Situated Learning among Open Source Software Developers

The Case of Google Chrome Project

Josef Hardi

School of Computing
Blekinge Institute of Technology
SE-371 79 Karlskrona
Sweden
This thesis is submitted to the School of Computing at Blekinge Institute of Technology in partial fulfillment of the requirements for the degree of European Master in Software Engineering (EMSE). The thesis is equivalent to 40 weeks of full time studies.

Contact Information:
Author(s):
Josef Hardi 840122-P372
E-mail: josef.hardi@gmail.com

University advisors:
Prof. Barbara Russo
Free University of Bozen-Bolzano, Italy
Dr. Richard Torkar
Blekinge Institute of Technology, Sweden

School of Computing
Blekinge Institute of Technology
SE-371 79 Karlskrona
Sweden

Internet : www.bth.se/com
Phone : +46 455 38 50 00
Fax : +46 455 38 50 57
The presence of learning in organizations is important for success and survival. Recent research into open source software developers has primarily suggested a social constructivist view where knowledge is constructed in the social relationships within the team culture. I report results from a case study that investigated the presence of situated learning among open source developers at an earlier time of a project. Thirty-eight developers were systematically selected and examined on their performance, experience and roles during ten months of maintenance work. I followed a model of learning curve effects that associated the improvement in the average resolving time with the accumulated experience. I found a strong relationship between the two variables and confirmed the presence of learning. In addition, I found a less convincing evidence to affirm knowledge depreciated among open source software developers. The depreciation factor was estimated to be 94 percent, compared to other studies which ranged between 65 to 85 percent. An additional investigation was conducted around the organization structure to understand whether core and peripheral members have different average resolving time. The finding was inconclusive to claim both groups have different means towards issue resolution. The consistency in the result of this thesis and several other related research efforts suggests that learning is likely to be an intrinsic characteristic of open source software development rather than just a speculation in the literatures.

**Keywords:** Situated learning, learning curve, open source, longitudinal study.
Acknowledgments

I am indebted to the many people who have helped make this thesis done. My biggest debt is to Professor Barbara Russo in the Center for Applied Software Engineering (CASE) at University of Bozen-Bolzano who over one year has supervised me with great patience. She repeatedly gave many thoughts and inputs to this work and never gave up to convince me to stay focus on the objectives. Also to Dr. Richard Torkar from Blekinge Institute of Technology who is my co-supervisor in Sweden. Clearly, without their guidance and persistent helps, this thesis would not have been possible.

My sincere appreciation goes also to all my friends during my study period. Especially to Nattakarn “Joy” Phaphoom who has been a great friend for the past three years. To Dmitrii Miliaev and Sergei Pugach for their technical insights for coding the scripts. To Joy Okonkwo for the writing and grammar reviews. And also to everyone else who might not contribute to this thesis directly but their supports and trust are tremendously valuable.

Last but not least, my gratitude goes to my parent who always love and believe in me. To my brothers and sister to whom I always feel comfortable. In the end, I am so grateful to have you all.

Bozen-Bolzano, 24 September 2010
Josef Hardi
Abstract

Acknowledgments

1 Introduction

1.1 Situated learning in open source software developers communities

1.2 Goal of the thesis

1.3 Research questions and hypotheses

1.4 Contributions

1.5 Structure of thesis

2 Literature Review

2.1 The continuum of learning by context

2.2 Community of developers as a community of practice

2.3 The presence of learning in the community of developers

3 Research Method

3.1 Source of data

3.2 Methods of data collection

3.2.1 Issue report data

3.2.2 Review interaction data

3.3 Analysis of data material

3.3.1 Analysis of issue report data

3.3.2 Analysis of review interaction data

3.4 Construct the input data sets

3.5 Data fitting to learning curve models

3.5.1 Key variables

3.5.2 Empirical models

4 Results

4.1 The presence of organizational learning
Chapter 1

Introduction

1.1 Situated learning in open source software developers communities

According to the social constructivist theory, people construct knowledge as they interact with the world. Knowledge is assumed to be dynamic, relational and based on human interactions. Hence, knowledge cannot be separate from the activity and context where it originates and from the people who participate. Authentic learning and social interactions help the participants to negotiate meanings in the workplace that result in shared knowledge and understanding [6]. More importantly, these activities help to capture the tacit knowledge that is hard to articulate but, nevertheless, very demanding in the organizations.

A concrete model of situated learning can be discovered in communities of practice. Communities of practice integrate knowledge that is often tacit in nature but visible and observable in the common practice of competent practitioners. Learning in these communities does not happen like in the formal classroom way (i.e., with instructions) but, instead, learning implies becoming a practitioner [22]. The process of integrating oneself to the communities of practitioner entails the process of enculturation, in which people experience the culture of the organization directly through involvements and reflections. The role of the experts is to provide access to the experience and knowledge which help the learners reaching their potentials [37, 13].

