaborators	Users with access to the repository.	
repo_labels	Label list is labeled to the repositories	
repos	A dump of every public repository	
issues_events	An event on an issue	
users	Github users.	
watchers	Users that have starred (was watched) a project	

Table 4.2: Example of Data of Activity and Activity Periods of Contributors in t_1 and t_2

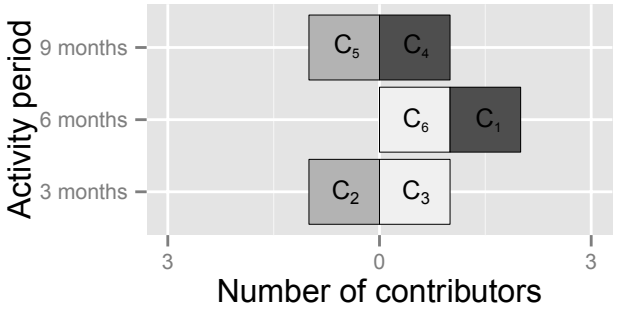
	Pyramids in t_1		Pyramids in t_2	
	Working months		Working months	
Contributor	coding	non-coding	coding	non-coding
C_1	1	-	4	-
C_2	-	-	-	2
C_3	-	3	2	6
C_4	5	2	8	5
C_5	-	4	-	7
C_6	3	5	6	8

In total, this dataset includes 16 development activities, but to focus on contributors' activities, we use only six development activities. Table 4.1 shows the name of 16 development activities and an explanation of the content of these activities. Pull_request and commits are considered to be coding-related activities, whereas commit_comments, issue_comments, pull_request_comments and issue_events are considered non-coding activities. Events, followers, org_members, repo_collaborators, repo_labels, repos, users and watchers are not related to contributors' activities. "Forks" is generally a contributor's activity; however, fork itself does not contribute to the development, and also fork is often done by a person before he/she participate in a development as a contributor. So we excluded it from the contributors' activity list.

We classified contributors as coding and non-coding contributors. **Coding** contributors are contributors who have at least one code-related activity in their existing periods. **Non-coding** contributors are contributors who have not experienced code-related activities but have experienced non-coding activities. We obtained the dates of those events for each contributor, and identified the contribution period from the first event until the last event. Details of how to obtain the data are explained in Appendix A. Contribution periods are divided into **coding periods** and **non-coding periods** based on the classification of the activity events. If a contributor has only non-coding activities in his/her early period, the



(a) Pyramids in t_1.



(b) Pyramids in t_2.

Figure 4.4: Examples of Software Population Pyramids in t_1 and t_2

period is regarded as a non-coding period and he/she is regarded as a non-coding contributor. If a contributor has coding-related activities, the period is regarded as a coding period and he/she is regarded as a coding contributor.

Table 4.2 shows an example of data of activity and activity periods of contributors, and Figure 4.4 shows software population pyramids that plot the data of Table 4.2. The time t_2 is three months later to the time t_1. The X-axis is the number of contributors. The center is zero, the right side shows the number of coding contributors, and left side shows non-coding contributors. The Y-axis is the activity period of contributors. For example, contributor C_1 has coding activity periods of one month in t_1 and four months in t_2. Therefore, he/she is plotted as C_1 in location in Figure 4.4 (a) and Figure 4.4 (b) as a coding

contributor. Contributor C_2 has a non-coding activity period of two months in t_2. He/she is plotted as C_2 in location in Figure 4.4 (b) as a non-coding contributor. In contrast, contributor C_3 has a non-coding activity period of three months in t_1. He/she is plotted as C_3 in location in Figure 4.4 (a) as a non-coding contributor. However, he/she has a coding activity period of two months in t_2. Therefore, he/she is plotted as C_3 in location in Figure 4.4 (b), having moved to the contributor side.

The method of calculating activity periods used here does not take into account actual activity between start and end. In a previous study, we analyzed the frequency of activities of contributors, finding that, although some contributors continued to make small contributions for long periods, there is no contributor that stops activities in a project and then rejoins the project later [73]. However, it is important to take into account the frequencies of contributions. This could be the future work of this study.

4.4. Results

4.4.1 Characteristics of Population Structures

We classify the shapes of software population pyramid, and investigate their characteristics. For this purpose, we propose two new measures. One is the proportion of the number of non-coding contributors (non) to the number of coding contributors (coding), called the Coding Contributors Ratio (CCR). CCR is defined as follows:

$$CCR = \begin{cases} \dfrac{coding - non}{coding} & (coding \geq non) \\ \dfrac{coding - non}{non} & (coding < non) \end{cases}$$

CCR ranges from -1 to 1. Higher values mean that more contributors are coding contributors, and lower values mean that more contributors are non-coding contributors. If the value is close to 0, the number of coding contributors and the number of non-coding contributors are similar.

The other proposed measure is the proportion of the number of experienced contributors to the number of newcomers (new contributors), called the New

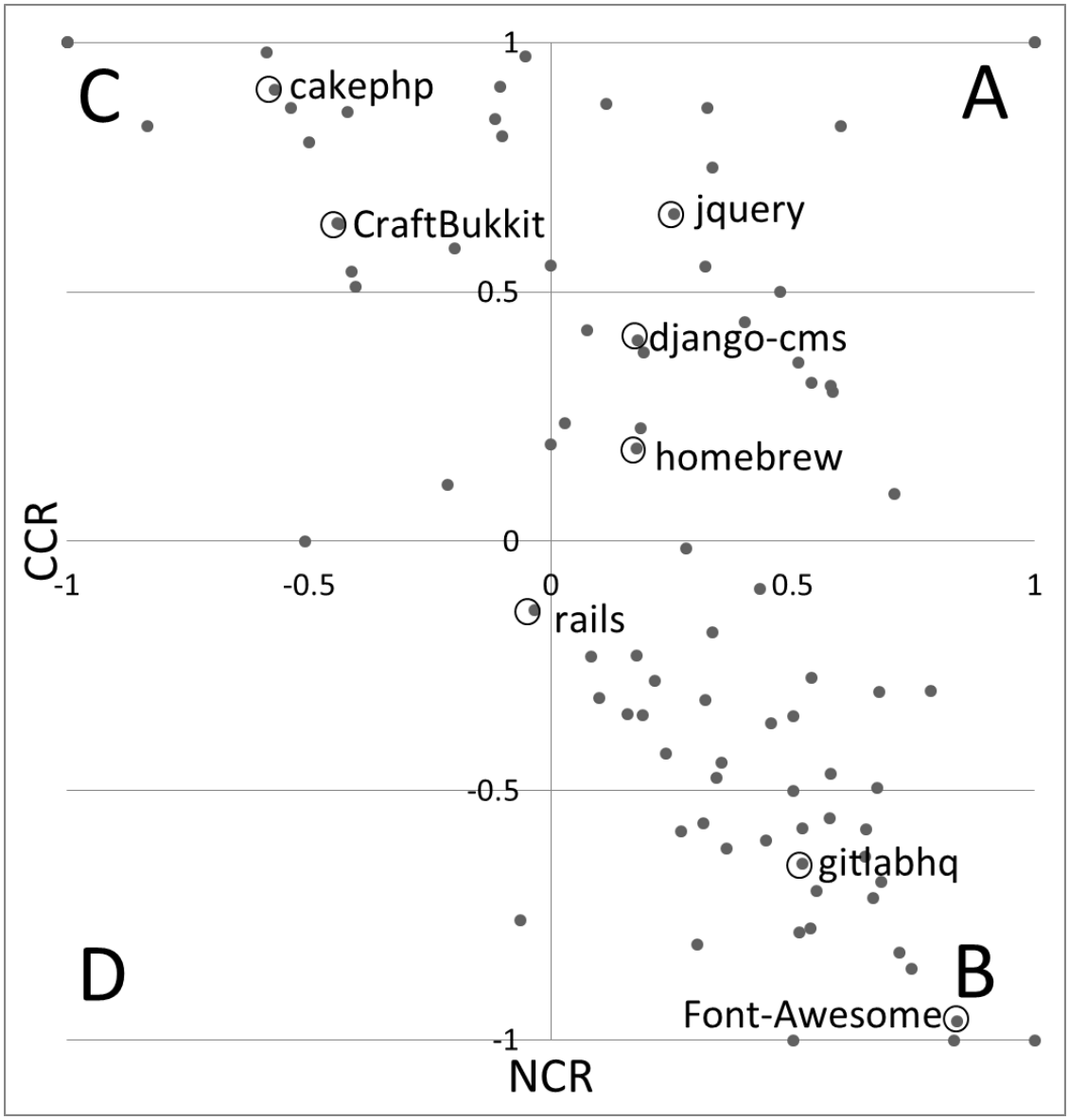


Figure 4.5: Distribution of CCR and NCR of OSS Projects in GitHub.

Contributors Ratio (NCR). In this study, we define newcomers as contributors who have less than three months of activity periods, and we define experienced contributors as those with longer activity periods. NCR is defined as follows:

$$NCR = \begin{cases} \frac{new - experience}{new} & (new \geq experience) \\ \frac{new - experience}{experience} & (new < experience) \end{cases}$$

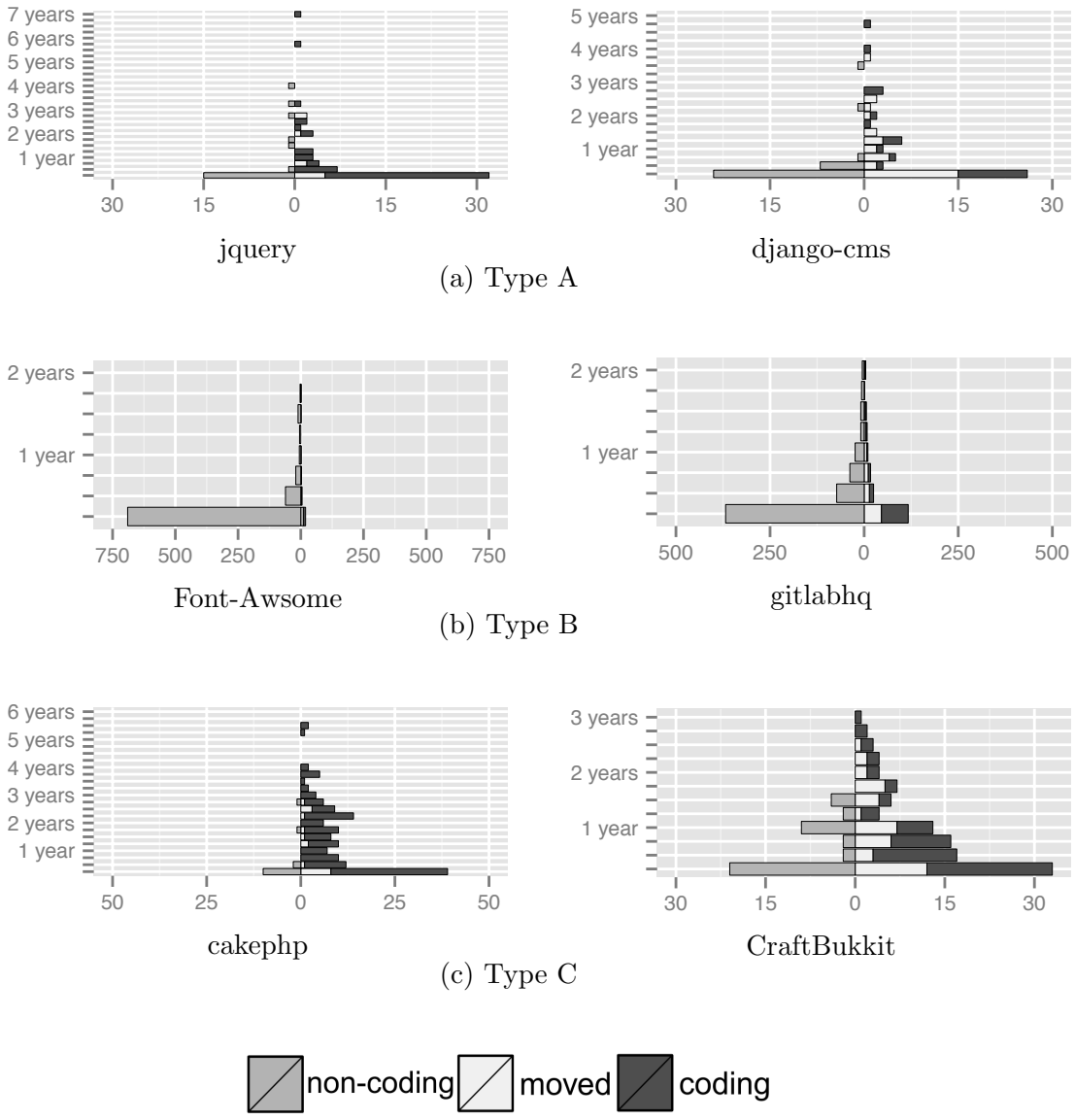
NCR ranges from -1 to 1. Higher values mean that more contributors are new contributors, and lower values mean that more contributors are experienced contributors. If the value is close to 0, the number of new contributors and the number of experienced contributors are similar.

Figure 4.5 shows the distribution of the projects using the CCR and the NCR in September 2013. Because four projects did not have any contributors in this period, we could not plot them. For that reason, there are 86 projects displayed in Figure 4.5. With this distribution, we can classify the projects into the following four types:

- Type A: There are more newcomers than experienced contributors, and more coding contributors than non-coding ones in a project. So, the shape of software population pyramid on the right side is larger than the left side, and the bottom is larger than others, also experienced contributors are plotted intermittently.

- Type B: There are more newcomers than experienced contributors, and more non-coding contributors than coding ones in a project. So, the shape of software population pyramid on the right side is larger than the left side,and the bottom part is larger than other parts. Also, experienced contributors are plotted intermittently.

- Type C: There are more experienced contributors than newcomers, and more coding contributors than non-coding ones in a project. So, the shape of software population pyramid on the left side is larger than the right side, and the bottom part is larger than other parts. Also, experienced contributors are plotted intermittently.



(a) Type A

(b) Type B

(c) Type C

Figure 4.6: Examples of Software Population Pyramids of each Type. (Note that scales are different)

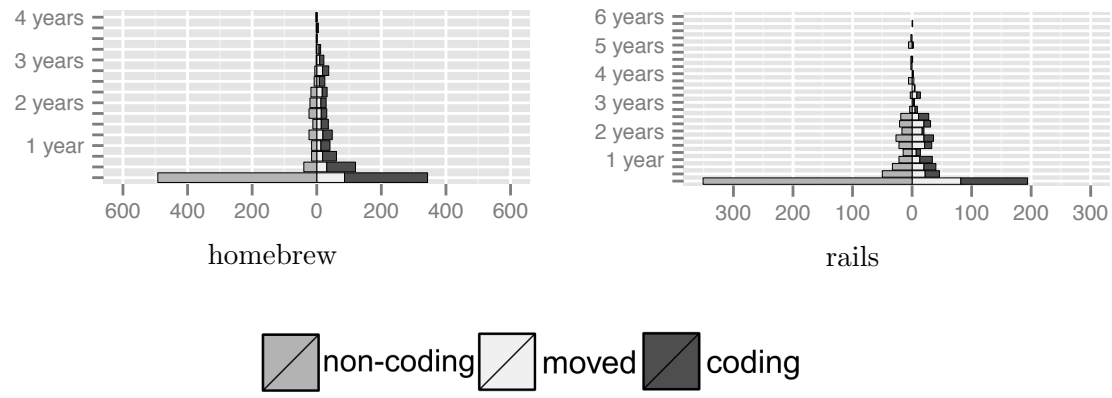


Figure 4.7: Examples of Software Population Pyramids (CCR and NCR are close to 0). Scales are Different

- Type D: There are more experienced contributors than newcomers, and more non-coding contributors than coding ones in a project. So, the shape of software population pyramid on the left side is larger than the right side, and experienced contributors are plotted continuously.

There are 23 projects categorized Type A, 42 projects as Type B, 18 projects as Type C, and three projects as Type D. Figure 4.6 presents examples of software population pyramids belonging to Type A, Type B, and Type C.

(a) Type A In these projects, there are few experienced non-coding contributors. In django-cms, there are many moved contributors. Because many developers moved from non-coding to coding, these projects have many coding contributors.

(b) Type B Font-Awesome has many non-coding contributors. This project makes Web icon fonts, and many people sent requests for new icons to this project. Therefore, many non-coding contributors leave this project immediately following a short period of contribution.

(c) Type C There are many moved contributors in CraftBukkit. Also, there are many coding newcomers. However, many coding contributors continue their activities, because there are more experienced contributors than there are newcomers.

In Figure 4.6, we see that the shapes of the pyramids are different from each other. OSS projects are managed by voluntary contributors, so contributors may

not correspond to the many bug reports in projects of Type B. Additionally, it is difficult to obtain coding newcomers, because there are no moved newcomers. However, there are a few contributors that report bugs, such as in Type A or C, so these projects have little chance of improving the quality of the OSS through bug reports.

Figure 4.7 shows the software population pyramids of homebrew and rails. Homebrew belongs to Type A, and rails belongs to Type D. In these projects, both CCR and NCR values are close to 0. These projects are continually gaining contributors because their software population pyramids do not have intermittent bars. In addition, there are many moved contributors. We can see that these projects succeeded in attracting and retaining new/experienced and coding/non-coding contributors.

Projects that are plotted close to the center of the graph are well balanced in CCR and NCR. In these projects, there are an almost equal the number of contributors and newcomers, and almost equal the number of non-coding contributors and coding ones too. It is our important future work to consider adding another (5th) project type to distinguish such projects from others. For example, if we distinguish the projects that plotted around the origin belonging to the top 10% and others, six project such as homebrew, rails, bitcoin, diaspora, openFrameworks and redis meet that definition.

4.4.2 Population Projection

For project managers, it is important to maintain experienced contributors. Therefore, we propose a population projection method of the number of contributors in OSS projects using demographic methods.

Cohort component population projection

We predict the number of contributors using a simplified cohort component population projection. In demography, a cohort is a group of subjects share a particular event during a particular time span. Cohort component population projection is the simplest population projection method. Isserman offers a way to project the

size of populations [45]. Isserman's method uses the survival rate [90], as well as fertility, mortality, and migration data. We can project the size of populations at a certain age cohort using following formulas:

$$\begin{aligned} &Population\ of\ age\ (X\ +\ n)\ in\ year\ (T\ +\ n)\ = \\ &Survival\ Rate \times Population\ of\ age\ X\ in\ year\ T \end{aligned}$$

where

$$Survival\ Rate = \frac{Population\ of\ age\ (X\ +\ n)\ in\ year\ T}{Population\ of\ age\ X\ in\ year\ (T\ -\ n)}$$

X is the age of the cohort being examined, n is an interval of time usually set at ten years representing the period of time between the two most recent censuses, and T is the year of the most recent census. We replace each variable in our software population pyramids such that X is the activity period of the cohort being examined, n is an interval of time set at three months representing the period of time between the two most recent contributors counting, and *3 m in year T* is the month of most recent contributors counting.

For example, we consider a case of a projection 10 to 19 year-old population in 2020. In this projection, we use 0 to 9 year-old population in 2000 and 10 to 19 year-old population in 2010 to calculate a survival rate of 0 to 9 year-old population. Here, the survival rate is calculated as follows.

$$Survival\ Rate = \frac{Population\ of\ age\ (10\ to\ 19)\ in\ 2010}{Population\ of\ age\ (0\ to\ 9)\ in\ 2000}$$

where

$$\begin{aligned} Population\ of\ age\ (10\ to\ 19)\ in\ 2020\ = \\ Survival\ Rate\ of\ 0\ to\ 9 \times \\ Population\ of\ age\ (0\ to\ 9)\ in\ 2010 \end{aligned}$$

In this way, to calculate the population of each cohort and to sum them. The cohort component method includes birth and net migrants in general. Births are the same as newcomers to an OSS project in our study. Births are derived from the number of mothers and the birth rate. However, these input data do not exist for the number of contributors, so we use following very simple formula to calculate newcomers:

$$newcomer = (P\_\{T\} + P\_\{T - n\}) / 2$$

where

$$P\_\{T\} = Population\ activity\ period\ 3\ m\ in\ year\ T$$

Additionally, we do not consider that contributors move to other projects in our study, so we do not calculate net migration.

Evaluation

With the cohort component population projection method, we project a future population size for the 36 projects that have more than 100 contributors. There are four projects categorized as Type A, 21 projects as Type B, nine projects as Type C, and two projects as Type D. In this study, we project the number of contributors of September 2013 by calculating the survival rate from the number of contributors of March and June 2013. In order to verify the projection accuracy of our proposed method, we compared it with the baseline method, which assumes that the number of contributors of September and June 2013 are the same. Populations are projected for non-coding, moved, and coding contributors, separately.

To evaluate the projection accuracy, we compare the projection error of our propose method to the baseline method one. MRE (Magnitude of Relative Error) [25] or MER (Magnitude of Error Relative to estimate) [56] are used to evaluate the prediction accuracy. We use ABRE (Absolute Balanced Relative Error) [67] as an evaluation metric of the prediction accuracy for the number of contributors remaining.

Measured values of the number of contributors is denoted as x, and the predicted value of the number of contributors is denoted as $\widehat{x}$. Each indicator is determined by the following equation:

$$MRE = \frac{|x - \widehat{x}|}{x}$$

$$MER = \frac{|x - \widehat{x}|}{\widehat{x}}$$

$$ABRE = \begin{cases} \dfrac{|\widehat{x} - x|}{x} & (\widehat{x} - x \geq 0) \\ \dfrac{|\widehat{x} - x|}{\widehat{x}} & (\widehat{x} - x < 0) \end{cases}$$

For these metrics, lower values indicate higher accuracy. MRE is the relative error of the predicted value to the actual value and MER is the relative error between predicted and actual values to the predicted value. However, MRE and MER share the problem that these measures cannot distinguish excessive prediction and too little prediction. In this study, we evaluated projection accuracy by using the ABRE to evaluate the balance between excessive prediction and too little prediction.

To investigate the projection accuracy, we used the Wilcoxon non-parametric statistical hypothesis test. Wilcoxon test is generally used when comparing two related samples to assess whether their population mean ranks differ. It can be used as an alternative to the paired Student's t-test, t-test for matching pairs, or the t-test for dependent samples when the population cannot be assumed to be normally distributed. With the Wilcoxon test, we test the difference between the ABREs of our propose method and the ABREs of the baseline method. In particular, we test each project type (Type A-D) and each contribution type (non-coding, moved, coding). Then, we test their measures of central tendency, and investigate whether there are significant differences in projection accuracy.

Table 4.3 shows the median of the ABREs. If the ABRE value is close to 0, the projection accuracy is high. In Table 4.3, projection accuracies of our proposed method are higher than the baseline method in all predictions. Table 4.4 shows the result of the Wilcoxon test (95% confidence), where bold numbers indicate the

Table 4.3: Median of ABRE

	non-coding		moved	
	cohort	baseline method	cohort	baseline method
Type A	0.4993	0.5000	0.3027	0.5000
Type B	0.5000	0.6332	0.3923	0.5000
Type C	0.6711	1.0000	0.2500	0.7179
Type D	0.3137	1.0000	0.2378	0.2500
All types	0.5000	0.6667	0.3299	0.5000
	coding		All	
	cohort	baseline method	cohort	baseline method
Type A	0.1865	0.3731	0.2534	0.5000
Type B	0.5000	0.5750	0.5000	0.5917
Type C	0.3684	0.6250	0.3333	0.7500
Type D	0.4808	0.6154	0.2875	0.6667
All types	0.4074	0.5417	0.4000	0.6000

Table 4.4: Result of Wilcoxon Test

	p-value			
	non-coding	moved	coding	All
Type A	**0.02748**	**0.00158**	0.26867	**0.00014**
Type B	**0.00037**	**0.01845**	0.06200	**0.00001**
Type C	0.05935	**0.00035**	0.16000	**0.00013**
Type D	**0.00001**	**0.02700**	**0.02901**	**0.00000**
All types	**0.00000**	**0.00000**	**0.00116**	**0.00000**

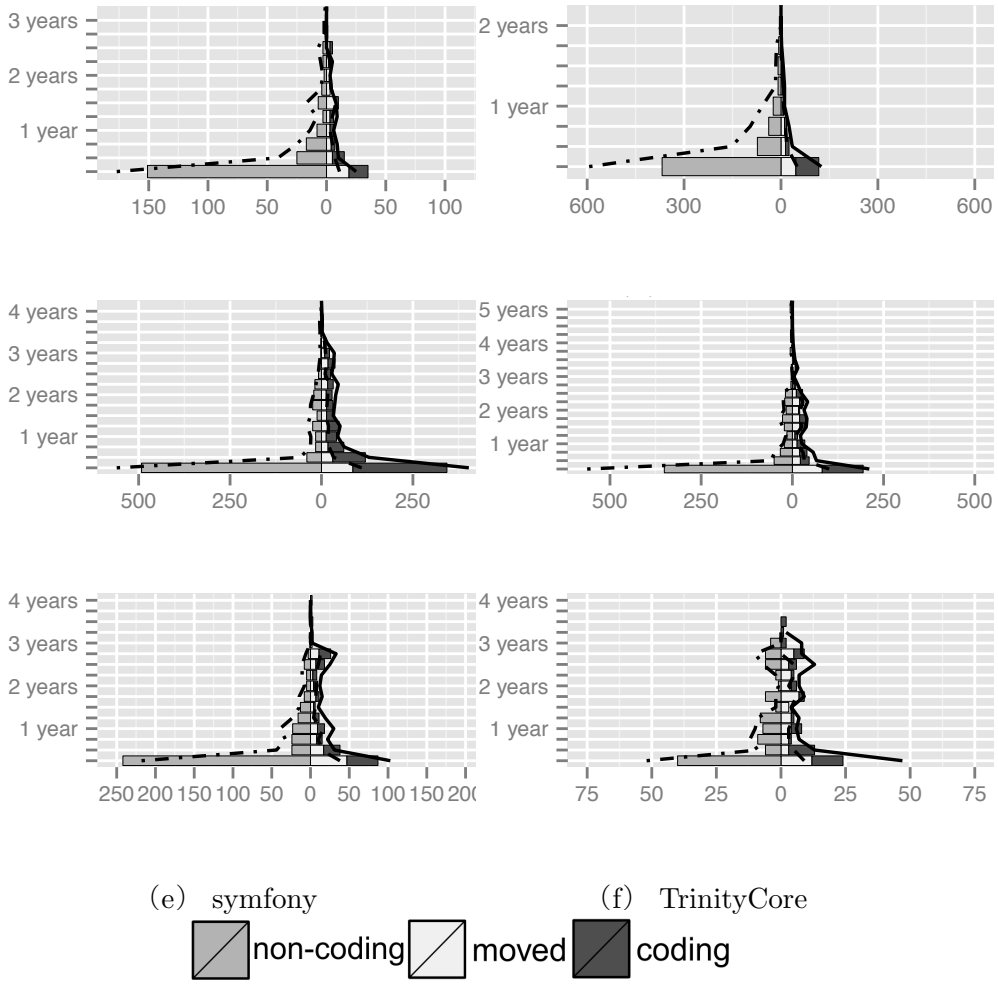


Figure 4.8: Comparing Measured and Predicted Values of the Number of Contributors

statistical significant improvements by the proposed (cohort) method. In Table 4.4, projection of non-coding contributors has no significant difference in Type C, and projections of coding contributors have no significant difference in Type A, B, and C. However, projection of all contributors and all types have significant difference. In this result, the projection accuracy of our proposed method was higher than the baseline method. On the other hand, there was no difference in the predictive accuracy between different project types in this projection. This result shows the possibility that the reduction of contributors depends little on type of activities, the number of newcomers or experienced contributors.

Figure 4.8 shows actual software population pyramids and lines of predicted values. We can see that most of the lines are surprisingly well fitted to the observed data, especially near the top of each pyramid. In this study, we define short-term contributors as contributors that have activity period of less than one year, and define long-term contributors as contributors that have activity period of equal to or more than one year. The median of ABRE of short-term contributors is 0.4055, and median of ABRE of long-term contributors is 0.3333. The result of the Wilcoxon rank test (95% confidence) showed that the difference is significant (p-value = 0.0460), which indicates that the projection of the number of long-term contributors is higher accuracy than the projection of the number of short-term contributors.

4.5. Discussion

In this study, we focus on contributors to OSS projects using a demographic approach. In OSS projects, many people are involved in development, meaning that human resources are very important for OSS. We conclude that we can predict the future of participation in OSS projects by analyzing them from a demographic perspective.

In the field of demography, researchers create population pyramids to analyze the current situation of selected countries. We proposed a population pyramid for OSS projects called the software population pyramid. Contributors are considered the constituent member of the communities, and the contribution periods are regarded as experience periods or lifetimes. A software population pyramid con-

sists of two back-to-back bar graphs, with the population plotted on the X-axis and experience on the Y-axis. One of the bar graphs shows coding contributors and the other shows non-coding contributors in a particular population in three-month experience groups.

We classified shapes of the software population pyramid and compared them. To classify the shape of these pyramids, we proposed two new measures, CCR and NCR. CCR is the proportion of the number of non-coding to the number of coding contributors, and NCR is the proportion of the number of experienced contributors to the number of newcomers. Using these measures, we classified 86 software population pyramids into four types as follows.

- Type A: There are more newcomers than experienced contributors, and more coding contributors than non-coding ones in a project.
- Type B: There are more newcomers than experienced contributors, and more non-coding contributors than coding ones in a project.
- Type C: There are more experienced contributors than newcomers, and more coding contributors than non-coding ones in a project.
- Type D: There are more experienced contributors than newcomers, and more non-coding contributors than coding ones in a project.

There were 23 projects categorized as Type A, 42 projects as Type B, 18 projects as Type C, and three projects as Type D. The result indicates that, for projects in Type A and Type B, contributors do not stay long time in their projects after their contributions. For projects of Type C and Type D, they cannot get enough newcomers; thus, they should consider how to recruit newcomers. So, those projects should consider how to get newcomers. Especially, Type C projects should attract non-coding newcomers, e.g. who post many issues, so that some of them might become coding newcomers as well.

Through empirical research, we found that the shapes and the transitions of software population pyramids vary depending on the status of the development communities. However, it is difficult to clarify the components of the contributor population of OSS projects using only these values. Other, new metrics are

needed to clarify contributors' components in OSS projects. For example, the number of activities or frequency of activities should be considered.

The demographic approach of population projection is a powerful way to predict future population dynamics. In this study, we projected the number of contributors of September 2013 using the simplified cohort component population projection that calculates the survival rate from the number of contributors of March and June 2013. In order to verify the projection accuracy of our proposed method, we compare it with the baseline method, which assumes that the number of contributors of September and June 2013 are the same. To statistically compare the projection accuracy, we used the Wilcoxon non-parametric statistical hypothesis test. As a result, the projection accuracy of our proposed method was higher than the baseline method. However, this projection method cannot predict long-term contribution patterns because it does not predict newcomers in a narrow sense. Therefore, our future work includes improving the accuracy of these predictions and expanding the prediction to account for newcomers and to extend predictions into the long-term future. We believe this perspective is also important for OSS projects to manage sustainable development communities.

4.6. Summary

In this chapter, the community structure of OSS projects is analyzed using a demographic approach. There are four types of population structures in OSS development communities in terms of experiences and contributions. In addition, the future population was predicted accurately using a cohort component population projection method. This method predicts a population of the next period using a survival rate calculated from past populations.

To the best of my knowledge, this is the first study that applied demography to the field of OSS research. This new approach addressing OSS-related problems based on demography will hopefully bring new insights, since studying population is novel in OSS research. Understanding current and future structures of OSS projects can help practitioners to monitor a project, gain awareness of what is happening, manage risks, and evaluate past decisions.

Deployment in this OSS-HCI represents the workforce in a project. In this section, a measure that shows the balance between coding contributors and non-coding contributors is defined. This measure analyzes whether a project has various workforces. Also, know-how in the OSS-HCI represents maintaining and obtaining contributors. In this section, measures that show the balance between new contributors and existing contributors are defined. These measures analyze whether a project can maintain and obtain contributors. These points are applied to deployment and know-how in OSS-HCI.

Chapter 5

OSS Human Capital Index

5.1. Overview

In this chapter, an OSS Human Capital Index (OSS-HCI) is constructed. The HCI indicators are based on the results of the mapping study in Chapter 2. In detail, the set of indicators are introduced and metrics are defined for the HCI. Some of the metrics are derived from the previous studies of contributors' activities (Chapter 3) and community structure (Chapter 4).

In this thesis, OSS projects are assessed based on the four dimensions of OSS Human Capital. The goal is to determine whether or not the defined indexes provide meaningful and interesting insights into the relationship between the OSS HCI. The following research questions guide this thesis:

RQ_1: *Can projects be classified based on the OSS Human Capital Index (HCI)?*

RQ_2: *How effective is HCI in identifying and tracking OSS projects?*

RQ_3: *Which metrics affect OSS-HCI?*

RQ_4: *Is the OSS-HCI useful compared to other evaluations of OSS projects?*

To answer RQ_1, an index using indicators was constructed. Then, the index was used to classify 1,418 OSS projects. To answer RQ_2, case studies that depict different patterns of Human Capital over time were investigated. To answer RQ_3, a regression analysis was conducted to choose the valid metrics. To answer RQ_4, other evaluations of OSS projects and the OSS-HCI were compared.

Table 5.1: Proposed Human Capital Indicators.

Dimension	Metrics	Indicator
Capacity	CAP1	Individual Developer activity profiling
	CAP2	
	CAP3	
	CAP4	
Deployment	DEP1	Community Structural Complexity
	DEP2	Core vs. Peripheral Developer workload equality
	DEP3	Contributor Participation rates
	DEP4	Productivity rates
Development	DEV1	End-user Participation
Know-how	KH1	Core vs. Peripheral knowledge
	KH2	Knowledge loss rates
	KH3	Onboarding rates

5.2. Human Capital Index Construction

Table 5.1 describes the proposed indicators and is mapped to respective inspired consolidated papers from the mapping study in Chapter 2. Figure 5.1 depicts an overview of the process used to construct the HCI Score. Comprised of three steps, first the indicator metrics are calculated (Step1), then (Step2) they are normalized and a score based on the three dimensions is summarized (i.e., deployment, development and know-how). Finally, (Step3) the HCI Index score for each project is calculated.

Indicator Metrics (Step1)

The rationale and definition of each indicator is described by dimension. Then, the metrics used to calculate each indicator are explained.

Capacity Individual Developer activity profiling This indicator shows the individual rating score of the developer. Examples could include a study about measuring team personality and climate (i.e., S19). This study measures neuroticism, extroversion and the conscientiousness of developers through experiment. Another study is related to the personality profiles of developers (i.e., S16). This study analyzes message exchanges or developers' tasks using code and assets data

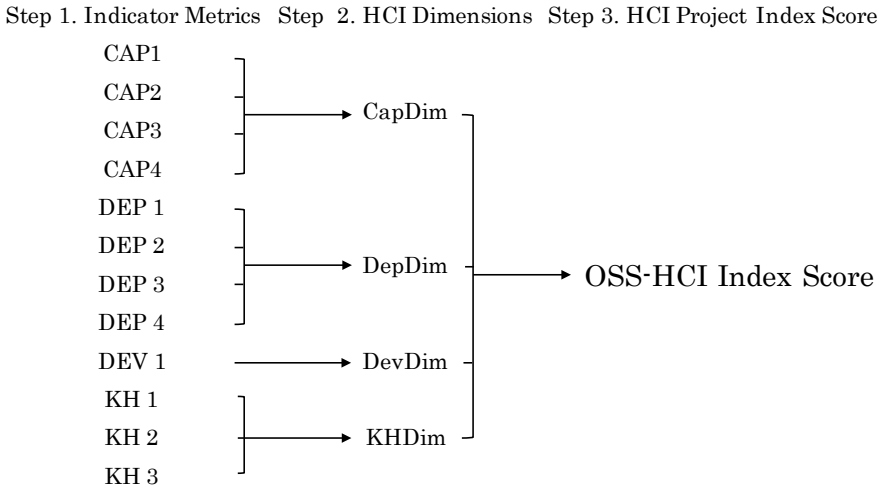


Figure 5.1: Construction of the HCI Index and HCI Dimension Plot

sources.

In this thesis study, four metrics regarding individual developer activity as capacity are defined.

1. **(CAP1) The Number of Private Repositories by Each User** - This measure shows whether a contributor learns software development in his or her own repositories. CAP1 is expressed by how many private repositories a contributor has. This measure is the proportion of contributors that have a certain number of repositories. A certain number is based on the median of the number of repositories that a contributor has. In this measure, the median is four. At that time, this measure is expressed by the proportion of the number of contributors in a year y on contributors (i) that create (i.e., $Labor_i, r$; r for r repositories) over four repositories in a year y-1. Then, CAP1_y is given by:

$$\text{CAP1}_y = \sum_{i=1}^{n} \frac{i.year = y - 1 \wedge Labor_{i,r>4}}{i.year = y} \tag{5.1}$$

   This maximum value is 1, and the minimum value is 0.

2. **(CAP2) The Number of Available Languages by Each User** - CAP2 is expressed by how many languages a contributor can use. This measure

is calculated from the language in the project that submitted commits by contributors. It is the ratio of contributors that have a certain number of languages that contributors can use. A certain number is based on the median of the number of languages that a contributor can use. In this measure, the median is two. At that time, this measure is expressed by the proportion of the number of contributors in a year y on contributors (i) that can use (i.e., $Labor_i, r$; r for l languages) over two language in a year y-1. Then, CAP1_y is given by:

$$\text{CAP2}_y = \sum_{i=1}^{n} \frac{i.year = y - 1 \wedge Labor_{i,l>2}}{i.year = y} \tag{5.2}$$

The maximum of this value is 1, and the minimum value is 0.

3. **(CAP3) The Number of Repositories that Submitted Issues by Each User** - CAP3 is expressed by how many repositories a contributor contributed to issues. This measure is the ratio of contributors that have a certain number of repositories that contributors contribute to issues. A certain number is based on the median of the number of repositories that a contributor contributes to issues. In this measure, the median is three. At that time, this measure is expressed the proportion of the number of contributors in a year y on contributors (i) that contribute (i.e., $Labor_i, r$ for r repositories) to issues over three repositories in a year y-1. Then, CAP1_y is given by:

$$\text{CAP3}_y = \sum_{i=1}^{n} \frac{i.year = y - 1 \wedge Labor_{i,r>3}}{i.year = y} \tag{5.3}$$

This maximum value is 1, and the minimum value is 0.

4. **(CAP4) The Number of Repositories that Submitted Comments by Each User** - CAP4 is expressed by how many repositories a contributor contributed to comment. This measure is the ratio of contributors that have a certain number of repositories that contributors contribute to comments. A certain number is based on the median of the number of repositories

that a contributor contributes to comments. In this measure, the median is three. At that time, this measure is expressed by the proportion of the number of contributors in a year y on contributors (i) that contribute (i.e., $Labor_i, r$ for r repositories) to comments over four repositories in a year y-1. Then, $\mathrm{CAP1}_y$ is given by:

$$\mathrm{CAP4}_y = \sum_{i=1}^{n} \frac{i.year = y - 1 \wedge Labor_{i,r>3}}{i.year = y} \tag{5.4}$$

This maximum value is 1, and the minimum value is 0.

Deployment

1. **(DEP1) Community Structural Complexity** - This indicator rates the complexity of the community of contributors and how they actively participate. An example is studying about population structure in OSS projects (i.e., S34). In detail, this study creates a population pyramid by contributors' activity periods in OSS projects. Another example is the study about the impacts of organizational factors on software quality (i.e., S59), where authors conduct observations of an in-house software development project within a large telecommunications company.