If we focus on online communities, particularly the community of open source software developers, the appearance of learning is somewhat dubious. The most reason is because the practices in this community are fully mediated by technology. All developers are not in direct face-to-face contact and distant with each other. Developers face many challenges in the lack of or misunderstanding in communication, coordination difficulties and knowledge management problems. These challenges give the developers hard time to understand the tasks and communicate with others, which subsequently
Chapter 1. Introduction

prevent the knowledge creation process. Nevertheless, many open source proponents believe that the open source innovation offers significant learning opportunities from its best-practices [35, 21, 31].

However, little is known about learning in online communities. The social constructivist theory has less to say concerning online knowledge building. This thesis contributes to the existing literature by providing quantitative evaluations of learning phenomenon. In particular, my study was designed to assess the learning and knowledge acquisition in open source developers. I observed the maintenance activity of 38 long-term developers over a ten-month period and analyzed their improvement over time. I examined the effect of the learning curve on the parameter service time per individual and his accumulated experience. As new participants get acquainted with the project issues and tasks, their service time can be expectedly enhanced and, hence, learning is evidently seen.

1.2 Goal of the thesis

The main goal of this thesis is to gain a better understanding of learning in open source software developers communities. Compare to other existing studies, this thesis is significant because the observation went further into individuals rather than overviewing the project teams. The fine-grained of detail in the data set and analysis provides an in-depth view of what learning in a networked group of community may look like.

I decomposed my main goal into three sub-goals. The first is to investigate the presence of learning from problem solving and mentoring. The open source community offers a great deal of real issues related to the project development. Each individual developer is free to take any issue that is interesting to solve. Asking for an assistance is welcome in the community. Typically, there are several people who serve as the experts and they are willing to answer questions from other members who want to get involved. Mentoring is a common practice in open source community to get the new people be familiar with the project.

The second sub-goal is to investigate how knowledge is constructed. According to Bruner [7], we construct new knowledge based upon our current knowledge by taking into account the past experiences. However, how much past experiences should be taken into account is still a debate. Argote et al. [1] questioned the common assumption that suggested learning is cumulative, that it persists over time and does not evidence depreciation. The group did not convince by this assumption by arguing that “forgetting” cases may occur in organizations and, therefore, learning should have a rate of effectiveness.
This rate is measured as a percentage of the past experiences that are still arise in the current knowledge.

Lastly, the third sub-goal is to relate learning process and team roles. In organizational context, roles or positions are used to group people based on their specialized works and functions. They are usually assigned formally by the company management to each employee. However, in the open source community culture, roles are not strictly assigned but instead they arise naturally from a bottom up appointment as a result of the developer’s commitment [3]. Accordingly, roles in open source community may reflect the true competence of a developer. I limited the discussion about team roles into two categories: core and periphery, and observed whether the two categories have a different mean in learning process.

1.3 Research questions and hypotheses

Going through to the existing studies, our understanding of learning in open source community is no means complete. An early empirical study using data from Apache and Mozilla projects had shown an indication that learning curve effects occurred in software debugging activities [17]. In a recent article, Au et al. [2] extended the study by adding more sample projects from Sourceforge. Their conclusion provides a better generalization about learning in open source communities of developers.

However, both studies share the same data granularity level for the analysis. Both data are based on a collective aggregation strategy that takes the average or cumulative value of weekly data sets. These data are much in favor of “group effects” rather than “individual gains”. As a result, the conclusion is more favorable to the team’s productivity than to individual gains in learning. Thus, while these studies are fruitful in the context of organization level, little is known about the precise nature of learning in individual level. My study was initiated to have more insights in individual level by answering these two research questions as follows:

1. Is learning present in open source software developers at the earlier stage of the project?
2. What are the individual factors that affect the learning process?

To answer these questions, I conducted a case study in the Google Chrome project. The study investigated 38 software developers who participated in the maintenance activity of the project. I gathered longitudinal data for a ten-month period from December 2008 until October 2009. Each subject was
evaluated based on empirical data coming from the project artifacts. Subsequently, I designed the study to test the following three hypotheses.

**Hypothesis 1.** *As the number of issues resolved to date increases, the average issue resolution time decreases.* The first hypothesis explores the existence of learning curve effects. In general, the learning curve function relates improvements in performance to the accumulated production experience. In the context of maintenance works, performance can be measured by observing changes of individual service time in resolving project-related issues (e.g., fixing bugs or adding new features). The production experience can be described as the number of resolved issues. Subsequently, the interpretation may sound like: as an individual takes more and more participations and develops the skills, we can expect some observable improvements (in the time or outputs) in the subsequent trials. A log-linear model function was used to study the pattern between performance and the accumulated experience to measure quantitatively the evidence of learning effects. I collected the data from the earliest phase of the project to increase the observable effects.

**Hypothesis 2.** *Learning depreciates over time among the open source software developers.* The second hypothesis is intended to examine learning persistency among the open source developers. An assumption is often made that learning is cumulative, suggesting the stock of knowledge is simply equal to lagged cumulative experience without having a depreciation. However, knowledge may not constantly increase as people can fail to repeat the tasks due to their complexity or due to interruptions, thus affecting the effectiveness of learning activity. A significant knowledge depreciation was reported by Argote et al. [1] on a ship production process, showing an evidence that “forgetting” may occur in organizations. They estimated roughly that knowledge depreciates from 65% to 85% each month. I use this figures to formulate my hypothesis test that learning is depreciated considerably if the factor is equal or less than 85%. I believe this investigation of learning persistency may enrich our understanding about learning processes, especially in online communities such as communities of developers.

**Hypothesis 3.** *Participants that are rewarded as core developers resolve issues faster.* This hypothesis explores the project team structure and examines how different roles may affect learning process. I analyzed the message exchanges

---

1The assumption may not hold in general where the acquired skill and knowledge can be specific to a certain part of the system or to a specific area of code he has fixed. Thus, we can also expect some degradation, for example, when he tries to fix a different part of the system.
between developers to construct the social connectivity. By evaluating the network pattern, I estimated the role of developers and studied the behavior. Participants in different roles may exhibit learning process differently because the communication structure is different. Core developers are mostly acted as the central “hub” and all the contributors are connected to and through this hub [9]. Thus very likely, this situation introduces different access of information to the participants and, subsequently, leads to different learning experience.

1.4 Contributions

The main contributions and uniqueness of the work presented in this thesis are mentioned as follows:

1. **Thorough investigation of knowledge-building in a community of developers.** Existing studies regarding learning in open source project teams do not focus on individuals in the data set (e.g., [17, 2]). The investigation was focused more on team-level data that aggregate the measurements of each individual. Consequently, the resulting data are too general and have less distinctions. The end result might also be bias. For example, the increasing number of fixed bugs may be caused by community growth instead of learning effects per se. Similarly, different data sets may not be comparable if the sample units varied all the time. It should be noted that open source communities are typically dynamic. Developers are free to join in or leave out the community as they want and, therefore, assuming no such effect in the team participation may lead to an unwanted bias. My investigation is thorough because the observation involved examinations of individuals over time. Using this method, each developer is treated as a different individual and, therefore, the learning pattern that emerged can be truly identified.

Additionally, my contribution completes the spectrum of similar studies. Huntley [17] underpinned the study by starting the observation using the Apache and Mozilla project. Later, Au et al. [2] extended the observation by adding more sample projects and generalized the result using collective data. This thesis, in contrast, narrowed the observation to individuals using longitudinal data, allowing a more in-depth examination of the case study.
2. Techniques for link extraction from the code review histories to build social connectivity. Existing studies on analyzing the social network of open source team have focused on bug report (e.g., [10, 16]) and mailing list (e.g., [33]). Reporting bugs is claimed to be the center of open source project activities that brings a mixed of participants to work together [8]. Similarly, mailing lists are served as gateways to the community in which participants are invited to join in the discussion and start contributing. Such tools enable close interactions that frame a networked group of community.

Code review may provide a promising, alternative way to analyze the social structure of open source team. Data in code review share similarities with bug reports and email conversations. Lines can be drawn between participants based on the message exchanges and we can interpret their meanings. To my knowledge, this thesis is the first that uses code review to discuss team structure in a community of developers. I developed a method to extract and prepare the data for the core and periphery structure analysis. The model computation follows the analysis approach by Borgatti and Everett [4]. I found the method suitable for my purpose for estimating developer’s role in different time period. In a subsequent evaluation, the results were compared using other project artifacts and I found that they were likely compatible to each other.

1.5 Structure of thesis

This thesis consists of six chapters. The following paragraphs describe the content of this thesis and give an overview of the subjects studied in this work.

• **Chapter 1** provides the introduction and background of the thesis in terms of the goals, the research questions and the contributions.

• **Chapter 2** presents the overview of prior literatures that are related to this thesis.

• **Chapter 3** details the research method for measuring the variables and obtaining the data.

• **Chapter 4** presents and discusses the main finding of this thesis.

• **Chapter 5** presents the limitations of this thesis.

• **Chapter 6** provides the final conclusion of the thesis and directions for future work.
2.1 The continuum of learning by context

Reviews on the literature on knowledge construction and learning have shown a diversity of learning concepts in which contextualization is being placed as the central to the argument. The notion of “contextualization” emphasizes on how learning should be facilitated in a context, that is, the circumstances that establish the settings for learning and knowing can be fully understood and meaningful.

The significance of learning in context has been a long discussion in the literature. There is a classical debate about how learning should be transpired at school. Most education systems adopt instructional systems design that often regards learning as a mechanism for transferring information from head to head (i.e., from teachers to students). In this paradigm, learning is decontextualized and isolated in the school environments, and students are asked to receive the knowledge for granted and replicated the content and structure in their ways of thinking [18]. Psychologists called this learning model as *instructivism* which is basically based on direct instruction mode and passive learning. In contrast, another paradigm called *constructivism* argues that learning and knowing require active interpretations from the learners and they both are subjective matters. The main reason of this argument is because people all affected by the world somewhat differently. Each person has his own unique set of experiences and placed on his own belief about them. According to constructivists, knowledge is constructed gradually and stacked upon personal prior knowledge by time-to-time [29]. Clearly, in this scenario, teachers cannot assume that all their students will understand the new information in the same way but rather they will require a variety of different experiences to construct their own new knowledge.