   To calculate DEP1, CCR is used as described in Chapter 4. This value is close to 0 when the number of coding contributors and the number of non-coding contributors are similar. We judge that the project has a high score, in the case where the number of coding contributors and non-coding contributors is equal. Then, in this section, DEP1 in year y is changed as follows:

$$\mathrm{DEP1}_y = 1 \ - \ |CCR_y| \tag{5.5}$$

   The maximum of this value is 1, and the minimum value is 0.

2. **(DEP2) Core Developer vs. Peripheral Developer workload equality** - These rates are in regard to the work activities of developers in the

community. Example studies include observation of the variation and specialization of the workload in an ecosystem community to identify developers' activity types and comparing the number of files that developers modify (i.e., S76).

The developer workload equality is defined as an indicator of the three factors of how community activities are performed in a project based on our previous studies: (a) workrate (defined as labor) of each contributor [73], (b) attractiveness of new contributors to a community [73], and (c) active retention of experienced members [75]. Labor is measured as community contributions within the projects. Similar to Rigby et al. [91], this thesis takes into account the contributor experience to evaluate attractiveness and retention factors. Any source code changes as contributions (i.e., comments made by contributors are ignored) are also considered. Let $Labor_{i,m}$ be the number of contributions an individual i has made in year y. This is the weighted measure of *Labor* as Workforce, where the Workforce for a contributor i in year y, who has the experience of working period e_i years, is formally defined as follows:

Using e_i as the number of years since a contributor i first joined the project, the function $WF(m)$ describes the yearly contributions of the member to the OSS project. A linear decay function is used to consider a factor of the contributor's experience (i.e., a more experienced contributor will have less weight than a newcomer to the projects in their recent contributions). The Gini coefficient of all $WF_m(e_i)$ of contributors in year y is used as the indicator of DEP2 in a year y. The Gini coefficient is a measure of statistical dispersion intended to represent the income or wealth distribution of a nation's residents. This measure is a commonly used measure for the investigation of rich-poor gap.

The original Gini coefficient is calculated from the Lorenz curve which plots the total income of a population (y-axis) and the number of populations (x-axis). The Lorenz curve $L(F)$ can be define using the probability density function $WF_y(e_i)$ as follows:

$$G = \frac{\int_0^{WF_y} e_i(WF'_y) dWF'_y}{\int_0^1 e_i(WF'_y) dWF'_y} \tag{5.6}$$

The Gini coefficient is used for the workforce. In this thesis, Workforce is compatible with income in the original Gini coefficient. The Gini coefficient ranges from 0 (complete equality) to 1 (complete inequality). When a project has a workload equality, it has a high score. Therefore, in this section, DEP2_y is changed as follows:

$$\text{DEP2}_y = 1 \ - \ G \tag{5.7}$$

The maximum of this value is 1, and the minimum value is 0.

3. **(DEP3) Contributor Participation rates** - This indicator describes contributors' participation in a particular activity such as code review (i.e., S53, S78). S53 investigates the phenomena of inactive code review developers from activities such as pull requests, while S78 is a study about review participation in code review, introducing several metrics such as purpose, history, and the prior activity of reviewers and patch authors.

   Participation Rate is the proportion of contributors working on the project throughout the year. This measure is expressed by the proportion of the number of contributors in a year y on contributors (i) that work (i.e., $Labor_{i,m}$ for m months) over three months in a year y. Then, DEP3_y is given by:

$$\text{DEP3}_y \ = \sum_{i=1}^{n} \frac{i.year = y \wedge Labor_{i,m>3}}{i.year = y} \tag{5.8}$$

   The maximum of this value is 1, and the minimum value is 0.

4. **(DEP4) Productivity rates** - This indicator measures the productivity of developers. For instance, study S24 evaluates developer performance for software-intensive products. This study interviews managers to understand from a managerial perspective how they engage in software product development activities, and to evaluate performance in large organizations. On

the other hand, study S44 is about sensing developers' emotions, progress and the use of biometric measures.

In country economics, Gross Domestic Product (GDP) measures the total of goods and services produced in a given year within the borders of a given country [84]. In this thesis, DEP4 is simply expressed as the number of commits(Commit$_{i,y}$) in a year (y) for any contributor (i). Then, DEP4$_y$ is given by:

$$\text{DEP4}_y = \sum_{i=1}^{n} Commit_{i,y} \tag{5.9}$$

This measure is not from 0 to 1 unlike other values since this measure is normalized. The maximum value is 20,066 and the minimum value is 0.

Development

1. **(DEV1) End-user participation** - This indicator is user centric, measuring from a user's perspective of skills development. For instance, there are studies that involve discovering how end-user programmers and their communities use public repositories (i.e., S22). This thesis analyzes end-user programmer communities, the characteristics of artifacts in community repositories, and how authors evolve over time. More recently, there is work that investigates factors that impact the popularity of GitHub repositories (i.e., S55). This work analyzes stars awarded to GitHub projects and identifies the popularity growth of these repositories.

   DEV1 is expressed as whether the contributor who made a fork submitted the pull requests. This measure is the proportion of the cumulative total of unique forks on the cumulative total of unique new pull requests made by the submitter in year y. Then, DEV1$_y$ is given by:

$$\text{DEV1}_y = \sum_{y} \frac{PR_y \wedge \nexists PR_{y-1}}{Forks_y} \tag{5.10}$$

   The maximum of this value is 1, and the minimum value is 0.

Know-how

1. **(KH1) Core vs. Peripheral knowledge** - This indicator explores the knowledge of core vs. peripheral developers. In this chapter, this measure is called **Issue Solving Power**. For instance, study S15 classifies developers into core and peripheral by count and network metrics. This study measures metrics related to commits and emails within the project. Another example of comparison of core contributors with peripheral contributors is a study about determining developers' expertise and roles (i.e., S20). This study analyzes bug trackers and source code repositories to characterize developers.

   Our definition of OSS knowledge is based on the evolution of product, and accumulated source code patches. One approach to measure the evolution, especially for projects that use Git version control system, is by the number of Pull Requests (PR). PR tells other contributors about changes that they wish to make to the product. Once a PR is opened, contributors can discuss and review the potential changes with the community and can add follow-up commits before the changes are merged into the product source code. After the change is merged, the PR will be closed.

   KH1 is defined as the number of completed Pull Request in year y. To add weight to more recent PRs, a weighted measure $PR_m(pr)$ is used to return the number of years that a PR (pr) took to close (i.e., $\text{PR}_y \geq 1$). Thus, PRs taking more than a year to complete have less weighting. KH1_y is formally defined as:

$$\text{KH1}_y = \sum_{pr \in \text{KH1}} \frac{1}{PR_y(pr)} \tag{5.11}$$

   It is important to note that neither the sizes nor the difficulties of Pull Requests are distinguished from other Pull Request in this thesis.

   This measure is not from 0 to 1 unlike other values since this measure is normalized. The maximum value is 9,334 and the minimum value is 0.

2. **(KH2) Knowledge loss rates** - This indicator investigates the loss of knowledge by a contributor leaving the community. For instance, study S40 quantifies and investigates how to mitigate turnover-induced knowledge loss. In detail, it quantifies the extent of abandoned source files using source code history and assesses knowledge loss by turnover. KH2 is expressed as whether a project can keep existing contributors or not. This measure is the proportion of the number of contributors who worked over a year in the previous year on the number of contributors (i) that continue to work in the next year y (over a two year period). Then, KH2_y is defined as:

$$\text{KH2}_y = \sum_{i=1}^{n} \frac{i.year = y \wedge Labor_{i,m>24}}{i.year = y-1 \wedge Labor_{i,m>12}} \tag{5.12}$$

   The maximum of this value is 1, and the minimum value is 0.

3. **(KH3) Onboarding rates** - This indicator measures the retention of contributors to a project. In our thesis, a study that explores a precursor to joining a project was identified. That study analyzed the technical factors of past experience and social factors of past connections to understand onboarding in software projects. *Onboarding rates* are expressed as whether a project can keep newcomers or not. This measure is the proportion of the number of contributors that start to work from the previous year over the number of contributors that continue to work that year y (over a one year period). Then, KH3 is defined as:

$$\text{KH3}_y = \sum_{i=1}^{n} \frac{i.year = y \wedge Labor_{i,m>24}}{i.year = y-1 \wedge Labor_{i,m<12}} \tag{5.13}$$

   The maximum of this value is 1, and the minimum value is 0.

Normalization of Metrics (using harmonic mean) To standardize the data within all the projects in this thesis, the harmonic mean was used. This mean was used to reduce bias against projects with too few contributors. For example, a small project may have an easier time balancing DEP1 is 1 when the project only has one non-coding contributor and one coding contributor. In this

thesis, however, a project with many contributors is considered healthier than a project which has few contributors.

Therefore, the harmonic mean of values that are affected by the number of contributors, DEP1, DEP3, KH2 and KH3 are considered. Also, in the metrics of DEV1, the harmonic mean of the number of forks is used. The value of the number of contributors (contrib.) is very large because the number of contributors (contrib') is normalized.

Therefore, each metric (i.e., value) is normalized by the formula $value' = \frac{2 \times value \times contrib'.}{value + contrib'}$ where we use normalized rescaling $contrib' = \frac{contrib - min(contrib)}{max(contrib) - min(contrib)}$. In this case, the maximum (contrib) is 5,659 and the min (contrib) number of contributors is 1. Also, the above definition to forks is used. In this case, then, the maximum number of (forks) is 5,775 and the min (forks) number of forks is 0.

HCI Dimensions (Step 2)

Deployment dimensions and know-how in a year y metric should be summarized, but outliers significantly affect arithmetic means. Hence, the geometric mean to each metric is taken as:

$$DepDim_y = \sqrt[4]{\mathrm{DEP1}_y \times \mathrm{DEP2}_y \times \mathrm{DEP3}_y \times \mathrm{DEP4}_y} \tag{5.14}$$

$$DevDim_y = \mathrm{DEV1}_y \tag{5.15}$$

$$KHDim_y = \sqrt[3]{\mathrm{KH1}_y \times \mathrm{KH2}_y \times \mathrm{KH3}_y} \tag{5.16}$$

Development has only one metric DEV1 since we use DEV1 is used as the DevDim directory.

HCI Project Index Score (Step 3)

To calculate the final score, an arithmetic mean for each dimension is used since the outliers' problem is solved using geometric means in the summarization of dimensions. Finally, the HCI Score is as follows:

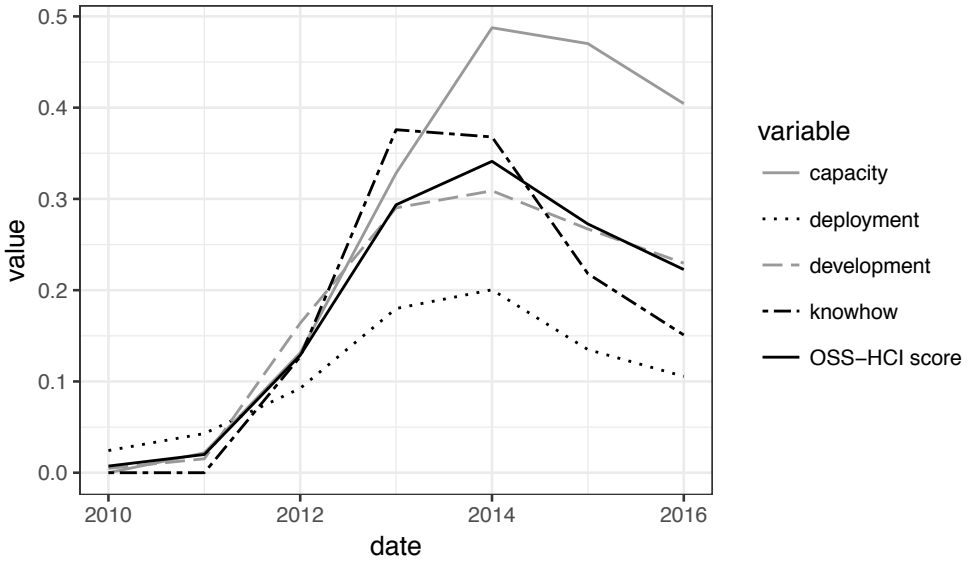


Figure 5.2: Sample of HCI Dimension Plot

$$HCI\ Score_y = \frac{DepDim_y + DevDim_y + KHDim_y}{3} \tag{5.17}$$

HCI Dimension Plot Figure 5.2 shows a sample of the HCI Dimension Plot used to describe the discovery. The HCI Dimension Plot has a period in the x-axis and each dimension and OSS-HCI score in the y-axis. In this sample, the dimensions increase together from 2010 to 2014, and decrease from 2014 to 2016. The development process of each project can be seen above.

5.3. Method

To answer the four research questions in this dissertation, a quantitative (RQ_1), case study (RQ_2) and evaluation (RQ_3, RQ_4) approaches are used. Below, the approach, a dataset method of analysis used for this thesis, is discussed.

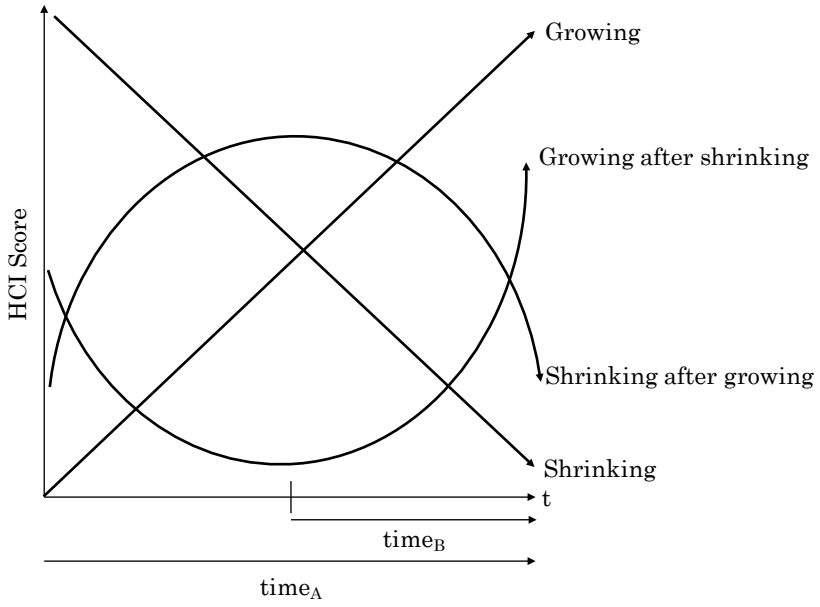


Figure 5.3: A Classification of Health Types based on the HCI Score. The Four Patterns Are Classified as: (a) Growing, (b) Growing after Shrinking, (c) Shrinking after Growing and (d) Shrinking.

5.3.1 Research Method

Research Method for RQ_1

Approach The approach in this thesis to answering RQ_1 is through a quantitative evaluation of GitHub projects by calculating the HCI score for the target projects. To do this, first the HCI index is generated and then analyzed and projects.

Dataset Collection and Preprocessing For this empirical study, 1,418 OSS projects provided by the Gousios [38] were first collected and analyzed. From this dataset, two of the related contribution activities (i.e., `commits`, `pull requests` or `comments`) for metrics calculations were selected. These 1,418 OSS projects had over pull requests every year consistently from 2010 to 2016 since there are no absolutely dead projects naturally.

Table 5.2: Classification Patterns Based on HCI Score. Four types (a) Growing, (b) Growing after Shrinking, (c) Shrinking after Growing and (d) Shrinking are shown.

Health Type	Pattern	MM vector direction
Growing	A project ascends from the first three years, also ascending from the middle three years.	$+MM_{time_A} \wedge +MM_{time_B}$
Growing after Shrinking	This project ascends from the first three years, but declines from the middle three years.	$+MM_{time_A} \wedge -MM_{time_B}$
Shrinking after Growing	This project declines from the first three years, but ascends from the middle three years.	$-MM_{time_A} \wedge +MM_{time_B}$
Shrinking	This project declines from the first three years, but also declines from the middle three years.	$-MM_{time_A} \wedge -MM_{time_B}$

HCI Score Over Time To evaluate and classify OSS health, the development of each project was calculated over a year. First, the moving mean (MM) of $HCI\ Score_y$ was calculated at triennial time intervals. As shown in Figure 5.3, the time intervals are at two points ($time_A$ and $time_B$). A is the full length of the time sequence (i.e., $time_A$), while B is the second half of the time series (i.e., $time_B$). MM_{timeA} represents the growth of last three years compared to the first three years (i.e., 2010, 2011, 2012 and 2014, 2015, 2016). Also, MM_{timeB} represents the growth of the last three years compared to the middle three years (i.e., 2012, 2013, 2014 and 2014, 2015, 2016).

As shown in Table 5.2, the four types for each project can be classified based on the MM. The direction of the MM vector is what is of interest here. A negative MM (-MM) indicates a Shrinking or Shrinking after Growing project, while a positive MM (+MM) indicates a Growing or becoming Growing project.

Therefore, to answer RQ_1, 1,418 projects were calculated and classified into the four groups and then project summary characteristics were analyzed (i.e.,

Table 5.3: Summary of the Selected Projects in Case Study 1 (snapshot as of December 2017)

project	Life Span	# of Committers	# of Commits	# of Pull Requests	# of Issues
HOMEBREW[1]	8 years and 9 months	5,591	63,881	33,606	17,046
RAILS[2]	13 years and 1 month	3,421	65,619	20,490	7,305
NETTY[3]	9 years and 4 months	299	8,550	3,618	3.862
ANGULAR.JS[4]	7 years and 11 months	1,604	8,639	7,634	8,699
PARROT[5]	16 years and 4 months	113	49,488	300	920

averages # contributors and # commits per project).

Research Method for RQ_2

Approach The approach in this thesis to answering RQ_2 is through a case study of GitHub projects from RQ_1. Then, the HCI index is generation and HCI Dimension Plots for each project analyze any patterns or insights within the project.

To answer RQ_2, the effectiveness is analyzed to show different patterns and insights that can be drawn from the HCI Dimension Plots. The projects are classified by health type.

In this thesis, two types of case studies were conducted as follows:

1. Examples of tracking of some OSS projects through the result of classification in RQ_1.

2. The use case in the OSS-HCI.

Use case In this use case, a situation is assumed in which the best PHP framework from the OSS-HCI score is chosen by users. There are many PHP frameworks, and some users may be confused as to which frameworks they should use. This case study verifies whether this OSS-HCI can help users choose the best OSS from other OSSs that have similar functions.

Dataset From the selected 1,418 projects in RQ_1, five projects were chosen as examples of each classification of health types and four projects were chosen to compare PHP frameworks. Tables 5.3 and 5.4 show detailed information for each

Table 5.4: Summary of the Selected Projects in Case Study 2 (snapshot as of January 2018)

project	Life Span	# of Committers	# of Commits	# of Pull Requests	# of Issues
ZENDFREAMWORK[6]	8 years and 10 months	688	27,056	5,631	2,133
SYMFONY[7]	8 years and 1 month	1,569	35,189	16,170	9,757
CAKEPHP[8]	12 years and 10 months	496	34,187	6,939	4.710
KOHANA[9]	9 years and 2 months	23	1,269	69	43

of these projects. It is difficult to judge whether or not these selected projects are successful. However, considering that the four projects have been active for 6 to 16 years, it may be assumed that they are representatives of typical OSS projects and their communities.

Research Method for RQ_3

Approach The approach in this thesis to answering RQ_3 is to conduct regression analysis to OSS-HCI metrics, and clarify which metrics affect the OSS-HCI score. To do this, first each metric is combined and valid metrics are chosen.

Data setting The result of OSS-HCI metrics in 1,418 projects is used. The regression analysis predicts an average of OSS-HCI scores from 2014 to 2016 as objective variables. Each metric is averaged every two years as explanatory variables. This means that the averages of each metric of 2010, 2011 and 2012, 2013 are used. Formerly 48 metrics through seven years were used, but 24 metrics are used as explanatory variables in this thesis. In this thesis, each metric such as CAP1_1011 and CAP1_1213 are shown.

Evaluation In this thesis, regression analysis is conducted in a round robin and optimal metrics and the number by Akaike's Information Criterion (AIC) are chosen. AIC is an estimator of the relative quality of statistical models for a given set of data. This measure has a lower value compared to other models, which is better.