Organizing these two views on a continuum of contextualization gives a polarized spectrum of learning. Figure 2.1 illustrates the opposing arguments where context is found to be more crucial in one model than in another. At
one end of the edge, the instructivists believe that learning should be delivered in systematical ways in which learners are being directed to the objective truth by the masters. The term “objective” refers to an agreement of the existence of a single reality where the external world is explained as it is. Other interpretations outside this common belief are stigmatized to be wrong and learner’s knowledge need to be re-evaluated. The constructivists, at the other end, reject the simple and easy approach offered by the instructivists. According to them, there are many ways to structure the world and there are multiple realities, or meanings, for any event or concept. Therefore, nobody is obligated to tell about the realities, and that realities are personal constructions, developed individually and independently of each other.

Although the two theories are opposing, the arising gap between them opens a new ground to develop. Jonassen [18] asserted that there is a need to mediate the importance between constructive learning and instructive teaching. He argued “[...] since learning obviously entails constructivist and objectivist activities, the most realistic model of learning lies somewhere on the continuum between these positions.” Constructivism sounds reasonable when focusing and discovering the individual competence by developing their own personal value scale but it is hard for people to decide from where to start. According to Rieber [30], this problem can be solved by the help of instructional design systems. He argued instructions give an initial point of

---

**Figure 2.1:** The continuum of learning in context, as mentioned in [25], [18]
entry for learning that, subsequently, can trigger productive interactions to take place. In succession, this creates the link between constructivism and instructivism.

There is also an obvious argument around the conceptions of the nature of learning that says humans cannot develop by themselves. It is doubtful that novices can rediscover the complex human knowledge without having a guidance from someone who already has the understanding of that knowledge. According to Vygotsky, much important learning by a child occurs through social interaction with a skillful tutor [37]. He introduced the term of “Zone of Proximal Development”, or ZPD, to express a condition in which a learner requires a collaboration with more capable peers to help him obtain a skill. Once the skill is obtained, the learner can move away for the peers and be independent on his own. In general, ZPD explains the role of interactions between learners and instructors.

In the context of adult learning, one of the most significant and intuitive papers was written by Brown and colleagues [6]. The authors introduced the notion of situated cognition to argue that learning occurs more effectively in context, and that context is an important part in conveying the knowledge base related to that learning process. The argument adds the assumption that just providing assistance or instructions to train people are not enough, instructors should also need to know how to create an environment that makes learning more meaningful and relevant. Jonassen and Strobel [19] cited that meaningful learning requires a meaningful task that emerges from some authentic activities. The definition of authentic activities simply means as the ordinary practices of everyday members and that are framed by the culture [6]. Several case studies in apprenticeships have demonstrated this type of learning with various different formalities and cultural influences (e.g., see the many studies cited in Lave and Wenger’s review [23]).

As in the literature, the term “community of practice” is often used over “apprenticeships” to avoid the connotation of outdated and obsolete education [23]. Although, it should be noted that this association is not truly correct. Apprenticeships can still be found clearly in the modern era, especially wherever high level of knowledge and skill are in demand (e.g., engineering, medicine, academy, professional sports and arts). I have chosen to use the term of “community of practice” to emphasize the wider description of learning in social context instead of simply a master–apprentice relationship.
Chapter 2. Literature Review

2.2 Community of developers as a community of practice

According to cognitive anthropologists Jean Lave and Etienne Wenger, the communities of practice are defined as “groups of people who share a concern or a passion for something they do and learn how to do it better as they interact regularly” [38]. They described three characteristics that embody communities of practice as domain, community and practice. Table 2.1 describes the agreements between communities of practice and communities of developers based on those characteristics.

Table 2.1: The agreement between communities of practice and open source communities

<table>
<thead>
<tr>
<th></th>
<th>Communities of practice</th>
<th>Communities of developers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Domain</td>
<td>A community of practice has an identity defined by a shared domain of interest. Having a membership in the community means a commitment to the domain.</td>
<td>The identity of an communities of developers is predefined well in the source code. The meaning of the code and the significance of the project can only be understood by those who are committed in the development.</td>
</tr>
<tr>
<td>Community</td>
<td>A group of people connected by a specific domain interacts and engages in joint activities and discussions. The relationships are based on mutual respect and the chance to learn from each other.</td>
<td>A group of developers engaged in a production of a software. They interact intensively using mailing lists as the central mean of communication to reach all the members in the project. Using this medium, members share ideas, answers and sometimes disputes that enrich the understanding.</td>
</tr>
<tr>
<td>Practice</td>
<td>Members of community of practice are practitioners. They actively engaged in a discipline or profession as part of their daily life. Each member owns a collection of practices that can be shared with the others through stories and/or actions.</td>
<td>Many of the members of communities of developers are the finest programmers. They come from various backgrounds from hobbyists and students to enterprise employees. Inside the community, they help each other by sharing code, tips and advices based on personal experience.</td>
</tr>
</tbody>
</table>