Research Method for RQ_4

Approach The approach in this thesis to answering RQ_4 is to compare the OSS-HCI and other evaluations of OSS projects. For this study, two evaluations of OSS projects, Git Awards[10] and Libraries.io[11] were used to evaluate the OSS-HCI.

- **Libraries.io** monitors project releases, analyses each project's code, community, distribution and documentation.
- **Git Awards** ranks GitHub users by their own approach.

5.3.2 Threats to Validity

In this chapter, 1,418 projects were selected from GHTorrent. To analyze temporal transition, only consistent projects from 2010 to 2016 were selected. Therefore, really dead projects and new projects were exempt. Also, only nine projects were chosen for the case studies since the result is not generalized.

12 indicators are defined in this chapter based on our mapping study. Although these indicators are created from many primary studies, it is difficult to know whether effective indicators were chosen or not.

The 1,418 projects were analyzed based on the 12 indicators. However, it is not clear if the actual result is correct. To identify whether the result is correct or not, interviews regarding questionnaires should be sent to project's contributors.

5.4. Results

5.4.1 Project Classification Based on OSS-HCI

Table 5.5 shows the results of the classification and summary about contributors and commits. Based on the evaluation in Section 5.3.1, the projects are classified into four types. From this table, it is clear that many projects are Growing or Growing after Shrinking and 288 projects are Shrinking. Also, many projects

[10]http://git-awards.com/
[11]https://libraries.io

Table 5.5: Results of Classification for RQ_1. Note that Coverage Is Based on 1,418 Projects.

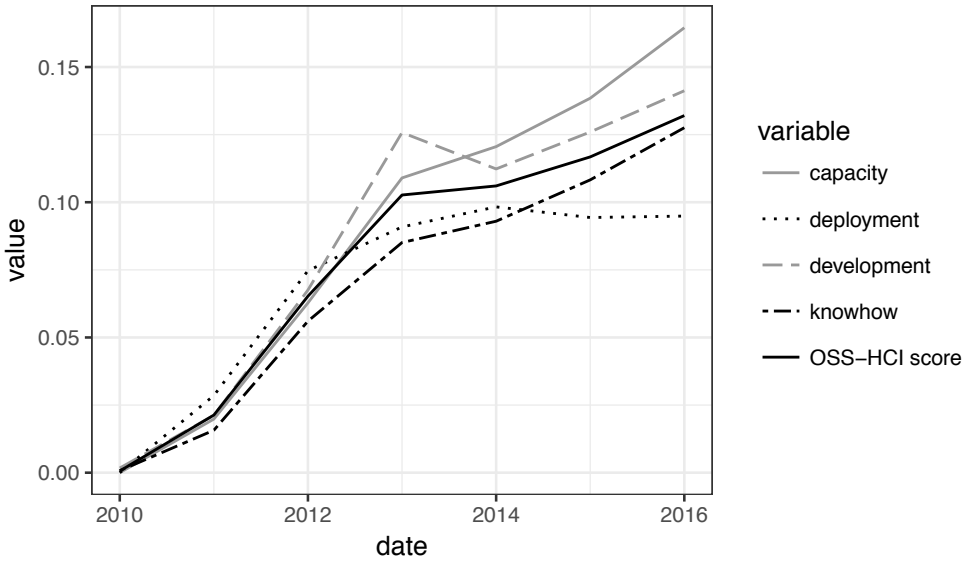
Health Type	# Projects	Coverage (%)	avg. (med.) contributors	avg. (med.) commits	avg. (med.) HCI Score
Growing	595	42.0%	72 (21)	288 (35)	0.016 (0.007)
Growing after Shrinking	594	41.9%	71 (23)	172 (26)	0.015 (0.007)
Shrinking after Growing	1	0.1%	7 (6)	23 (16)	0.001 (0.001)
Shrinking	228	16.0%	52 (20)	122 (20)	0.012 (0.006)
	1,418	100%	68 (22)	212 (28)	0.015 (0.007)

change from Growing to Growing after Shrinking and a few projects change from Shrinking to Shrinking after Growing. From this result, the project shows Shrinking, and a difficulty in Growing, except for Growing projects which can shrink (Shrinking). As mentioned in Section 5.3.1, this analysis is used only in ongoing projects since there are few Shrinking projects compared to Growing projects.

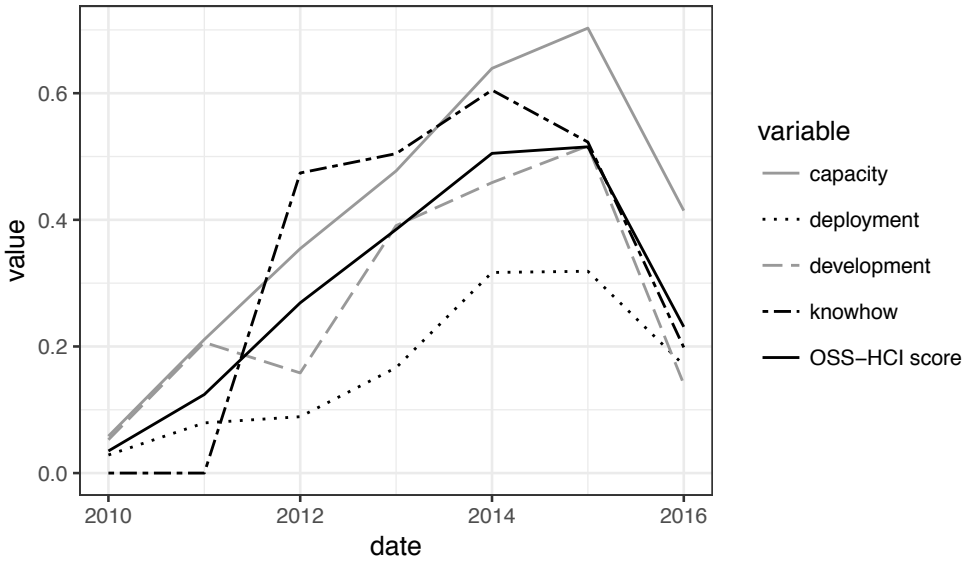
In regard to the number of contributors, Growing after Shrinking projects have many contributors, and the number of commits per person is high. Also, the value of HCI score is high compared to other projects. In this result, many Growing after Shrinking projects rapidly grow like Growing projects, and shrink after growing later. On the other hand, the Shrinking project has only a few contributors and commits.

Thus, an answer to the RQ_1 can be found; namely, "*Can projects be classified based on the OSS Human Capital Index (HCI)?*":

> *"Yes, projects could be classified into (a) Growing, (b) Growing after Shrinking, (c) Shrinking after Growing and (d) Shrinking. 42% of projects are Growing, and there are 42% Shrinking after Growing projects. On the other hand, there are many Shrinking projects compared to Shrinking after Growing projects. Once the project shrinks, it is difficult to become a Growing project, but Growing projects can shrink (Shrinking projects)."*

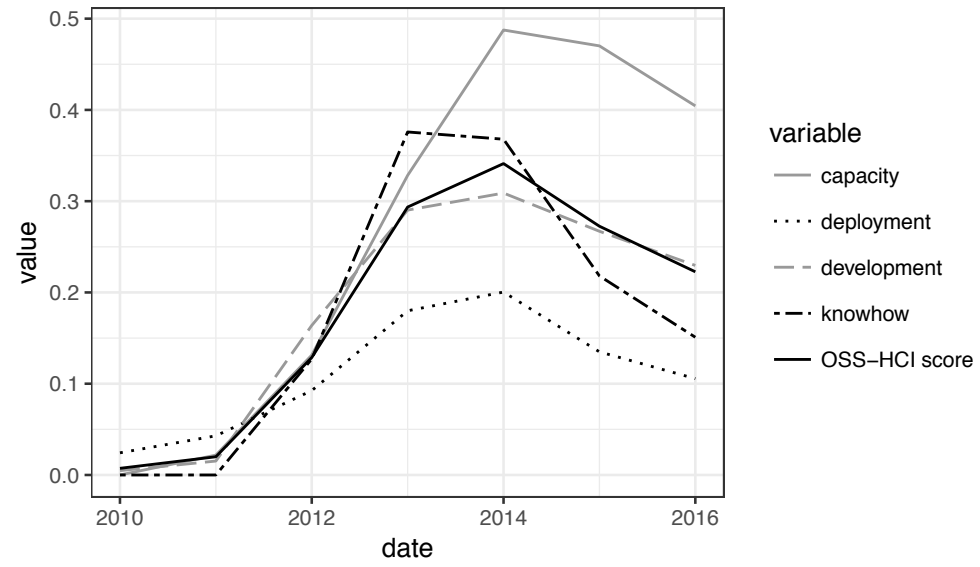


(a) NETTY

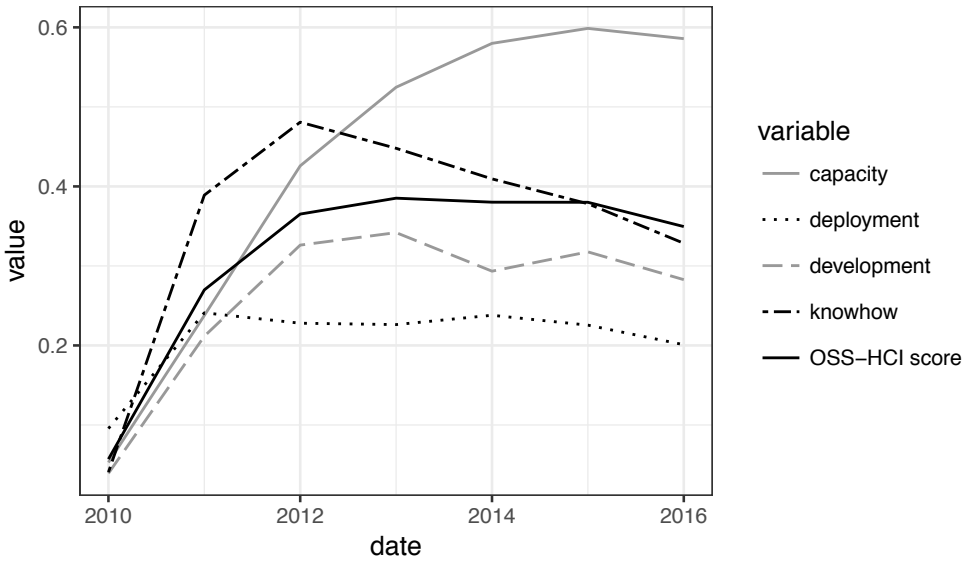


(b) HOMEBREW

Figure 5.4: HCI Dimension Plot of Growing Projects



(a) ANGULAR.JS

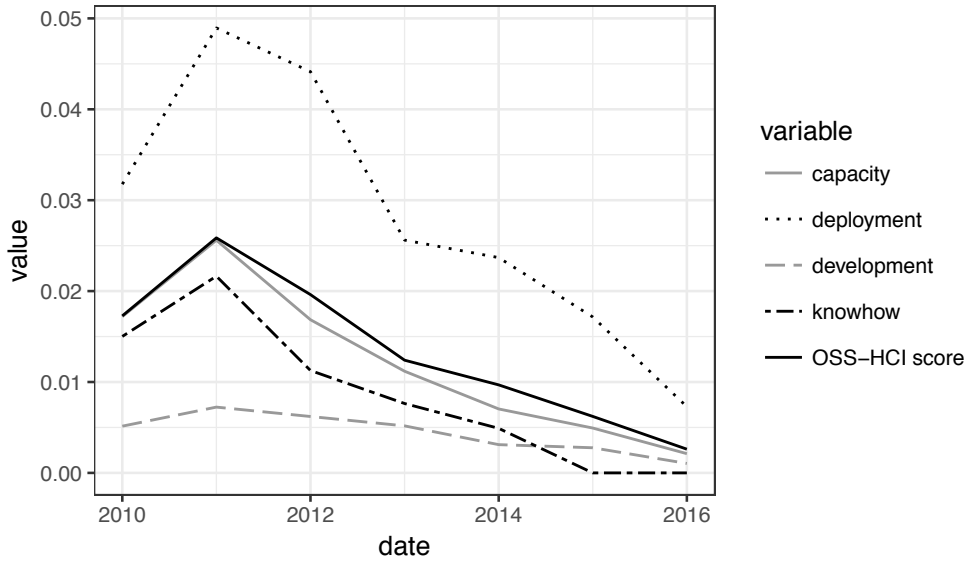


(b) RAILS

Figure 5.5: HCI Dimension Plot of Growing after Shrinking Projects

Table 5.6: Case Studies Classified by Health Type

Health type	OSS project
Growing	HOMEBREW, NETTY, CAKEPHP
Growing after Shrinking	RAILS, ANGULAR.JS, ZENDFREAMWORK, SYMFONY
Shrinking after Growing	-
Shrinking	PARROT, KOHANA



PARROT

Figure 5.6: HCI Dimension Plot of Shrinking Project

5.4.2 Tracking OSS-HCI in OSS Projects

Case Study 1 Table 5.6 shows the result of the classification of selected projects. Figure 5.4 shows the HCI dimension plot of Growing projects. Figure 5.4 (a) shows an ascendant process of the projects. This project increase capacity, deployment, development, and know-how every year. On the other hand, although HOMEBREW is classified as Growing, the shape shows it Shrinking. In particular, deployment greatly decreases since this project turned into a legacy in 2016. This is an example of an exception to a false positive. This phenomenon was caused by placing too high a value on the HCI scores in 2014 and 2015.

Figure 5.5 shows the HCI dimension plot of Growing after Shrinking projects.

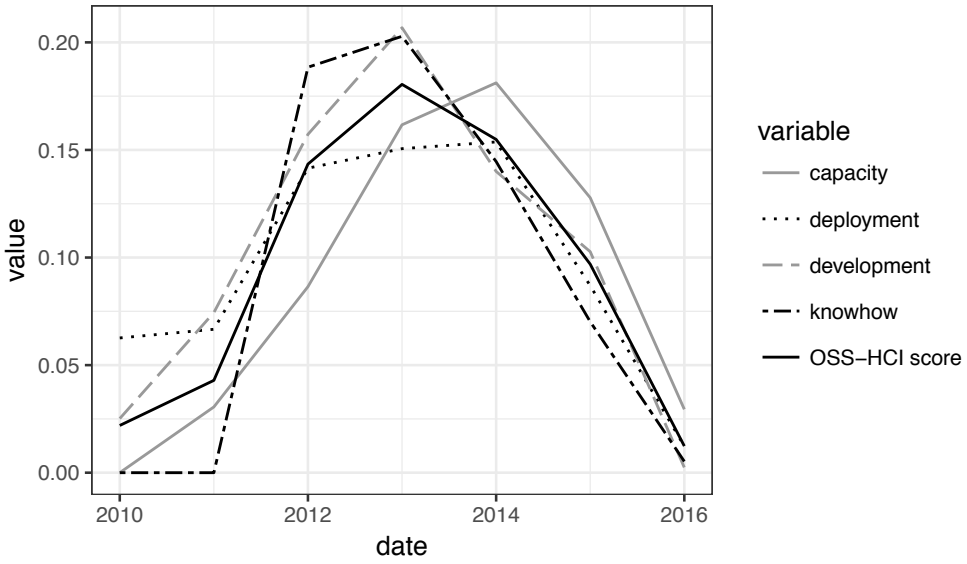
In Figure 5.5 (a), a peak can be seen. This project has the peak of capacity, development, and deployment in the same year. Also, the four dimensions of this project decrease slowly after its peak. Although RAILS is a large project, this project is classified as Growing after Shrinking project. Actually, deployment, development, and know-how have decreased in recent years in this project. These are large projects that are no longer in development but just remain in maintenance. On the other hand, this project still has the capacity to increase. Then, experienced contributors go on increasing year after year in this project. From this result, this project can be seen as keeping richly experienced contributors.

Figure 5.6 shows the HCI dimension plot of the Shrinking projects. The four dimensions decrease consistently every year in PARROT. Deployment, in particular, greatly decreases in this project. However, development increased rapidly in 2013, and this project could not continue into the next year. Even if development greatly increases, it is difficult to become a Growing project.

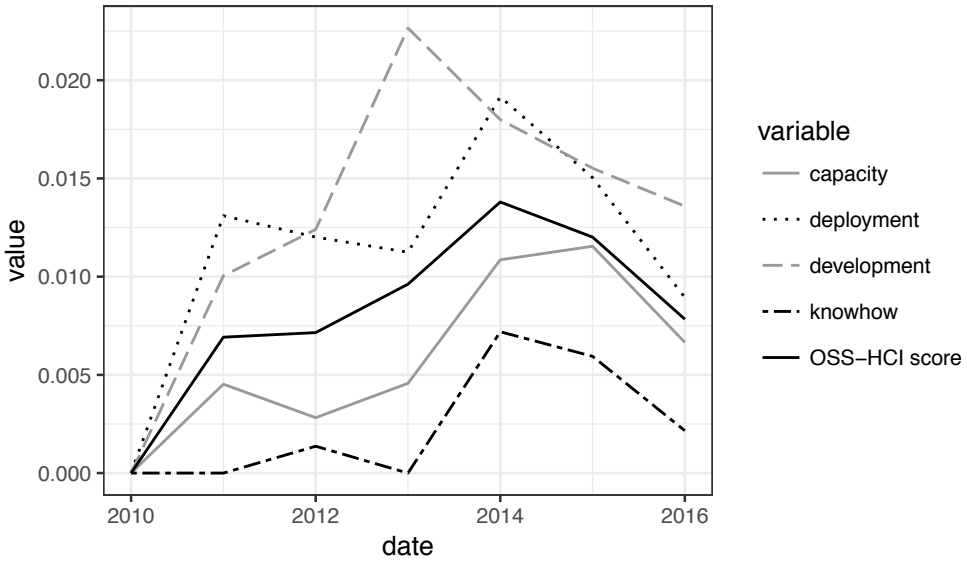
Case Study 2 Figures 5.7 and 5.8 show the HCI dimension plot of the PHP framework projects. Figure 5.7 (a), the ascendant and descendant processes of the project can be seen. This project increased in the four dimensions in the first half and decreased in the four dimensions in the last half. According to the result, this project may be in decline. In addition, Figure 5.7 (b) shows an increase and decrease of each dimension. Finally, each dimension of this project decreased compared to the peak, and each dimension increased compared to first period. From this result, it can be seen that this project did decline completely, but may do so in the future.

On the other hand, each dimension in Figure 5.8 (a) constantly increased each year. This project is developing now, so it is not clear if this project will be stable or if it will decline in the future. Although Figure 5.8 (b) illustrates a large project and is classified as a Growing after Shrinking project. In reality, deployment, development and know-how remained the same from 2012 in this project. This project has equally maintained as RAILS. On the other hand, this project capacity keeps increasing; therefore, this project will maintain enough experienced contributors, and become stable.