Aligned with the concept of communities of practice at physical workplaces, Henri and Pudelko [14] explored communities of practice in virtual settings. According to them, the social context and the emergence of the vir-
virtual communities of practice are stemmed from the existing, real communities of practice. The only significant difference is the usage of communication medium (e.g., the Internet) in which makes possible a broader membership. The authors also make a clear distinction between other types of virtual online communities, such as newsgroups (p. 478) and e-learnings (p. 481) to avoid confusion.

Lave and Wenger [23] described learning in communities of practice as a sociocultural transformation. The role of active participation helps to develop skills and capabilities that are commonly described as tacit knowledge. The process involves novices who wanted to become proficient at their practice got engaged in the learning situation and progressively became part of a community of practice. By fully participating in the “ritual” of a community, the novices can be fully certain of their knowledge acquisition and they are able to reflect the best practices. This process of assimilating group practices into their own is described as legitimate peripheral participation (LPP). There are some evidences that this LPP process is often used in communities of developers.

In an article, Krogh et al. [36] observed the process by which new people join the existing open source community and how they initially contribute code. They found that often new members spent significant effort scrutinizing the mailing list to understand the community and the project in which they started to engage. This “lurking” activity helped them to increase confident and their sense of belonging to the new group. Others expressed their interest by directly responding to the group’s needs, such as, reporting bugs or offering bug-fixes. All these efforts are done to create legitimate characters inside the team as described by the LPP process.

The journey to reach full participation in communities of developers is also clearly understood. Krogh et al. [36] translated the LPP process by referring it to a joining script. The authors asserted that there is a certain cost, e.g., in the amount and type of activity, in order to generate the transition from the peripheral to central participation. For example, the transition from “joiner” to “newcomer” occurs when the joiner is granted CVS access as the reward from his mentor in demonstrating his programming skill.

### 2.3 The presence of learning in the community of developers

Up to this point, attention has been focused upon the theoretical background of learning and that communities of developers are likely compatible with the
Chapter 2. Literature Review

Theories. There are also some efforts concentrated in proving the assertions. At least there are two studies concerning learning observation in open source communities.

The initial study was conducted by Huntley [17] who included data from two prominent open source projects, i.e., Apache and Mozilla. The data were obtained from the debugging activities in weekly basis. Learning phenomenon was predicted using time-series analysis to detect the association between debugging performance and debugging outcomes. He observed that while the two projects employed similar approaches and tools, both showed a significant difference in respect to the association, i.e., the learning processes. He added that learning outcomes are depended to the situation and they should be taken care of in different organizations.

Other similar study was recently conducted by Au et al. [2] who included a much larger number of open source projects. Their objective was to find whether learning processes commonly present in open source projects. In general, the authors used a similar model as Huntley [17] although several new variables were also introduced, such as project size, project category, and developer experience. Their conclusions showed an agreement that communities of developers can provide a place to develop skills in software development.

![Figure 2.2: The investigation spectrum](image)

The overall studies, including this thesis, create a spectrum of investigation regarding learning phenomenon in communities of developers. Figure 2.2 illustrates my intention in adding a new perspective in this topic. Neither Au [2] nor Huntley [17] used individuals as their observational unit but instead they focused on the collective team achievement. I found this approach is less thorough and raises uncertainties in the interpretation. For example, the increasing number of fixed bugs may be caused by community growth
instead of learning effects per se. I also notice that the sample units (i.e., the developers) can vary in different time intervals which may lead to an inconsistency in the measurement.

My other concern is supported by Fiol and Lyles [11] that state organizational learning is not simply the sum of each member’s learning. They added: “It [organizational understanding] results in associations, cognitive systems, and memories that are developed and shared by members of the organization” [Ibid, p. 804]. Open source communities are characterized by free sharing and creative practices among the participants that foster organizational learning. Each individual is unique in the way they understand the materials and construct the knowledge and skills. One possible way to observe this phenomenon is by measuring the learning experience per individual over time and then interpret the trend as reflecting learning event. In this way, we can avoid a premature assumption as the previous studies have done with collective data.
Chapter 3

Research Method

In this chapter, I outline the research method employed in this study. It adopted a longitudinal study design to investigate changes over time. The descriptor of the data comes from the surveys that were performed repeatedly in different time periods. Singer and Willett [32] pointed three important features of a longitudinal study, as follows:

1. **Multiple waves of data.** The meaning of “waves” is related to the subsequent trials of measuring an event where each trial is separated by some amount of time. The number of waves may help displaying the individual growth trajectory, that is, the way his or her outcome values rise and fall over time. In general, more waves are better and allow to posit more elaborate statistical models, reduce restrictive assumptions and give better shape of the person’s individual growth trajectory.

2. **A sensible metric for time.** There are two requirements for this feature: the *metric unit* and the *time spacing*. The metric unit concerns with the time scale, such as, days, weeks, months or years. Other units can also be used to clock the time, such as, grades, miles, number of sessions or release versions. All the options are valid and the choice depends on what makes the outcomes and the research questions sensible. The spacing of the waves of data collection is flexible as well; as long as it can sufficiently collect the data to provide a reasonable view of each person’s individual growth trajectory. In consequence, it is also not necessary to have an equal spaced waves.

3. **A continuous outcome that changes systematically over time.** An important concern in obtaining longitudinal data is to ensure the outcomes of the measurements are always valid over the time. There are three things to be considered, namely, *metric, validity* and *precision*. The first is related to the equivalence of the outcome scores over time. Using an identical instrument for the measurement during the whole progress of the study may guarantee this equivalence. The second is to ensure
the validity of the instrument over time. Researchers may find the instrument is not valid anymore in measuring the same event because the subjects are more mature in the subsequent time. The last matter deals with how to minimize errors introduced by instrument administration. Having a “reliable enough” precision depends on the customary of the number and spacing of the waves of data collection and, therefore, a careful research design may help reducing the error effects.

In practice, longitudinal data analysis represents a marriage between regression and time-series analysis [12]. As with most of regression data sets, longitudinal data are composed of a sample of subjects. The data sets also have a time dimension to record the multiple observations that occurred in different series of time period. By merging these two approaches together, longitudinal data can provide rich information on growth and development that happen in social situations.

The use of longitudinal data analysis has been supported by researchers who argued that the approach is powerful to observe phenomenon that contains variables which are variant across time [12, 32]. Longitudinal data analysis allows the observation of a process that has gradual changes through a series of states, revealing the trajectory of development processes. For example, this analysis is suitable to study the development trajectory of reading ability of children age 6 to 16. This is worth considering in the design of this study as well because of its focus on learning as a process among open source software developers.

My research method proceeds in four phases: (1) data collection, (2) analysis on data material, (3) input data construction and (4) data fitting to learning curve models. I developed methods that focus on data collection using automated text mining approach. The approach involves the process of structuring the input text for data parsing, data cleaning and data storage into a local database. The data analysis includes applying measurements to all subjects in each release. I introduced three measurements with respect to performance, experience and team role. Next, I systematically selected the subjects based on their participation in the development process and chose thirty-eight long-term developers. Accordingly, I refined the existing data from the data analysis and constructed the input data for the data fitting. Lastly, I estimated several regression models based on learning curve model to fit my input data and to test my hypotheses.
Chapter 3. Research Method

3.1 Source of data

Before I begin with the details of each step in my research methodology, I am going to briefly explain the sources of data used by the study. These data sources come from the online collaboration tools that are typically used by the open source community to control day-by-day workflow.

**Issue tracking system:** An issue tracking system is a software tool that maintains the lists of project-related issues, coming from users and developers. The tool is basically a database system with a user interface that manages user inputs and data presentation. The end-users can create a new issue, add details, write comments, and change its status by confirming or resolving the issues. Each user can have several issues assigned to them. Users have also the option to reassign an issue to another user and the system will send notification to the corresponding party. The lifespan of an issue effectively ends once it is resolved, though there may be some subsequent activity as the team verifies that the bug is in fact resolved.

The Chrome issue database\(^1\) manages over 47,000 issue reports. Table 3.1 shows the data available from the issue reports. An issue is entered into the system at an open date with a unique id and an initial status (e.g., “unconfirmed”). At minimum, an issue report should contain the problem description, the steps to replicate the issue, the expected behavior, and the type and area in which the issue can be categorized. Anytime a user makes a change to the status, the system will record the time and the action so as to maintain the history of the issue’s lifespan. For example, if someone took an issue, the system will record the assigned date, the name of the owner and change the status into “assigned”. There are other labels as well but these are sufficient for my purposes.

The issue report data can be used to measure developer productivity levels over the course of the project. Possible measures of productivity within a given month include the number of issues resolved and the average time needed to resolve issues. From these productivity measures, one can also estimate the developer experience and his working performance.