Based on the above results, in this case study, it can be concluded that SYM-

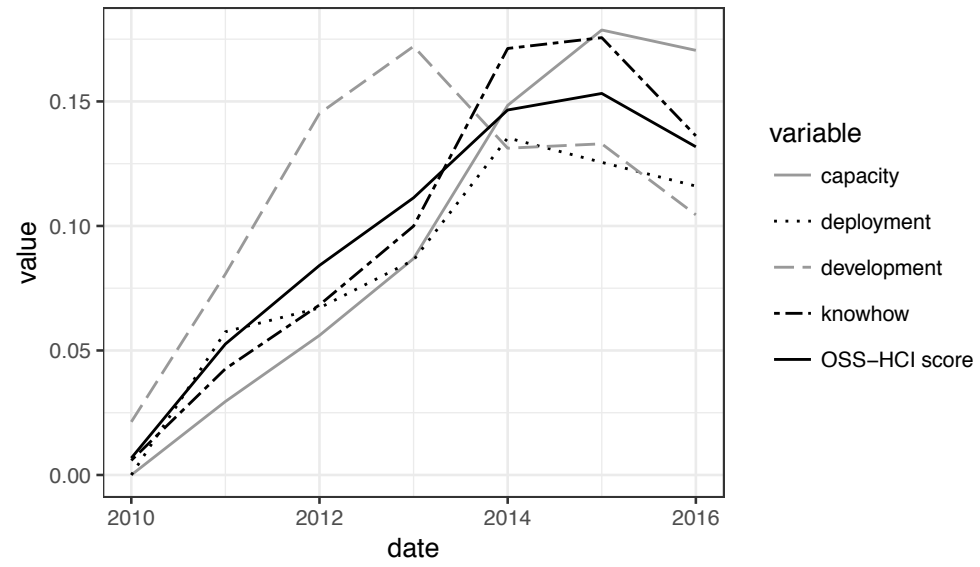


(a) ZENDFREAMWORK

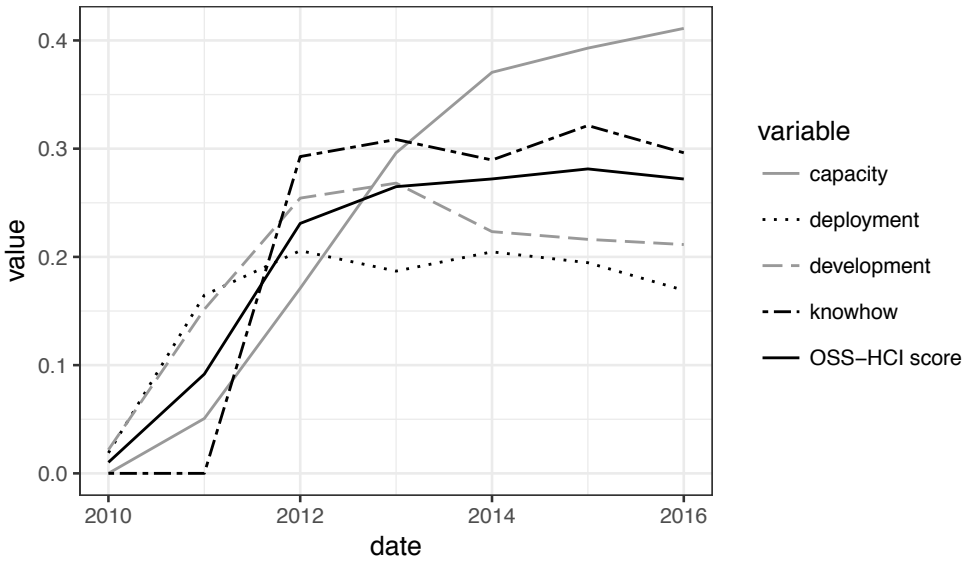


(b) KOHANA

Figure 5.7: HCI Dimension Plot of Projects of PHP Frameworks 1



(a) CAKEPHP



(b) SYMFONY

Figure 5.8: HCI Dimension Plot of Projects of PHP Frameworks 2

FONY is the best in the PHP frameworks since this project is stable. Also, CAKEPHP seems to do well in recently, it should be observed carefully in the days ahead.

Thus, an answer to RQ_2 can be found; "*How effective is HCI in identifying and tracking OSS projects?*":

- *"The HCI dimension plot can investigate the development of a project. In particular, a Growing project increases in each of the four dimensions every year. On the other hand, even if development increases, if a project cannot keep existing contributors, a project will be considered Shrinking."*
- *"The OSS-HCI can help choose the best of the OSS projects from OSSs that have similar functions. Whether the project is stable or not can be cited as an important indicator."*

5.4.3 Selecting Metrics Useful for Predicting a Future OSS-HCI Score

Through the results of regression analysis and by calculating each AIC, the metrics shown in Table 5.7 are the most valid in all combinations of metrics. This result shows CAP4_1213 is most influential metric on the OSS-HCI score. This metric and CAP4_1011 also has a negative effect on the OSS-HCI score. CAP4 is a metric of the ratio of contributors who contribute to comments at a certain number of repositories.

On the other hand, CAP2_1213 and CAP2_1011 have a positive effect on the OSS-HCI score. These metrics are the ratio of contributors who can use a certain number of languages. Also, KH3_1213 has a positive effect on the OSS-HCI score. This measure is the proportion of contributors working from the previous year and contributors working for more than one year. Also, Table 5.7 shows the p-value of each metric in the t-test. The result of the t-test (95% confidence) shows that the difference is significant.

Thus, an answer to RQ_3 can be found; namely, "*Which metrics affect OSS-HCI?*" as follows:

Table 5.7: Result of Regression Analysis

Metrics	Estimate	p-value
CAP1_1213	0.48564	0.01718
CAP2_1011	**0.32741**	0.01530
CAP2_1213	**0.69379**	8.9E-05
CAP4_1011	**-0.45655**	1.2E-05
CAP4_1213	**-0.82258**	2.6E-10
DEP1_1213	-0.18660	0.00195
DEP3_1011	-0.23390	0.01038
DEP3_1213	-0.16154	0.01110
DEP4_1011	0.04729	0.03709
DEP4_1213	0.08449	6.8E-06
DEV1_1011	-0.14915	0.02920
DEV1_1213	0.26009	7.0E-18
KH1_1213	0.11144	7.1E-08
KH3_1213	**0.65710**	4.5E-18

Table 5.8: Summary of each Evaluation

item	Git Awards	Libraries.io	OSS-HCI
# of targeted repositories	-	2,580,249	59,366,947
# of metrics	1	14	12

*"*CAP4 *has a negative effect on the OSS-HCI score. On the other hand,* CAP2 *and* KH3_*1213 have a positive effect on the OSS-HCI score."*

5.4.4 Comparing Other Evaluations of OSS Projects

Table 5.8 shows a summary of each evaluation. The targeted repositories of the Git Awards are the only active repositories at that time since they are used by the GitHub API that only collected data recently. The Git Awards has one metric that calculates the popularity of the repositories. This measure is expressed as the

number of stars in the user's repositories. Therefore, the number of repositories is n, and the popularity of repositories is defined as:

$$\sum_{i=1}^{n} stars_i \ + \ (1.0 - \frac{1.0}{n})$$

In this evaluation, homebrew is ranked first, and rails is ranked third in the ruby ranking. However, this evaluation only entertains stars in each repository since it cannot understand inside situations. On the other hand, the OSS-HCI in this dissertation can grasp inside situations by the metrics of deployment and know-how, and outside situations by the metrics of development.

Libraries.io has 14 metrics. For example, whether a project presents read me, basic information, source repository and license, and whether there are recent releases, dependent projects or repositories. Most of metrics are evaluated "yes" or "no". Certainly, these perspectives are important; however, these perspectives can easily be found by simply going to the GitHub page of a project. Also, this evaluation cannot understand the details of human factors in a project. The OSS-HCI in this work is able to consider human aspects in a project based on 12 metrics.

Thus, an answer to the RQ_4 can be found; namely, "*Is the OSS-HCI useful compared to other evaluations of OSS projects?*" as follows:

> *"Yes. This OSS-HCI can grasp outside and inside situations of human factors compared to other evaluations."*

5.5. Summary

In this study, an empirical study was conducted to measure the HCI of OSS projects. The goal was to determine whether or not the defined indexes provide meaningful and interesting insights into the relationship between the OSS HCI.

To answer RQ_1, an index was constructed using the indicators. The answer to the RQ_1; namely, "*Can projects be classified based on the OSS Human Capital Index (HCI)?*" as follows:

> *"Yes, projects could be classified into (a) Growing, (b) Growing after Shrinking, (c) Shrinking after Growing and (d) Shrinking. 42%*

of projects are Growing, and there are 42% Growing after Shrinking projects. On the other hand, there are many Shrinking projects compared to Shrinking after Growing projects. Once the project shrinks, it is difficult to become a Growing project, but Growing projects can shrink (Shrinking projects).”

To answer RQ_2, two case studies that depicted different patterns of Human Capital over time were investigated. The answer to RQ_2; namely, “*How effective is HCI in identifying and tracking OSS projects?*” as follows:

- *“The HCI dimension plot can investigate the development of a project. In particular, a Growing project increases in each of the four dimensions every year. On the other hand, even if development increases, if a project cannot keep existing contributors, a project will be considered Shrinking.”*
- *“The OSS-HCI can help choose the best of the OSS projects from OSSs that have similar functions. Whether the project is stable or not can be cited as an important indicator.”*

To answer RQ_3, a regression analysis as to which metrics affect the OSS-HCI score was conducted. The answer to the RQ_3; namely, “*Which metrics affect OSS-HCI?*” as follows:

“CAP4 has a negative effect on the OSS-HCI score. On the other hand, CAP2 and KH3_1213 have a positive effect on the OSS-HCI score.”

To answer RQ_4, the OSS-HCI in this thesis and other evaluations of OSS projects were compared. The answer to the RQ_4; namely, “*Is the OSS-HCI useful compared to other evaluations of OSS projects?*” as follows:

“Yes. This OSS-HCI can grasp outside and inside situations of human factors compared to other evaluations.”

Chapter 6

Contributions and Conclusions

In this dissertation, human activities as Human Capital in OSS development were categorized. In sum, this dissertation contributes to the following:

- A framework for OSS Human Capital that consists of four dimensions.
- Metrics (software population pyramid) from two case studies that measure community structure and contributors' retention.
- Metrics for the Human Capital Index (HCI), which identify four types of OSS projects.

To propose a framework for OSS Human Capital, a systematic mapping study was carried out and previous studies were classified into four dimensions: capital for skill attainment, deployment for the workforce, development for access to learning, and know-how for knowledge sharing. A key outcome of this mapping study is a set of indicators for constructing a HCI. These dimensions were applied to software engineering.

Based on these dimensions, several case studies were conducted. First, a study of the characteristics of contributors' activities in OSS development for capacity were conducted. To clarify the characteristics of contributors' activities, GitHub activity by each contributor to represent the capacity dimensions of Human Capital was used.

Contributors were categorized based on measures such as whether they prefer communication by coding or comments, or whether they are specialists or generalists. This dissertation shows that active software projects have various kinds of contributors characterized by different types of development activities.

Second, the community structure of OSS projects using a demographic approach was analyzed and software population pyramids that represent the deployment and know-how dimensions of Human Capital were proposed.

In this dissertation, eight case studies from four types of project in our defined two definitions CCR and NCR were investigated. Also, a prediction of a population in a software community was given. Next, a future population prediction method using a cohort component method in demography was applied to a prediction of future OSS communities.

Four types of population structures were found in OSS development communities in terms of experiences and contributions. In addition, the future population was predicted accurately using a cohort component population projection method. This method predicts a population of the next period using a survival rate calculated from past populations.

To the best of my knowledge, this is the first study that applied demography to the field of OSS research. This approach addressing OSS-related problems based on demography will hopefully bring new insights because studying population is novel in OSS research.

Finally, a HCI was constructed and an empirical study was conducted to measure the HCI of OSS projects to classify 1,418 OSS projects. As a result, it is clear that once a project shrinks (Shrinking), it is difficult for the project to grow (Growing), but Growing projects can shrink again (Shrinking). Furthermore, the HCI dimension plot can be used to investigate the development of a project. In particular, a Growing project increases in each dimension every year. On the other hand, even if development increase, if a project cannot keep its existing contributors, a project will shrink (Shrinking). In addition, a regression analysis which includes capacity and know-how metrics, was conducted it was found that these metrics affect the OSS-HCI score.

This dissertation presents a measure of contributors' activities as Workload. This measure should be important in clarifying whether a project has experienced contributors and/or casual contributors.
This dissertation provides an evidence-based comprehensive framework to help practitioners understand the Human Capital used in their projects.

Acknowledgements

本研究を進めるにあたり，多くの方々からご指導，ご協力，ご助言をいただきました．この場をお借りして，深くお礼申し上げます．

奈良先端科学技術大学院大学 情報科学研究科 松本 健一 教授には本論文の主指導教官を担当していただきました．先生には博士前期課程からの 5 年間，長きに渡り指導していただき，研究の相談や発表練習などの場面で丁寧なご指導，ご助言をいただきました．また，日々の研究だけではなく，多方面に渡り多大なご支援をいただきました．深く感謝いたします．

奈良先端科学技術大学院大学 情報科学研究科 安本 慶一 教授には本論文の副指導教官を担当していただきました．先生には博士前期課程から副指導教官を担当していただき，学内での発表において，多数のご質問とご指導をいただきました．先生からいただいた異なる分野からの鋭いご指摘や改善点は，研究を客観的に見つめなおす上で重要なものとなりました．心より感謝申し上げます．

岡山大学 自然科学研究科 門田 暁人 教授には, 本論文の審査委員をご担当いただきました．先生には博士前期課程から審査委員をご担当いただき，岡山大学出張の際にも大変お世話になりました．先生の的確なご指摘やアドバイスは，研究方針や論文執筆などで行き詰まった時の大きな助けとなりました．深く感謝申し上げます．

奈良先端科学技術大学院大学 情報科学研究科 畑 秀明 助教には本論文の副指導教官を担当していただきました．先生には研究や学生生活の大部分において多くのサポートをいただきました．特に研究においては，何度も深いディスカッションを重ねてくださいました．5 年間を通して，人間としても研究者としても，大きく成長できたと感じています．心より感謝申し上げます．

奈良先端科学技術大学院大学 情報科学研究科 Raula Gaikovina Kula 助教には本論文の副指導教官を担当していただきました．1 年という短い期間ではありましたが，先生との議論は非常に有意義で，研究の進め方やコラボレーションのあり方を根本から見つめ直すことができました．修了後も，ご指導いただいたことを忘れること無く研究活動を続けたいと思います．ありがとうございました．

奈良先端科学技術大学院大学 情報科学研究科 石尾 隆 准教授には，研究において数々のご指導をいただきました．先生が提案してくださった研究に対する様々なご意見は，研究を磨き上げる上で大きな助けとなりました．心よりお礼申し上げます．

奈良先端科学技術大学院大学 情報科学研究科 伊原 彰紀 助教には，研究や

学生生活において数々のご指導をいただきました．先生のご指導は厳しいながらも的確なものばかりで，自身を大きく成長させてくれました．心より感謝いたします．

奈良工業高等専門学校 情報工学科 上野 秀剛 准教授には，在学時から長きに渡り手広くご指導・サポートをしていただきました．研究や学生生活で悩む私への先生の献身的なアドバイスは，大学院での活動を有意義なものにするための助けとなりました．深く感謝申し上げます．

Adelaide 大学 寺本 裕美 先生には，本論文執筆にあたり多くのご助言をいただきました．先生にはプレゼンや論文の添削などで，多くのご指導をしていただきました．心より感謝いたします．

ソフトウェア工学研究室 秘書の高岸詔子さんには，学生生活や学会発表において，様々な面でサポートしていただきました．高岸さんのおかげで，学生生活における手続きを大変スムーズに行うことができ，研究活動に専念することができました．ありがとうございました．

同研究室の学生の皆様には，発表練習や論文の添削にお付き合いいただくなど，多大なるご支援とご協力をいただきました．皆様のおかげで，非常に充実した 5 年間を過ごすことができました．心より感謝を申し上げます．

最後に，私を見守ってくださった家族，苦楽を共にした博士課程の友人，卒業後・修了後も変わらず交流してくださった高専・大学・大学院の友人，趣味活動を共にしてくださった皆様との関わりは，研究に行き詰まったときの心の支えとなりました．ありがとうございました．

Appendix

A. Reference of Systematic Mapping

SMS01 [37] Abdul Rehman Gilal, Jafreezal Jaafar, Mazni Omar, Shuib Basri, and Ahmad Waqas. A rule-based model for software development team composition: Team leader role with personality types and gender classification. *Information and Software Technology*, Volume 74, pages 105–113, 2016.

SMS02 [78] Marc Palyart, Gail C. Murphy, and Vaden Masrani. A Study of Social Interactions in Open Source Component Use. *IEEE Transactions on Software Engineering*, Volume PP, Number 99, pages 1–14, 2017.

SMS03 [6] Muneera Bano and Didar Zowghi. A systematic review on the relationship between user involvement and system success. *Information and Software Technology*, Volume 58, pages 148–169, 2015.

SMS04 [16] Elizabeth Bjarnason, Kari Smolander, Emelie Engström, and Per Runeson. A theory of distances in software engineering. *Information and Software Technology*, Volume 70, pages 204–219, 2016.

SMS05 [39] Georgios Gousios, Martin Pinzger, and Arie Van Deursen. An Exploratory Study of the Pull-Based Software Development Model. *Proceedings of the 36th International Conference on Software Engineering*, pages 345–355, 2014.

SMS06 [112] Aurora Vizcaíno, Félix García, José Carlos Villar, Mario Piattini, and Javier Portillo. Applying Q-methodology to analyse the success factors in GSD. *Information and Software Technology*, Volume 55, pages 1200–1211, 2013.

SMS07 [96] Klaus-Benedikt Schultis, Christoph Elsner, and Daniel Lohmann. Architecture Challenges for Internal Software Ecosystems: A Large-Scale Industry Case Study. *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 279–288, 2014.

SMS08 [76] Marco Ortu, Bram Adams, Giuseppe Destefanis, Parastou Tourani, Michele Marchesi, and Roberto Tonelli. Are Bullies more Productive? Empirical Study of Affectiveness vs. Issue Fixing Time. *IEEE/ACM 12th Working Conference on Mining Software Repositories*, pages 303–313, 2015.

SMS09 [3] Silvia T Acuña, Marta N Gómez, Jo E Hannay, Natalia Juristo, and Dietmar Pfahl. Are team personality and climate related to satisfaction and software quality? Aggregating results from a twice replicated experiment. *Information and Software Technology*, Volume 57, pages 141–156, 2015.

SMS10 [31] Prem Devanbu, Thomas Zimmermann, and Christian Bird. Belief & Evidence in Empirical Software Engineering. *IEEE/ACM 38th International Conference on Software Engineering*, pages 108–119, 2016.

SMS11 [114] Krzysztof Wnuk, Per Runeson, Matilda Lantz, and Oskar Weijden. Bridges and barriers to hardware-dependent software ecosystem participation - A case study. *Information and Software Technology*, Volume 56, Number 11, pages 1493–1507, 2014.

SMS12 [118] Qi Xuan and Vladimir Filkov. Building It Together: Synchronous Development in OSS. *Proceedings of the 36th International Conference on Software Engineering*, pages 222–233, 2014.

SMS13 [72] Mahmood Niazi, Sajjad Mahmood, Mohammad Alshayeb, Mohammed Rehan Riaz, Kanaan Faisal, Narciso Cerpa, Siffat Ullah Khan, and Ita Richardson. Challenges of project management in global software development: A client-vendor analysis. *Information and Software Technology*, Volume 80, pages 1–19, 2016.

SMS14 [43] Hideaki Hata, Taiki Todo, Saya Onoue, and Kenichi Matsumoto. Characteristics of Sustainable OSS Projects : A TAheoretical and Empirical Study. *Proceedings of the Eighth International Workshop on Cooperative and Human Aspects of Software Engineering*, pages 15–21, 2015.

SMS15 [48] Mitchell Joblin, Sven Apel, Claus Hunsen, and Wolfgang Mauerer. Classifying Developers into Core and Peripheral: An Empirical Study on Count and Network Metrics. *IEEE/ACM 39th International Conference on Software Engineering*, pages 164–174, 2017.

SMS16 [62] Sherlock A Licorish and Stephen G Macdonell. Communication and personality profiles of global software developers. *Information and Software Technology*, Volume 64, pages 113–131, 2015.

SMS17 [42] Anja Guzzi, Alberto Bacchelli, Michele Lanza, Martin Pinzger, and Arie Van Deursen. Communication in open source software development mailing lists. *IEEE International Working Conference on Mining Software Repositories*, pages 277–286, 2013.

SMS18 [10] G R Bergersen, D I K Sjoberg, and T Dyba. Construction and Validation of an Instrument for Measuring Programming Skill. *IEEE Transactions on Software Engineering*, Volume 40, Number 12, pages 1163–1184, 2014.

SMS19 [22] Marcelo Cataldo and James D. Herbsleb. Coordination breakdowns and their impact on development productivity and software failures. *IEEE Transactions on Software Engineering*, Volume 39, Number 3, pages 343–360, 2013.