**Code review system:** The code review system is a tool to facilitate peer review and code quality assurance via online. The system provides a logging facility, in which developers can track the reviewing activity, including comments and the patch history, over time. Reviewers can instantly insert comments under each line of code he wants to criticize. Likewise, the patch owner can reply

\(^1\)URL http://code.google.com/p/chromium/issues/list (last accessed on July 2010).
Chapter 3. Research Method

Table 3.1: Issue report data schema

<table>
<thead>
<tr>
<th>Label</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Issue Id</td>
<td>The issue number. e.g., 23412.</td>
</tr>
<tr>
<td>Type</td>
<td>The issue type. An issue can only have one type. e.g., Bug.</td>
</tr>
<tr>
<td>Area</td>
<td>The product area to which an issue belongs. An issue can belong to multiple areas. e.g., BrowserUI.</td>
</tr>
<tr>
<td>Status</td>
<td>The current status of the issue report. There are two categories of status label: e.g., Fixed.</td>
</tr>
<tr>
<td>Owner</td>
<td>A developer who responsible for the resolution of an issue. e.g., tony.</td>
</tr>
<tr>
<td>Open date</td>
<td>Date and time an end-user or a developer reported an issue. e.g., 2009-09-30T03:36:42.000Z (1254296202).</td>
</tr>
<tr>
<td>Assigned date</td>
<td>Date and time a developer took an issue. Subsequently, he or she becomes the owner of the issue. e.g., 2009-10-02T00:31:42.000Z (1254457902).</td>
</tr>
<tr>
<td>Started date</td>
<td>Date and time the owner started working on an issue. e.g., 2009-10-02T17:26:07.000Z (1254518767).</td>
</tr>
<tr>
<td>Close date</td>
<td>Date and time the owner closed an issue. e.g., 2009-10-02T21:06:50.000Z (1254532010).</td>
</tr>
</tbody>
</table>

\(^a\) in UNIX timestamp format.

back to the comments by providing the patch revision. The system also provides the “side-by-side” diff—a common debugging utility—to help scanning the differences between any two patch versions.

According to Chrome developers guide document, code review is a mandatory practice for contributing code. Every changes in the Chrome source code must be reviewed first before they get committed. The purpose is to ensure that the corresponding changes are appropriate and follow the coding conventions. In practice, code review is performed informally by peers who are familiar with the area of the code. During the reviewing process, comments and patch revisions are intensively exchanged from both parties (i.e., the patch

\(^2\)http://dev.chromium.org/developers (last accessed on July 2010).
Table 3.2: Review interaction data schema

<table>
<thead>
<tr>
<th>Label</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Review Id</td>
<td>The code review number.</td>
</tr>
<tr>
<td></td>
<td><em>e.g.</em>, 16620.</td>
</tr>
<tr>
<td>Owner</td>
<td>The patch owner.</td>
</tr>
<tr>
<td></td>
<td><em>e.g.</em>, pkasting.</td>
</tr>
<tr>
<td>Reply date</td>
<td>Date and time a patch owner posted a reply.</td>
</tr>
<tr>
<td></td>
<td><em>e.g.</em>, 2009-10-21T17:00:52.000Z (1256144452a).</td>
</tr>
<tr>
<td>Reviewer</td>
<td>The reviewers.</td>
</tr>
<tr>
<td></td>
<td><em>e.g.</em>, darin.</td>
</tr>
<tr>
<td>Comment date</td>
<td>Date and time a reviewer posted a comment.</td>
</tr>
<tr>
<td></td>
<td><em>e.g.</em>, 2009-10-22T05:06:59.000Z (1256188019a).</td>
</tr>
</tbody>
</table>

*a* in UNIX timestamp format.

owner and reviewers). When the reviewers are satisfied with the final revision, they will write like “OK”, “LGTM” or “Looks Good To Me” to indicate their approval. The patch owner then can then commit his work to the main repository and closes the review.

Code review plays an important role in the enculturation of newcomers into programming best practices. Coding examples are given publicly and newcomers can read and study the code structure made by the experts. When they start contributing, their patches will be evaluated with feedbacks based on what are the mistakes and how they can be improved. This retrospective approach can slowly develop and enhance the programming skill of each participant as they gradually move toward full participation in the developers community. The relationship between newcomers and experts is clearly visible in the reviewing activity.

Code review interaction data can be used to measure the activity level of developers over the course of the project. Table 3.2 shows the data available from code review page. The data helps to reveal the interconnections among developers during the reviewing activity. This measurement is important for drawing the social network of developers and observing their interactions.

Revision control system and blog: Revision control system perhaps is the most common tool used by the open source communities. It is designed to synchronize work and keep track of changes in the project artifacts made by developers working on the same set of files. The tool stores its revision control information in a central server, called the repository. Developers can add, remove and modify files from this repository as well as collect the versioning
Chapter 3. Research Method

Table 3.3: SVN log data schema

<table>
<thead>
<tr>
<th>Label</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Revision Id</td>
<td>The revision number. e.g., 28510.</td>
</tr>
<tr>
<td>Author</td>
<td>The developer who made the changes. e.g., estade.</td>
</tr>
<tr>
<td>Commit date</td>
<td>Date and time a commit was made. e.g., 2009-10-09T01:03:42.000Z (1255050222a).</td>
</tr>
<tr>
<td>Review URL</td>
<td>The Internet address of the review page, written in the log description. e.g., <a href="http://codereview.chromium.org/261033">http://codereview.chromium.org/261033</a>.</td>
</tr>
<tr>
<td>Issue Id</td>
<td>The issue id, written in the log description. e.g., 23339.</td>
</tr>
</tbody>
</table>

\(^a\) in UNIX timestamp format.

information of the files. When committing changes, it is encouraged to write a description and rationale so others can easily understand the current status of the file.