SMS20 [12] Pamela Bhattacharya, Iulian Neamtiu, and Michalis Faloutsos. Determining developers' expertise and role: A graph hierarchy-based approach. *Proceedings - 30th International Conference on Software Maintenance and Evolution*, pages 11–20, 2014.

SMS21 [21] Casey Casalnuovo, Bogdan Vasilescu, Premkumar Devanbu, and Vladimir Filkov. Developer onboarding in github: The role of prior social links and language experience. In *Proceedings of the 10th Joint Meeting on Foundations of Software Engineering*, pages 817–828, 2015.

SMS22 [102] T Stolee, Kathryn, Sebastian Elbaum, and Anita Sarma. Discovering how end-user programmers and their communities use public repositories: A study on Yahoo! Pipes. *Information and Software Technology*, Volume 55, pages 1289–1303, 2013.

SMS23 [86] Daryl Posnett, Raissa D 'souza, Premkumar Devanbu, and Vladimir Filkov. Dual Ecological Measures of Focus in Software Development. *35th International Conference on Software Engineering*, pages 452–461, 2013.

SMS24 [23] Stefan Cedergren and Stig Larsson. Evaluating performance in the development of software-intensive products. *Information and Software Technology*, Volume 56, pages 516–526, 2014.

SMS25 [7] Tobias Baum, Olga Liskin, Kai Niklas, and Kurt Schneider. Factors Influencing Code Review Processes in Industry. *Proceedings of the 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 85–96, 2016.

SMS26 [85] David Piorkowski, Austin Z Henley, Tahmid Nabi, Scott D Fleming, Christopher Scaffidi, and Margaret Burnett. Foraging and Navigations, Fundamentally: Developers' Predictions of Value and Cost. *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 97–108, 2016.

SMS27 [50] Mitchell Joblin, Wolfgang Mauerer, Sven Apel, Janet Siegmund, and Dirk Riehle. From Developer Networks to Verified Communities: A Fine-Grained Approach. *IEEE/ACM 37th IEEE International Conference on Software Engineering*, pages 563–573, 2015.

SMS28 [32] Anna Filippova, Erik Trainer, and James D Herbsleb. From Diversity by Numbers to Diversity as Process: Supporting Inclusiveness in Software Development Teams with Brainstorming. *IEEE/ACM 39th International Conference on Software Engineering*, pages 152–163, 2017.

SMS29 [79] Sebastiano Panichella, Gabriele Bavota, Massimiliano Di Penta, Gerardo Canfora, and Giuliano Antoniol. How developers' collaborations identified from different sources tell us about code changes. *Proceedings - 30th International Conference on Software Maintenance and Evolution*, pages 251–260, 2014.

SMS30 [65] Wanwangying Ma, Lin Chen, Xiangyu Zhang, Yuming Zhou, and Baowen Xu. How do Developers Fix Cross-project Correlated Bugs? A case study on the GitHub scientific Python ecosystem. *IEEE/ACM 39th International Conference on Software Engineering*, pages 381–392, 2017.

SMS31 [99] Jefferson O. Silva, Igor Wiese, Daniel German, Igor Steinmacher, and Marco A Gerosa. How Long and How Much : What to Expect from Summer of Code Participants ? *Proceedings of 33rd International Conference on Software Maintenance and Evolution*, pages 69–79, 2017.

SMS32 [107] Jason Tsay, Laura Dabbish, and James Herbsleb. Influence of Social and Technical Factors for Evaluating Contribution in GitHub. *Proceedings of the 36th International Conference on Software Engineering*, pages 356–366, 2014.

SMS33 [34] Santiago Gala-Pérez, Gregorio Robles, Jesús M González-Barahona, and Israel Herraiz. Intensive Metrics for the Study of the Evolution of Open Source Projects: Case Studies from Apache Software Foundation Projects. *10th Working Conference on Mining Software Repositories*, pages 159–168, 2013.

SMS34 [75] Saya Onoue, Hideaki Hata, Akito Monden, and Kenichi Matsumoto. Investigating and projecting population structures in open source software projects: A case study of projects in GitHub. *IEICE Transactions on Information and Systems*, Volume E99D, Number 5, pages 1304–1315, 2016.

SMS35 [108] Jason Tsay, Laura Dabbish, and James Herbsleb. Let's Talk About It: Evaluating Contributions through Discussion in GitHub. *Proceedings*

of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering, pages 144–154, 2014.

SMS36 [46] Slinger Jansen. Measuring the health of open source software ecosystems: Beyond the scope of project health. *Information and Software Technology*, Volume 56, Number 11, pages 1508–1519, 2014.

SMS37 [51] Eirini Kalliamvakou, Daniela Damian, Kelly Blincoe, Leif Singer, and Daniel M German. Open Source-Style Collaborative Development Practices in Commercial Projects Using GitHub. *IEEE/ACM 37th IEEE International Conference on Software Engineering*, pages 574–585, 2015.

SMS38 [33] Denae Ford, Justin Smith, Philip J Guo, and Chris Parnin. Paradise Unplugged: Identifying Barriers for Female Participation on Stack Overflow. *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 846–857, 2016.

SMS39 [117] Xin Xia, David Lo, Lingfeng Bao, Abhishek Sharma, and Shanping Li. Personality and Project Success : Insights from a Large-Scale Study with Professionals. *IEEE International Conference on Software Maintenance and Evolution*, pages 318–328, 2017.

SMS40 [92] Peter C Rigby, Yue Cai Zhu, Samuel M Donadelli, and Audris Mockus. Quantifying and Mitigating Turnover-Induced Knowledge Loss: Case Studies of Chrome and a project at Avaya. *IEEE/ACM 38th International Conference on Software Engineering*, pages 1006–1016, 2016.

SMS41 [111] J M Verner, O P Brereton, B A Kitchenham, M Turner, and M Niazi. Risks and risk mitigation in global software development: A tertiary study. *Information and Software Technology*, pages 54–78, 2014.

SMS42 [35] Mohammad Gharehyazie and Vladimir Filkov. Tracing distributed collaborative development in apache software foundation projects. *Empirical Software Engineering*, Volume 22, Number 4, pages 1795–1830, 2017.

SMS43 [27] Ricardo M Czekster, Paulo Fernandes, Lucelene Lopes, Afonso Sales, Alan R Santos, and Thais Webber. Stochastic Performance Analysis of Global Software Development Teams. *ACM Transactions on Software Engineering and Methodology*, Volume 25, Number 3, pages 26:1–26:32, 2016.

SMS44 [68] Sebastian C Müller and Thomas Fritz. Stuck and Frustrated or In Flow and Happy: Sensing Developers' Emotions and Progress. *IEEE/ACM 37th IEEE International Conference on Software Engineering*, pages 688–699, 2015.

SMS45 [93] Julia Rubin and Martin Rinard. The Challenges of Staying Together While Moving Fast: An Exploratory Study. *IEEE/ACM 38th International Conference on Software Engineering*, pages 982–993, 2016.

SMS46 [87] Paul Ralph and Paul Kelly. The Dimensions of Software Engineering Success. *Proceedings of the 36th International Conference on Software Engineering*, pages 24–35, 2014.

SMS47 [100] Arjumand Bano Soomro, Norsaremah Salleh, Emilia Mendes, John Grundy, Giles Burch, and Azlin Nordin. The effect of software engineers' personality traits on team climate and performance: A Systematic Literature Review. *Information and Software Technology*, Volume 73, pages 52–65, 2016.

SMS48 [71] Anh Nguyen-Duc, Daniela S Cruzes, and Reidar Conradi. The impact of global dispersion on coordination, team performance and software quality – a systematic literature review. *Information and Software Technology*, Volume 57, pages 277–294, 2015.

SMS49 [54] Youngsoo Kim and Lingxiao Jiang. The Learning Curves in Open-Source Software (OSS) Development Network. *Proceedings of the 36th International Conference on Software Engineering*, pages 41:41–41:48, 2014.

SMS50 [29] Daniela Damian, Remko Helms, Irwin Kwan, Sabrina Marczak, and Benjamin Koelewijn. The Role of Domain Knowledge and Cross-Functional

Communication in Socio-Technical Coordination. *35th International Conference on Software Engineering*, pages 442–451, 2013.

SMS51 [69] Kivanç Muşlu, Christian Bird, Nachiappan Nagappan, and Jacek Czerwonka. Transition from Centralized to Decentralized Version Control Systems: A Case Study on Reasons, Barriers, and Outcomes. *IEEE/ACM 36th IEEE International Conference on Software Engineering*, pages 334–344, 2014.

SMS52 [83] Shaun Phillips, Thomas Zimmermann, and Christian Bird. Understanding and Improving Software Build Teams. *Proceedings of the 36th International Conference on Software Engineering*, pages 735–744, 2014.

SMS53 [47] Jing Jiang, David Lo, Xinyu Ma, Fuli Feng, and Li Zhang. Understanding inactive yet available assignees in GitHub. *Information and Software Technology*, Volume 91, pages 44–55, 2017.

SMS54 [61] Sherlock A Licorish and Stephen G Macdonell. Understanding the attitudes, knowledge sharing behaviors and task performance of core developers: A longitudinal study. *Information and Software Technology*, Volume 56, Number 12, pages 1578 – 1596, 2014.

SMS55 [18] Hudson Borges, Andre Hora, and Marco Tulio Valente. Understanding the factors that impact the popularity of GitHub repositories. *Proceedings - 2016 IEEE International Conference on Software Maintenance and Evolution*, pages 334–344, 2016.

SMS56 [59] Amanda Lee, Jeffrey C Carver, and Amiangshu Bosu. Understanding the Impressions, Motivations, and Barriers of One Time Code Contributors to FLOSS Projects: A Survey. *IEEE/ACM 39th International Conference on Software Engineering*, pages 187–197, 2017.

SMS57 [17] Kelly Blincoe, Jyoti Sheoran, Sean Goggins, Eva Petakovic, and Daniela Damian. Understanding the popular users: Following, affiliation in-

fluence and leadership on GitHub. *Information and Software Technology*, Volume 70, pages 30–39, 2016.

SMS58 [123] Mansooreh Zahedi and Muhammad Ali Babar. Why does site visit matter in global software development: A knowle dge-base d perspective. *Information and Software Technology*, Volume 80, pages 36–56, 2016.

SMS59 [58] Mathieu Lavallée and Pierre N Robillard. Why Good Developers Write Bad Code: An Observational Case Study of the Impacts of Organizational Factors on Software Quality. *IEEE/ACM 37th IEEE International Conference on Software Engineering*, pages 677–687, 2015.

SMS60 [24] Jailton Coelho and Marco Tulio Valente. Why Modern Open Source Projects Fail. *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, pages 186–196, 2017.

SMS61 [40] Georgios Gousios, Margaret-Anne Storey, and Alberto Bacchelli. Work Practices and Challenges in Pull-Based Development: The Contributor's Perspective. *IEEE/ACM 38th International Conference on Software Engineering*, pages 285–296, 2016.

SMS62 [41] Georgios Gousios, Andy Zaidman, Margaret-Anne Storey, and Arie Van Deursen. Work Practices and Challenges in Pull-Based Development: The Integrator's Perspective. *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, pages 358–368, 2015.

SMS63 [103] Margaret Anne Storey, Alexey Zagalsky, Fernando Figueira Filho, Leif Singer, and Daniel M. German. How Social and Communication Channels Shape and Challenge a Participatory Culture in Software Development. *IEEE Transactions on Software Engineering*, Volume 43, Number 2, pages 185–204, 2017.

SMS64 [4] Bram Adams, Ryan Kavanagh, Ahmed E. Hassan, and Daniel M. German. An empirical study of integration activities in distributions of open source software. *Empirical Software Engineering*, Volume 21, Number 3, pages 960–1001, 2016.

SMS65 [60] Per Lenberg, Lars Göran Wallgren Tengberg, and Robert Feldt. An initial analysis of software engineers' attitudes towards organizational change. *Empirical Software Engineering*, Volume 22, Number 4, pages 2179–2205, 2017.

SMS66 [53] David Kavaler and Vladimir Filkov. Stochastic actor-oriented modeling for studying homophily and social influence in OSS projects. *Empirical Software Engineering*, Volume 22, Number 1, pages 407–435, 2017.

SMS67 [113] Yi Wang and David Redmiles. Cheap talk, cooperation, and trust in global software engineering. *Empirical Software Engineering*, Volume 21, Number 6, pages 2233–2267, 2016.

SMS68 [1] Ulrike Abelein, Barbara Paech, Daniela Damian, U Abelein, and B Paech. Understanding the Influence of User Participation and Involvement on System Success – A Systematic Mapping Study. *Empirical Software Engineering*, Volume 20, pages 28–81, 2015.

SMS69 [36] Mohammad Gharehyazie, Daryl Posnett, Bogdan Vasilescu, Vladimir Filkov, Yann-Gaël Guéhéneuc, Tom Mens, M Gharehyazie, D Posnett, V Filkov, and B Vasilescu. Developer initiation and social interactions in OSS: A case study of the Apache Software Foundation. *Empirical Software Eng*, Volume 20, pages 1318–1353, 2015.

SMS70 [49] Mitchell Joblin, Sven Apel, and Wolfgang Mauerer. Evolutionary trends of developer coordination: a network approach. *Empirical Software Engineerings*, Volume 22, Number 4, pages 2050–2094, 2017.

SMS71 [95] Ingo Scholtes, Pavlin Mavrodiev, and Frank Schweitzer. From Aristotle to Ringelmann: a large-scale analysis of team productivity and coordination in Open Source Software projects. *Empirical Software Engineering*, Volume 21, Number 2, pages 642–683, 2016.

SMS72 [122] Alexey Zagalsky, Daniel M. German, Margaret Anne Storey, Carlos Gómez Teshima, and Germán Poo-Caamaño. How the R community creates and curates knowledge: an extended study of stack overflow and mailing lists. *Empirical Software Engineering*, pages 1–34, 2017.

SMS73 [8] Olga Baysal, Oleksii Kononenko, Reid Holmes, and Michael W. Godfrey. Investigating technical and non-technical factors influencing modern code review. *Empirical Software Engineering*, Volume 21, Number 3, pages 932–959, 2016.

SMS74 [5] Özlem Albayrak and Jeffrey C Carver. Investigation of individual factors impacting the effectiveness of requirements inspections: a replicated experiment. *Empirical Software Engineering*, Volume 19, pages 241–266, 2014.

SMS75 [11] Nicolas Bettenburg, Ahmed E Hassan, Bram Adams, Daniel M German, N Bettenburg, A E Hassan, B Adams, and D M German. Management of community contributions A case study on the Android and Linux software ecosystems. *Empirical Software Engineering*, Volume 20, pages 525–289, 2015.

SMS76 [110] Bogdan Vasilescu, Alexander Serebrenik, Mathieu Goeminne, and Tom Mens. On the variation and specialisation of workload—A case study of the GNOME ecosystem community. *Empirical Software Engineering*, Volume 19, pages 955–1008, 2014.

SMS77 [77] Cristina Palomares, Carme Quer, and Xavier Franch. Requirements reuse and requirement patterns: a state of the practice survey. *Empirical Software Engineering*, pages 1–44, 2017.

SMS78 [106] Patanamon Thongtanunam, Shane McIntosh, Ahmed E. Hassan, and Hajimu Iida. Review participation in modern code review: An empirical study of the android, Qt, and OpenStack projects. *Empirical Software Engineering*, Volume 22, Number 2, pages 768–817, 2017.

B. OSS-HCI Scores for Case Studies

Table 6.1: OSS-HCI Scores in NETTY

date	2010	2011	2012	2013	2014	2015	2016
# of Contributors	4	62	252	448	424	482	577
CAP1	0.00000	0.02033	0.06404	0.10593	0.11848	0.13429	0.16145
CAP2	0.00106	0.02033	0.06725	0.11700	0.12255	0.14025	0.16512
CAP3	0.00000	0.01968	0.05691	0.10392	0.11984	0.13821	0.16436
CAP4	0.00000	0.01902	0.06341	0.10954	0.12143	0.14123	0.16722
DEP1	0.00000	0.02121	0.07811	0.12228	0.12736	0.13984	0.15914
DEP2	0.25179	0.10846	0.06486	0.03901	0.03809	0.03857	0.03422
DEP3	0.00106	0.01968	0.06575	0.09938	0.10000	0.10237	0.12265
DEP4	0.01824	0.01460	0.09349	0.14383	0.19237	0.14353	0.12135
DEV1	0.00172	0.02119	0.06753	0.12577	0.11232	0.12597	0.14122
KH1	0.00039	0.00849	0.02658	0.03712	0.05748	0.07666	0.09618
KH2	0.00106	0.02133	0.08319	0.14259	0.13281	0.14660	0.16891
KH3	0.00106	0.02133	0.07958	0.11643	0.10530	0.11294	0.12777
Capacity	0.00000	0.01983	0.06279	0.10899	0.12057	0.13847	0.16453
Deployment	0.00000	0.02852	0.07470	0.09087	0.09828	0.09435	0.09489
Development	0.00172	0.02119	0.06753	0.12577	0.11232	0.12597	0.14122
Know-how	0.00076	0.01569	0.05604	0.08510	0.09298	0.10827	0.12756
OSS-HCI Score	0.00062	0.02131	0.06526	0.10268	0.10604	0.11677	0.13205

Table 6.2: OSS-HCI Scores in HOMEBREW

date	2010	2011	2012	2013	2014	2015	2016
# of Contributors	410	1918	2472	3015	5659	5470	1661
CAP1	0.10585	0.31244	0.38803	0.47292	0.61503	0.66791	0.40340
CAP2	0.06469	0.24114	0.40164	0.52203	0.69893	0.74719	0.42350
CAP3	0.04638	0.16190	0.30832	0.44296	0.60133	0.67430	0.41019
CAP4	0.03647	0.16250	0.32857	0.47435	0.64617	0.72504	0.42152
DEP1	0.08759	0.25606	0.36021	0.42129	0.87762	0.86503	0.42050
DEP2	0.52338	0.36594	0.30583	0.28856	0.07879	0.05908	0.07330
DEP3	0.05825	0.19204	0.20826	0.20428	0.17304	0.20202	0.10074
DEP4	0.00025	0.00219	0.00274	0.03095	0.84247	1.00000	0.27260
DEV1	0.05269	0.20638	0.15819	0.39081	0.45880	0.51679	0.14046
KH1	0.06925	0.39744	0.50544	0.58109	0.72061	1.00000	0.26248
KH2	0.00000	0.00000	0.54622	0.59815	0.78662	0.67795	0.27760
KH3	0.00000	0.40861	0.38572	0.36959	0.39046	0.21084	0.10727
Capacity	0.05834	0.21100	0.35447	0.47724	0.63929	0.70281	0.41457
Deployment	0.02856	0.07926	0.08905	0.16650	0.31686	0.31877	0.17057
Development	0.05269	0.20638	0.15819	0.39081	0.45880	0.51679	0.14046
Know-how	0.00000	0.00000	0.47399	0.50458	0.60489	0.52286	0.19846
OSS-HCI Score	0.03490	0.12416	0.26892	0.38478	0.50496	0.51531	0.23101

Table 6.3: OSS-HCI Scores in ANGULAR.JS

date	2010	2011	2012	2013	2014	2015	2016
# of Contributors	23	78	834	3910	5073	3854	2204
CAP1	0.00000	0.02461	0.14615	0.32267	0.46532	0.44461	0.39536
CAP2	0.00000	0.02010	0.12129	0.32417	0.47880	0.45989	0.40184
CAP3	0.00000	0.02010	0.12768	0.32417	0.48197	0.46935	0.39971
CAP4	0.00000	0.02245	0.13144	0.34243	0.52617	0.50917	0.42120
DEP1	0.00769	0.02677	0.18904	0.41966	0.46421	0.31737	0.27372
DEP2	0.17224	0.08444	0.07046	0.13764	0.09236	0.08091	0.05047
DEP3	0.00770	0.02607	0.12045	0.19693	0.15849	0.11149	0.07614
DEP4	0.03419	0.05751	0.04555	0.09215	0.23747	0.11582	0.11801
DEV1	0.00476	0.01526	0.16385	0.29024	0.30889	0.26702	0.22958
KH1	0.00921	0.02524	0.04542	0.20132	0.26865	0.14206	0.09496
KH2	0.00000	0.00000	0.25204	0.64719	0.58620	0.43644	0.33687
KH3	0.00000	0.02619	0.17909	0.40742	0.31618	0.16733	0.10744
Capacity	0.00000	0.02174	0.13133	0.32826	0.48754	0.47016	0.40441
Deployment	0.02430	0.04290	0.09246	0.17993	0.20043	0.13494	0.10555
Development	0.00476	0.01526	0.16385	0.29024	0.30889	0.26702	0.22958
Know-how	0.00000	0.00000	0.12704	0.37582	0.36789	0.21810	0.15091
OSS-HCI Score	0.00727	0.01998	0.12867	0.29356	0.34119	0.27256	0.22261

Table 6.4: OSS-HCI Scores in RAILS

date	2010	2011	2012	2013	2014	2015	2016
# of Contributors	204	2519	3321	3654	3404	3249	3022
CAP1	0.06496	0.33234	0.43980	0.50577	0.56461	0.58609	0.57927
CAP2	0.05551	0.18493	0.38051	0.50129	0.57757	0.60477	0.59483
CAP3	0.04309	0.21952	0.43243	0.53029	0.57310	0.58357	0.56851
CAP4	0.05016	0.23665	0.45434	0.56373	0.60505	0.62109	0.60124
DEP1	0.05729	0.48878	0.50201	0.54031	0.55313	0.55611	0.51204
DEP2	0.08069	0.08583	0.08786	0.08131	0.07683	0.07793	0.06998
DEP3	0.06066	0.21907	0.25575	0.22095	0.20152	0.20773	0.19168
DEP4	0.29682	0.36754	0.23941	0.26986	0.37426	0.28745	0.23742
DEV1	0.03915	0.21163	0.32618	0.34188	0.29342	0.31768	0.28275
KH1	0.01436	0.22687	0.35124	0.34701	0.33365	0.33802	0.28206
KH2	0.06917	0.55106	0.67583	0.63240	0.63423	0.60106	0.54984
KH3	0.06744	0.47092	0.46817	0.40958	0.32435	0.26630	0.22856
Capacity	0.05284	0.23771	0.42582	0.52469	0.57989	0.59869	0.58582
Deployment	0.09552	0.24108	0.22796	0.22623	0.23794	0.22555	0.20095
Development	0.03915	0.21163	0.32618	0.34188	0.29342	0.31768	0.28275
Know-how	0.04062	0.38902	0.48078	0.44794	0.40944	0.37822	0.32849
OSS-HCI Score	0.05703	0.26986	0.36519	0.38519	0.38017	0.38004	0.34950

Table 6.5: OSS-HCI Scores in PARROT

date	2010	2011	2012	2013	2014	2015	2016
# of Contributors	63	81	50	33	21	15	7
CAP1	0.02017	0.02675	0.01693	0.01120	0.00703	0.00493	0.00212
CAP2	0.02050	0.02717	0.01708	0.01122	0.00704	0.00493	0.00212
CAP3	0.01297	0.02430	0.01666	0.01115	0.00703	0.00493	0.00212
CAP4	0.01629	0.02430	0.01672	0.01117	0.00703	0.00493	0.00212
DEP1	0.02159	0.02766	0.01699	0.01117	0.00704	0.00493	0.00212
DEP2	0.31923	0.34988	0.22082	0.23315	0.13149	0.16189	0.22668
DEP3	0.02115	0.02694	0.01697	0.01116	0.00697	0.00486	0.00211
DEP4	0.00698	0.02203	0.05960	0.01480	0.04879	0.02233	0.00279
DEV1	0.00514	0.00724	0.00621	0.00517	0.00311	0.00276	0.00104
KH1	0.00734	0.01341	0.00502	0.00363	0.00241	0.00021	0.00011
KH2	0.02145	0.02758	0.01697	0.01121	0.00702	0.00493	0.00211
KH3	0.02149	0.02740	0.01676	0.01093	0.00695	0.00000	0.00000
Capacity	0.01719	0.02559	0.01685	0.01118	0.00704	0.00493	0.00212
Deployment	0.03176	0.04895	0.04414	0.02561	0.02369	0.01715	0.00729
Development	0.00514	0.00724	0.00621	0.00517	0.00311	0.00276	0.00104
Know-how	0.01501	0.02164	0.01126	0.00763	0.00490	0.00000	0.00000
OSS-HCI Score	0.01728	0.02586	0.01961	0.01240	0.00968	0.00621	0.00261

Table 6.6: OSS-HCI Scores in ZENDFREAMWORK

date	2010	2011	2012	2013	2014	2015	2016
# of Contributors	104	151	545	916	749	429	87
CAP1	0.01258	0.04292	0.09092	0.15362	0.17091	0.12232	0.02932
CAP2	0.01258	0.03373	0.08555	0.15540	0.17913	0.12774	0.02935
CAP3	0.00000	0.02271	0.08083	0.16994	0.18626	0.12989	0.02951
CAP4	0.00000	0.02650	0.08884	0.16845	0.18896	0.13137	0.02962
DEP1	0.02988	0.04423	0.15365	0.27611	0.23330	0.13587	0.02611
DEP2	0.17600	0.14231	0.09299	0.07111	0.07063	0.06723	0.25786
DEP3	0.03431	0.04696	0.12703	0.18421	0.16293	0.10763	0.02285
DEP4	0.08527	0.06673	0.22107	0.14218	0.20801	0.05816	0.00015
DEV1	0.02516	0.07441	0.15725	0.20680	0.14003	0.10270	0.00241
KH1	0.00905	0.06113	0.26014	0.16358	0.10040	0.03273	0.00036
KH2	0.00000	0.00000	0.17016	0.24144	0.20519	0.12408	0.02093
KH3	0.03540	0.04912	0.15121	0.21121	0.14681	0.08504	0.01933
Capacity	0.00000	0.03055	0.08645	0.16168	0.18118	0.12778	0.02945
Deployment	0.06263	0.06664	0.14153	0.15059	0.15373	0.08696	0.01231
Development	0.02516	0.07441	0.15725	0.20680	0.14003	0.10270	0.00241
Know-how	0.00000	0.00000	0.18846	0.20281	0.14461	0.07016	0.00525
OSS-HCI Score	0.02195	0.04290	0.14342	0.18047	0.15489	0.09690	0.01236

Table 6.7: OSS-HCI Scores in KOHANA

date	2010	2011	2012	2013	2014	2015	2016
# of Contributors	6	14	9	14	32	34	20
CAP1	0.00176	0.00457	0.00282	0.00457	0.01085	0.01153	0.00665
CAP2	0.00000	0.00452	0.00282	0.00457	0.01086	0.01153	0.00663
CAP3	0.00000	0.00445	0.00282	0.00457	0.01086	0.01156	0.00667
CAP4	0.00000	0.00456	0.00282	0.00457	0.01086	0.01157	0.00667
DEP1	0.00176	0.00455	0.00281	0.00457	0.01088	0.01156	0.00665
DEP2	0.58788	0.38622	0.26885	0.41732	0.35345	0.31195	0.48658
DEP3	0.00000	0.00455	0.00282	0.00455	0.01065	0.01138	0.00661
DEP4	0.00115	0.00369	0.00977	0.00184	0.00329	0.00125	0.00030
DEV1	0.00000	0.01004	0.01240	0.02265	0.01799	0.01552	0.01359
KH1	0.00032	0.00032	0.00032	0.00075	0.00319	0.00161	0.00023
KH2	0.00000	0.00000	0.00282	0.00457	0.01087	0.01133	0.00667
KH3	0.00000	0.00458	0.00282	0.00000	0.01074	0.01148	0.00649
Capacity	0.00000	0.00453	0.00282	0.00457	0.01086	0.01155	0.00665
Deployment	0.00000	0.01310	0.01201	0.01124	0.01916	0.01504	0.00894
Development	0.00000	0.01004	0.01240	0.02265	0.01799	0.01552	0.01359
Know-how	0.00000	0.00000	0.00137	0.00000	0.00719	0.00594	0.00216
OSS-HCI Score	0.00000	0.00692	0.00715	0.00962	0.01380	0.01201	0.00784

Table 6.8: OSS-HCI Scores in CAKEPHP

date	2010	2011	2012	2013	2014	2015	2016
# of Contributors	29	133	198	317	615	795	653
CAP1	0.00923	0.04089	0.06105	0.08750	0.14266	0.16704	0.16484
CAP2	0.00000	0.02294	0.05790	0.09146	0.14958	0.18679	0.17072
CAP3	0.00000	0.02628	0.05035	0.08114	0.14958	0.17544	0.17010
CAP4	0.00000	0.03075	0.05547	0.08779	0.15201	0.18623	0.17665
DEP1	0.00000	0.03263	0.04666	0.10030	0.18728	0.21599	0.18556
DEP2	0.08251	0.08089	0.08268	0.05686	0.04402	0.04095	0.04652
DEP3	0.00976	0.04254	0.05982	0.08160	0.14110	0.15570	0.14597
DEP4	0.05008	0.09743	0.08776	0.11916	0.28860	0.18040	0.14373
DEV1	0.02133	0.08069	0.14519	0.17212	0.13123	0.13298	0.10447
KH1	0.00210	0.03823	0.07443	0.10171	0.17706	0.12832	0.10223
KH2	0.00978	0.04549	0.06676	0.10376	0.18775	0.23218	0.18811
KH3	0.00981	0.04458	0.06406	0.09459	0.15116	0.18180	0.13133
Capacity	0.00000	0.02951	0.05605	0.08689	0.14842	0.17868	0.17053
Deployment	0.00000	0.05751	0.06708	0.08629	0.13536	0.12555	0.11601
Development	0.02133	0.08069	0.14519	0.17212	0.13123	0.13298	0.10447
Know-how	0.00586	0.04264	0.06828	0.09994	0.17128	0.17562	0.13618
OSS-HCI Score	0.00680	0.05259	0.08415	0.11131	0.14657	0.15321	0.13180

Table 6.9: OSS-HCI Scores in SYMFONY

date	2010	2011	2012	2013	2014	2015	2016
# of Contributors	36	851	1457	1812	1932	1927	1918
CAP1	0.01113	0.09705	0.15190	0.25739	0.33243	0.35315	0.38206
CAP2	0.01012	0.07023	0.17066	0.27757	0.36989	0.39797	0.41088
CAP3	0.00000	0.02380	0.18421	0.31898	0.38557	0.40134	0.41735
CAP4	0.00000	0.04065	0.17904	0.33731	0.39761	0.42251	0.43605
DEP1	0.01158	0.26090	0.37474	0.40877	0.41587	0.38342	0.37618
DEP2	0.54858	0.19054	0.13215	0.08722	0.07675	0.06134	0.05174
DEP3	0.01184	0.17999	0.23619	0.22251	0.23757	0.23151	0.22311
DEP4	0.00169	0.08223	0.15324	0.15369	0.23144	0.26368	0.18868
DEV1	0.02179	0.15153	0.25431	0.26816	0.22335	0.21621	0.21141
KH1	0.00415	0.22070	0.23739	0.21198	0.19273	0.27058	0.25897
KH2	0.00000	0.00000	0.33979	0.42428	0.43771	0.44052	0.42191
KH3	0.00000	0.24117	0.31095	0.32643	0.28768	0.27847	0.23784
Capacity	0.00000	0.05068	0.17100	0.29610	0.37054	0.39291	0.41112
Deployment	0.01890	0.16469	0.20576	0.18687	0.20467	0.19466	0.16918
Development	0.02179	0.15153	0.25431	0.26816	0.22335	0.21621	0.21141
Know-how	0.00000	0.00000	0.29272	0.30849	0.28952	0.32138	0.29620
OSS-HCI Score	0.01017	0.09173	0.23095	0.26491	0.27202	0.28129	0.27198

References

[1] Ulrike Abelein, Barbara Paech, Daniela Damian, U Abelein, and B Paech. Understanding the Influence of User Participation and Involvement on System Success – A Systematic Mapping Study. *Empirical Software Engineering*, 20:28–81, 2015.

[2] Mohamed Ibrahim Abouelhoda, Stefan Kurtz, and Enno Ohlebusch. Replacing suffix trees with enhanced suffix arrays. *Journal of Discrete Algorithms*, 2(1):53 – 86, 2004.

[3] Silvia T Acuña, Marta N Gómez, Jo E Hannay, Natalia Juristo, and Dietmar Pfahl. Are team personality and climate related to satisfaction and software quality? Aggregating results from a twice replicated experiment. *Information and Software Technology*, 57:141–156, 2015.

[4] Bram Adams, Ryan Kavanagh, Ahmed E. Hassan, and Daniel M. German. An empirical study of integration activities in distributions of open source software. *Empirical Software Engineering*, 21(3):960–1001, 2016.

[5] Özlem Albayrak and Jeffrey C Carver. Investigation of individual factors impacting the effectiveness of requirements inspections: a replicated experiment. *Empirical Software Engineering*, 19:241–266, 2014.

[6] Muneera Bano and Didar Zowghi. A systematic review on the relationship between user involvement and system success. *Information and Software Technology*, 58:148–169, 2015.

[7] Tobias Baum, Olga Liskin, Kai Niklas, and Kurt Schneider. Factors Influencing Code Review Processes in Industry. *Proceedings of the 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 85–96, 2016.

[8] Olga Baysal, Oleksii Kononenko, Reid Holmes, and Michael W. Godfrey. Investigating technical and non-technical factors influencing modern code review. *Empirical Software Engineering*, 21(3):932–959, 2016.

[9] Gary Becker. Human capital. *National Bureau of Economic Research*, 1964.

[10] G R Bergersen, D I K Sjoberg, and T Dyba. Construction and Validation of an Instrument for Measuring Programming Skill. *IEEE Transactions on Software Engineering*, 40(12):1163–1184, 2014.

[11] Nicolas Bettenburg, Ahmed E Hassan, Bram Adams, Daniel M German, N Bettenburg, A E Hassan, B Adams, and D M German. Management of community contributions A case study on the Android and Linux software ecosystems. *Empirical Software Engineering*, 20:525–289, 2015.

[12] Pamela Bhattacharya, Iulian Neamtiu, and Michalis Faloutsos. Determining developers' expertise and role: A graph hierarchy-based approach. *Proceedings - 30th International Conference on Software Maintenance and Evolution*, pages 11–20, 2014.

[13] Christian Bird. Sociotechnical coordination and collaboration in open source software. *IEEE International Conference on Software Maintenance*, pages 568–573, 2011.

[14] Christian Bird, Alex Gourley, Premkumar Devanbu, Michael Gertz, and Anand Swaminathan. Mining email social networks. *International Workshop on Mining Software Repositories*, pages 137–143, 2006.

[15] Christian Bird, David Pattison, Raissa D'Souza, Vladimir Filkov, and Premkumar Devanbu. Latent Social Structure in Open Source Projects. *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 24–35, 2008.

[16] Elizabeth Bjarnason, Kari Smolander, Emelie Engström, and Per Runeson. A theory of distances in software engineering. *Information and Software Technology*, 70:204–219, 2016.

[17] Kelly Blincoe, Jyoti Sheoran, Sean Goggins, Eva Petakovic, and Daniela Damian. Understanding the popular users: Following, affiliation influence and leadership on GitHub. *Information and Software Technology*, 70:30–39, 2016.

[18] Hudson Borges, Andre Hora, and Marco Tulio Valente. Understanding the factors that impact the popularity of GitHub repositories. *Proceedings* -

2016 IEEE International Conference on Software Maintenance and Evolution, pages 334–344, 2016.

[19] Caius Brindescu, Mihai Codoban, Sergii Shmarkatiuk, and Danny Dig. How Do Centralized and Distributed Version Control Systems Impact Software Changes? *Proceedings of the 36th International Conference on Software Engineering*, pages 332–333, 2014.

[20] Raymond P L Buse and Thomas Zimmermann. Information needs for software development analytics. *Proceedings of the 34th International Conference on Software Engineering*, pages 987–996, 2012.

[21] Casey Casalnuovo, Bogdan Vasilescu, Premkumar Devanbu, and Vladimir Filkov. Developer onboarding in github: The role of prior social links and language experience. In *Proceedings of the 10th Joint Meeting on Foundations of Software Engineering*, pages 817–828, 2015.

[22] Marcelo Cataldo and James D. Herbsleb. Coordination breakdowns and their impact on development productivity and software failures. *IEEE Transactions on Software Engineering*, 39(3):343–360, 2013.

[23] Stefan Cedergren and Stig Larsson. Evaluating performance in the development of software-intensive products. *Information and Software Technology*, 56:516–526, 2014.

[24] Jailton Coelho and Marco Tulio Valente. Why Modern Open Source Projects Fail. *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, pages 186–196, 2017.

[25] S. D. Conte, H. E. Dunsmore, and V. Y. Shen. *Software Engineering Metrics and Models*. Benjamin-Cummings Publishing Co., Inc., Redwood City, CA, USA, 1986.

[26] Kevin Crowston and James Howison. Assesing the Health of Open Source Communities. *IEEE Computer*, 39(5):89–91, 2006.

[27] Ricardo M Czekster, Paulo Fernandes, Lucelene Lopes, Afonso Sales, Alan R Santos, and Thais Webber. Stochastic Performance Analysis of

Global Software Development Teams. *ACM Transactions on Software Engineering and Methodology*, 25(3):26:1–26:32, 2016.

[28] Laura Dabbish, Colleen Stuart, Jason Tsay, and Jim Herbsleb. Social Coding in GitHub: Transparency and Collaboration in an Open Software Repository. *Proceedings of the ACM 2012 Conference on Computer Supported Cooperative Work*, pages 1277–1286, 2012.

[29] Daniela Damian, Remko Helms, Irwin Kwan, Sabrina Marczak, and Benjamin Koelewijn. The Role of Domain Knowledge and Cross-Functional Communication in Socio-Technical Coordination. *35th International Conference on Software Engineering*, pages 442–451, 2013.

[30] Tom DeMarco and Tim Lister. *Peopleware: Productive Projects and Teams (3rd Edition)*. Addison-Wesley Professional, 3rd edition, 2013.

[31] Prem Devanbu, Thomas Zimmermann, and Christian Bird. Belief & Evidence in Empirical Software Engineering. *IEEE/ACM 38th International Conference on Software Engineering*, pages 108–119, 2016.

[32] Anna Filippova, Erik Trainer, and James D Herbsleb. From Diversity by Numbers to Diversity as Process: Supporting Inclusiveness in Software Development Teams with Brainstorming. *IEEE/ACM 39th International Conference on Software Engineering*, pages 152–163, 2017.

[33] Denae Ford, Justin Smith, Philip J Guo, and Chris Parnin. Paradise Unplugged: Identifying Barriers for Female Participation on Stack Overflow. *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 846–857, 2016.

[34] Santiago Gala-Pérez, Gregorio Robles, Jesús M González-Barahona, and Israel Herraiz. Intensive Metrics for the Study of the Evolution of Open Source Projects: Case Studies from Apache Software Foundation Projects. *10th Working Conference on Mining Software Repositories*, pages 159–168, 2013.

[35] Mohammad Gharehyazie and Vladimir Filkov. Tracing distributed col-

laborative development in apache software foundation projects. *Empirical Software Engineering*, 22(4):1795–1830, 2017.

[36] Mohammad Gharehyazie, Daryl Posnett, Bogdan Vasilescu, Vladimir Filkov, Yann-Gaël Guéhéneuc, Tom Mens, M Gharehyazie, D Posnett, V Filkov, and B Vasilescu. Developer initiation and social interactions in OSS: A case study of the Apache Software Foundation. *Empirical Software Eng*, 20:1318–1353, 2015.

[37] Abdul Rehman Gilal, Jafreezal Jaafar, Mazni Omar, Shuib Basri, and Ahmad Waqas. A rule-based model for software development team composition: Team leader role with personality types and gender classification. *Information and Software Technology*, 74:105–113, 2016.

[38] Georgios Gousios. The GHTorent dataset and tool suite. *IEEE International Working Conference on Mining Software Repositories*, pages 233–236, 2013.

[39] Georgios Gousios, Martin Pinzger, and Arie Van Deursen. An Exploratory Study of the Pull-Based Software Development Model. *Proceedings of the 36th International Conference on Software Engineering*, pages 345–355, 2014.

[40] Georgios Gousios, Margaret-Anne Storey, and Alberto Bacchelli. Work Practices and Challenges in Pull-Based Development: The Contributor's Perspective. *IEEE/ACM 38th International Conference on Software Engineering*, pages 285–296, 2016.

[41] Georgios Gousios, Andy Zaidman, Margaret-Anne Storey, and Arie Van Deursen. Work Practices and Challenges in Pull-Based Development: The Integrator's Perspective. *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, pages 358–368, 2015.

[42] Anja Guzzi, Alberto Bacchelli, Michele Lanza, Martin Pinzger, and Arie Van Deursen. Communication in open source software development mailing lists. *IEEE International Working Conference on Mining Software Repositories*, pages 277–286, 2013.

[43] Hideaki Hata, Taiki Todo, Saya Onoue, and Kenichi Matsumoto. Characteristics of Sustainable OSS Projects : A TAheoretical and Empirical Study. *Proceedings of the Eighth International Workshop on Cooperative and Human Aspects of Software Engineering*, pages 15–21, 2015.

[44] A. Haupt, T. Kane, and C. Haub. *PRB's Population Handbook 6th ed.* Population Reference Bureau, 2011.

[45] Andrew M. Isserman. *The Right People, the Right Rates*, pages 43–60. Journal of the American Planning Association 59.1, 1993.

[46] Slinger Jansen. Measuring the health of open source software ecosystems: Beyond the scope of project health. *Information and Software Technology*, 56(11):1508–1519, 2014.

[47] Jing Jiang, David Lo, Xinyu Ma, Fuli Feng, and Li Zhang. Understanding inactive yet available assignees in GitHub. *Information and Software Technology*, 91:44–55, 2017.

[48] Mitchell Joblin, Sven Apel, Claus Hunsen, and Wolfgang Mauerer. Classifying Developers into Core and Peripheral: An Empirical Study on Count and Network Metrics. *IEEE/ACM 39th International Conference on Software Engineering*, pages 164–174, 2017.

[49] Mitchell Joblin, Sven Apel, and Wolfgang Mauerer. Evolutionary trends of developer coordination: a network approach. *Empirical Software Engineerings*, 22(4):2050–2094, 2017.

[50] Mitchell Joblin, Wolfgang Mauerer, Sven Apel, Janet Siegmund, and Dirk Riehle. From Developer Networks to Verified Communities: A Fine-Grained Approach. *IEEE/ACM 37th IEEE International Conference on Software Engineering*, pages 563–573, 2015.

[51] Eirini Kalliamvakou, Daniela Damian, Kelly Blincoe, Leif Singer, and Daniel M German. Open Source-Style Collaborative Development Practices in Commercial Projects Using GitHub. *IEEE/ACM 37th IEEE International Conference on Software Engineering*, pages 574–585, 2015.

[52] Eirini Kalliamvakou, Georgios Gousios, Tudelftnl Kelly Blincoe, Leif Singer, Daniel M German, and Daniela Damian. The Promises and Perils of Mining GitHub. *Proceedings of the 11th Working Conference on Mining Software Repositories*, pages 92–101, 2014.

[53] David Kavaler and Vladimir Filkov. Stochastic actor-oriented modeling for studying homophily and social influence in OSS projects. *Empirical Software Engineering*, 22(1):407–435, 2017.

[54] Youngsoo Kim and Lingxiao Jiang. The Learning Curves in Open-Source Software (OSS) Development Network. *Proceedings of the 36th International Conference on Software Engineering*, pages 41:41–41:48, 2014.

[55] B. Kitchenham and S Charters. Guidelines for performing systematic literature reviews in software engineering, 2007.

[56] B a Kitchenham, S G MacDonell, L Pickard, and M J Shepperd. What accuracy statistics really measure. *IEE Proceedings - Software*, 148(3):81–85, 2001.

[57] Barbara A. Kitchenham, David Budgen, and O. Pearl Brereton. The value of mapping studies: A participantobserver case study. In *Proceedings of the 14th International Conference on Evaluation and Assessment in Software Engineering*, pages 25–33, 2010.

[58] Mathieu Lavallée and Pierre N Robillard. Why Good Developers Write Bad Code: An Observational Case Study of the Impacts of Organizational Factors on Software Quality. *IEEE/ACM 37th IEEE International Conference on Software Engineering*, pages 677–687, 2015.

[59] Amanda Lee, Jeffrey C Carver, and Amiangshu Bosu. Understanding the Impressions, Motivations, and Barriers of One Time Code Contributors to FLOSS Projects: A Survey. *IEEE/ACM 39th International Conference on Software Engineering*, pages 187–197, 2017.

[60] Per Lenberg, Lars Göran Wallgren Tengberg, and Robert Feldt. An initial analysis of software engineers' attitudes towards organizational change. *Empirical Software Engineering*, 22(4):2179–2205, 2017.

[61] Sherlock A Licorish and Stephen G Macdonell. Understanding the attitudes, knowledge sharing behaviors and task performance of core developers: A longitudinal study. *Information and Software Technology*, 56(12):1578 – 1596, 2014.

[62] Sherlock A Licorish and Stephen G Macdonell. Communication and personality profiles of global software developers. *Information and Software Technology*, 64:113–131, 2015.

[63] G. Liu and B.M. Fraumeni. Growth and stagnation in the world economy. *The World Economy: Growth or Stagnation?*, pages 429–468, 2016.

[64] Pablo Loyola and In-Young Ko. Population Dynamics in Open Source Communities : An Ecological Approach Applied to Github. *Proceedings of the 23rd International Conference on World Wide Web*, pages 993–998, 2014.

[65] Wanwangying Ma, Lin Chen, Xiangyu Zhang, Yuming Zhou, and Baowen Xu. How do Developers Fix Cross-project Correlated Bugs? A case study on the GitHub scientific Python ecosystem. *IEEE/ACM 39th International Conference on Software Engineering*, pages 381–392, 2017.

[66] Stephen MacDonell, Martin Shepperd, Barbara Kitchenham, and Emilia Mendes. How reliable are systematic reviews in empirical software engineering? *IEEE Transactions on Software Engineering*, 36(5):676–687, 2010.

[67] Y. Miyazaki, M. Terakado, K. Ozaki, and H. Nozaki. Robust regression for developing software estimation models. *Journal of Systems and Software*, 27:3–16, 1994.

[68] Sebastian C Müller and Thomas Fritz. Stuck and Frustrated or In Flow and Happy: Sensing Developers' Emotions and Progress. *IEEE/ACM 37th IEEE International Conference on Software Engineering*, pages 688–699, 2015.

[69] Kivanç Muşlu, Christian Bird, Nachiappan Nagappan, and Jacek Czerwonka. Transition from Centralized to Decentralized Version Control Sys-

tems: A Case Study on Reasons, Barriers, and Outcomes. *IEEE/ACM 36th IEEE International Conference on Software Engineering*, pages 334–344, 2014.

[70] Janine Nahapiet and Sumantra Ghoshal. Chapter 6 - social capital, intellectual capital, and the organizational advantage*. In Eric L. Lesser, editor, *Knowledge and Social Capital*, pages 119 – 157. Butterworth-Heinemann, Boston, 1998.

[71] Anh Nguyen-Duc, Daniela S Cruzes, and Reidar Conradi. The impact of global dispersion on coordination, team performance and software quality – a systematic literature review. *Information and Software Technology*, 57:277–294, 2015.

[72] Mahmood Niazi, Sajjad Mahmood, Mohammad Alshayeb, Mohammed Rehan Riaz, Kanaan Faisal, Narciso Cerpa, Siffat Ullah Khan, and Ita Richardson. Challenges of project management in global software development: A client-vendor analysis. *Information and Software Technology*, 80:1–19, 2016.

[73] Saya Onoue, Hideaki Hata, and Ken Ichi Matsumoto. A study of the characteristics of developers' activities in GitHub. *Proceedings of 5th International Workshop on Empirical Software Engineering in Practice, IWESEP' 2013*, pages 7–12, 2013.

[74] Saya Onoue, Hideaki Hata, and Kenichi Matsumoto. Software Population Pyramids : The Current and the Future of OSS Development Communities. *Proceedings - 2016 IEEE International Conference on Software Maintenance and Evolution, ICSME 2016 of 8th International Symposium on Empirical Software Engineering and Measurement*, pages 1–4, 2014.

[75] Saya Onoue, Hideaki Hata, Akito Monden, and Kenichi Matsumoto. Investigating and projecting population structures in open source software projects: A case study of projects in GitHub. *IEICE Transactions on Information and Systems*, E99D(5):1304–1315, 2016.

[76] Marco Ortu, Bram Adams, Giuseppe Destefanis, Parastou Tourani, Michele

Marchesi, and Roberto Tonelli. Are Bullies more Productive? Empirical Study of Affectiveness vs. Issue Fixing Time. *IEEE/ACM 12th Working Conference on Mining Software Repositories*, pages 303–313, 2015.

[77] Cristina Palomares, Carme Quer, and Xavier Franch. Requirements reuse and requirement patterns: a state of the practice survey. *Empirical Software Engineering*, pages 1–44, 2017.

[78] Marc Palyart, Gail C. Murphy, and Vaden Masrani. A Study of Social Interactions in Open Source Component Use. *IEEE Transactions on Software Engineering*, PP(99):1–14, 2017.

[79] Sebastiano Panichella, Gabriele Bavota, Massimiliano Di Penta, Gerardo Canfora, and Giuliano Antoniol. How developers' collaborations identified from different sources tell us about code changes. *Proceedings - 30th International Conference on Software Maintenance and Evolution*, pages 251–260, 2014.

[80] Sophie Pennec. *APPSIM - Cohort component population projections to validate and align the dynamic microsimulation model APPSIM.* National Centre for Social and Economic Modelling, 2009.

[81] Kai Petersen, Robert Feldt, Shahid Mujtaba, and Michael Mattsson. Systematic mapping studies in software engineering. In *Proceedings of the 12th International Conference on Evaluation and Assessment in Software Engineering*, pages 68–77, 2008.

[82] Kai Petersen, Sairam Vakkalanka, and Ludwik Kuzniarz. Guidelines for conducting systematic mapping studies in software engineering: An update. *Information and Software Technology*, 64:1–18, 2015.

[83] Shaun Phillips, Thomas Zimmermann, and Christian Bird. Understanding and Improving Software Build Teams. *Proceedings of the 36th International Conference on Software Engineering*, pages 735–744, 2014.

[84] Thomas Piketty. *Capital in the Twenty-First Century*, pages 43, 385–390. Éditions du Seuil, Belknap Press, 2013.

[85] David Piorkowski, Austin Z Henley, Tahmid Nabi, Scott D Fleming, Christopher Scaffidi, and Margaret Burnett. Foraging and Navigations, Fundamentally: Developers' Predictions of Value and Cost. *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 97–108, 2016.

[86] Daryl Posnett, Raissa D 'souza, Premkumar Devanbu, and Vladimir Filkov. Dual Ecological Measures of Focus in Software Development. *35th International Conference on Software Engineering*, pages 452–461, 2013.

[87] Paul Ralph and Paul Kelly. The Dimensions of Software Engineering Success. *Proceedings of the 36th International Conference on Software Engineering*, pages 24–35, 2014.

[88] Ayushi Rastogi and Ashish Sureka. What Community Contribution Pattern Says about Stability of Software Project? *21st Asia-Pacific Software Engineering Conference*, pages 31–34, 2014.

[89] E.S. Raymond. *The cathedral and the bazaar : musings on linux and open source by an accidental revolutionary.* " O'Reilly Media, Inc.", Sebastapol, CA, O' Reilly Media, 1990.

[90] James C. Raymondo. *Survival Rates: Census and Life Table Methods*, pages 43–60. Population Estimation and Projection, Quorum Books, New York, 1992.

[91] Peter C. Rigby, Daniel M. German, Laura Cowen, and Margaret-Anne Storey. Peer Review on Open-Source Software Projects. *ACM Transactions on Software Engineering and Methodology*, 23(4):35:1–35:33, 2014.

[92] Peter C Rigby, Yue Cai Zhu, Samuel M Donadelli, and Audris Mockus. Quantifying and Mitigating Turnover-Induced Knowledge Loss: Case Studies of Chrome and a project at Avaya. *IEEE/ACM 38th International Conference on Software Engineering*, pages 1006–1016, 2016.

[93] Julia Rubin and Martin Rinard. The Challenges of Staying Together While Moving Fast: An Exploratory Study. *IEEE/ACM 38th International Conference on Software Engineering*, pages 982–993, 2016.

[94] Norsaremah Salleh, Emilia Mendes, John Grundy, and Giles St. J Burch. An empirical study of the effects of conscientiousness in pair programming using the five-factor personality model. *ACM/IEEE 32nd International Conference on Software Engineering*, 1:577–586, 2010.

[95] Ingo Scholtes, Pavlin Mavrodiev, and Frank Schweitzer. From Aristotle to Ringelmann: a large-scale analysis of team productivity and coordination in Open Source Software projects. *Empirical Software Engineering*, 21(2):642–683, 2016.

[96] Klaus-Benedikt Schultis, Christoph Elsner, and Daniel Lohmann. Architecture Challenges for Internal Software Ecosystems: A Large-Scale Industry Case Study. *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 279–288, 2014.

[97] See T.W. Schultz. Investment in human capital. *American Economic Review*, (51):1–16, 1961.

[98] Panagiotis Sfetsos, Ioannis Stamelos, Lefteris Angelis, and Ignatios Deligiannis. An experimental investigation of personality types impact on pair effectiveness in pair programming. *Empirical Software Engineering*, 14(2):187–226, 2009.

[99] Jefferson O. Silva, Igor Wiese, Daniel German, Igor Steinmacher, and Marco A Gerosa. How Long and How Much : What to Expect from Summer of Code Participants ? *Proceedings of 33rd International Conference on Software Maintenance and Evolution*, pages 69–79, 2017.

[100] Arjumand Bano Soomro, Norsaremah Salleh, Emilia Mendes, John Grundy, Giles Burch, and Azlin Nordin. The effect of software engineers' personality traits on team climate and performance: A Systematic Literature Review. *Information and Software Technology*, 73:52–65, 2016.

[101] Igor Steinmacher, Igor Wiese, and Ana Paula Chaves. Why Do Newcomers Abandon Open Source Software Projects? *2013 6th International Workshop on Cooperative and Human Aspects of Software Engineering, CHASE' 2013*, pages 25–32, 2013.

[102] T Stolee, Kathryn, Sebastian Elbaum, and Anita Sarma. Discovering how end-user programmers and their communities use public repositories: A study on Yahoo! Pipes. *Information and Software Technology*, 55:1289–1303, 2013.

[103] Margaret Anne Storey, Alexey Zagalsky, Fernando Figueira Filho, Leif Singer, and Daniel M. German. How Social and Communication Channels Shape and Challenge a Participatory Culture in Software Development. *IEEE Transactions on Software Engineering*, 43(2):185–204, 2017.

[104] Ernest T Stringer. *Action research*. Thousand Oaks, California : SAGE, 2014.

[105] Pannavat Terdchanakul, Hideaki Hata, Passakorn Phannachitta, and Kenichi Matsumoto. Bug or not? bug report classification using n-gram IDF. In *Proc. of 33rd IEEE International Conference on Software Maintenance and Evolution*, pages 534–538, 2017.

[106] Patanamon Thongtanunam, Shane McIntosh, Ahmed E. Hassan, and Hajimu Iida. Review participation in modern code review: An empirical study of the android, Qt, and OpenStack projects. *Empirical Software Engineering*, 22(2):768–817, 2017.

[107] Jason Tsay, Laura Dabbish, and James Herbsleb. Influence of Social and Technical Factors for Evaluating Contribution in GitHub. *Proceedings of the 36th International Conference on Software Engineering*, pages 356–366, 2014.

[108] Jason Tsay, Laura Dabbish, and James Herbsleb. Let's Talk About It: Evaluating Contributions through Discussion in GitHub. *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 144–154, 2014.

[109] Bogdan Vasilescu. Human aspects, gamification, and social media in collaborative software engineering. *Companion Proceedings of the 36th International Conference on Software Engineering*, pages 646–649, 2014.

[110] Bogdan Vasilescu, Alexander Serebrenik, Mathieu Goeminne, and Tom Mens. On the variation and specialisation of workload—A case study of the GNOME ecosystem community. *Empirical Software Engineering*, 19:955–1008, 2014.

[111] J M Verner, O P Brereton, B A Kitchenham, M Turner, and M Niazi. Risks and risk mitigation in global software development: A tertiary study. *Information and Software Technology*, pages 54–78, 2014.

[112] Aurora Vizcaíno, Félix García, José Carlos Villar, Mario Piattini, and Javier Portillo. Applying Q-methodology to analyse the success factors in GSD. *Information and Software Technology*, 55:1200–1211, 2013.

[113] Yi Wang and David Redmiles. Cheap talk, cooperation, and trust in global software engineering. *Empirical Software Engineering*, 21(6):2233–2267, 2016.

[114] Krzysztof Wnuk, Per Runeson, Matilda Lantz, and Oskar Weijden. Bridges and barriers to hardware-dependent software ecosystem participation - A case study. *Information and Software Technology*, 56(11):1493–1507, 2014.

[115] Claes Wohlin, Per Runeson, Martin Höst, Magnus C. Ohlsson, Bjöorn Regnell, and Anders Wesslén. *Experimentation in Software Engineering: An Introduction*. Kluwer Academic Publishers, Norwell, MA, USA, 2000.

[116] Claes Wohlin, Darja Šmite, and Nils Brede Moe. A general theory of software engineering: Balancing human, social and organizational capitals. *Journal of Systems and Software*, 109(Supplement C):229 – 242, 2015.

[117] Xin Xia, David Lo, Lingfeng Bao, Abhishek Sharma, and Shanping Li. Personality and Project Success : Insights from a Large-Scale Study with Professionals. *IEEE International Conference on Software Maintenance and Evolution*, pages 318–328, 2017.

[118] Qi Xuan and Vladimir Filkov. Building It Together: Synchronous Development in OSS. *Proceedings of the 36th International Conference on Software Engineering*, pages 222–233, 2014.

[119] Kazuhiro Yamashita, Shane McIntosh, Yasutaka Kamei, and Naoyasu Ubayashi. Magnet or sticky? an OSS project-by-project typology. *Proceedings of the 11th Working Conference on Mining Software Repositories*, pages 344–347, 2014.

[120] Mark A. Youndt, ohan Subramaniam, and Scott A. Snell. Intellectual capital profiles: An examination of investments and returns. *Journal of management studies : JMS*, 41(2):335, 2004.

[121] Yasuhiro Yamamoto Yunwen Ye, Kumiyo Nakakoji and Kouichi Kishida. The co-evolution of systems and communities in free and open source software development. *Free/Open Source Software Development*, pages 59–82, 2004.

[122] Alexey Zagalsky, Daniel M. German, Margaret Anne Storey, Carlos Gómez Teshima, and Germán Poo-Caamaño. How the R community creates and curates knowledge: an extended study of stack overflow and mailing lists. *Empirical Software Engineering*, pages 1–34, 2017.

[123] Mansooreh Zahedi and Muhammad Ali Babar. Why does site visit matter in global software development: A knowle dge-base d perspective. *Information and Software Technology*, 80:36–56, 2016.

[124] Minghui Zhou and Audris Mockus. Does the initial environment impact the future of developers. *33rd International Conference on Software Engineering*, pages 271–280, 2011.

[125] Minghui Zhou and Audris Mockus. What Make Long Term Contributors: Willingness and Opportunity in OSS Community. *34rd International Conference on Software Engineering*, pages 518–528, 2012.