Table 3.4: Google Chrome release schedules

<table>
<thead>
<tr>
<th>Version</th>
<th>Date</th>
<th>Days</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>1.0.154.43</td>
<td>8 January 2009 (^a)</td>
</tr>
<tr>
<td>T2</td>
<td>1.0.154.48</td>
<td>2 February 2009</td>
</tr>
<tr>
<td>T3</td>
<td>2.0.169.1</td>
<td>11 March 2009</td>
</tr>
<tr>
<td>T4</td>
<td>2.0.172.8</td>
<td>19 April 2009</td>
</tr>
<tr>
<td>T5</td>
<td>2.0.172.30</td>
<td>20 May 2009</td>
</tr>
<tr>
<td>T6</td>
<td>2.0.172.33</td>
<td>18 June 2009</td>
</tr>
<tr>
<td>T7</td>
<td>3.0.193.2</td>
<td>15 July 2009</td>
</tr>
<tr>
<td>T8</td>
<td>3.0.195.6</td>
<td>4 August 2009</td>
</tr>
<tr>
<td>T9</td>
<td>3.0.195.20</td>
<td>11 September 2009</td>
</tr>
<tr>
<td>T10</td>
<td>3.0.195.27</td>
<td>9 October 2009</td>
</tr>
</tbody>
</table>

\(^a\) The iteration was started on 12 December 2008.

Chrome uses Subversion (SVN)\(^3\) as its revision control system. The repository manages 26 top-level projects including Chrome and maintains over 52,000 logs by the time this thesis is written (i.e., Chrome 5.0 has been released). Table 3.3 shows the data available from the SVN logs that are used in this study. As shown, the SVN logs bridge the issue reports and review

\(^3\) URL http://src.chromium.org/viewvc/ (last accessed on June 2010).
logs. This additional information allows me to make connections and support verification in the data collection.

Other source of data comes from the release blog\(^4\) that announces the official release for Google Chrome new versions. The blog provides information about the updates, security fixes and changes. The blog also logs the release date that is useful to divide the data based on the project iterations; see Table 3.4. On average, each release requires one month development time ($\bar{x} = 31.2$). Thus, the partition size is considered as equal to a monthly data collection. I will use the term “release data” and “monthly data” interchangeably across this thesis.

### 3.2 Methods of data collection

This section describes how to gather data systematically from remote, public accessible servers mechanically. In general, the procedures involve data extraction, data cleaning and data storage [15]. The data extraction and data cleaning were done automatically using program scripts and some filtering criteria. I used MySQL database system for the data storage.

There are two primary data sets as the outcomes. The first is the issue report data which encompass the records of issues lifespan and informal discussion. These data are used to measure work performance and code production experience. The second is the review interaction data. The set comprises the chronology of “comment-and-reply” events between developers during peer review. The data provide the information for analyzing social interactions.

#### 3.2.1 Issue report data

I used a similar methodology introduced by Sowe et al. to gather data from mailing lists [33]; see Figure 3.1. The procedure starts by choosing the information that are relevant for the analysis as the data schema. A program script implementing the schema was then used to extract the text data from the issue reports. Next, the collected data are stored in the database for further queries. In order to improve the quality of the data set and account for their relevancy to the study, certain reports had to be removed. The data cleaning process involved removing reports using several filtering criteria (explained later in the subsequent section).

\(^4\)URL http://googlechromereleases.blogspot.com/ (last accessed on July 2010).
Chapter 3. Research Method

Figure 3.1: Issue report data collection workflow
Chapter 3. Research Method

Data extraction

There are two strategies to extract data from the issue tracking database. The simplest and most direct approach is to use the “download” link located in the project website. The link will automatically trigger the production of a CSV file by showing the report list and then having it downloaded immediately. However, this approach does not facilitate a thorough data collection where users have less control for its output. Another option is to utilize the Google Issue Tracker Data API that allows the data collection to be programmed. This approach is better because it gives users more control to develop a custom client application according to their needs.

I chose the latter approach to perform the data collection. I developed a script using the API for Python; see Appendix A. The data extraction was performed on 24 April 2010 where I collected past reports from the period 1 December 2008 to 9 October 2009. In total, there were 23,616 issue reports coming from the development version 1.0 to 3.0. The output data were imported to a MySQL database for storage and further analysis.

Data cleaning

There was a fair number of issue reports that were not relevant to the study. The tracking system maintained issues not only for Chrome but also for other modules (e.g., WebKit, BuildTools). In addition, not every issues were resolved yet thus they were still in an open status. In details, the data cleaning process involved removing reports in the following categories:

- **Unrelated project areas**: I looked for reports that belong to the development area of Chrome. Sampling reading and examination showed ten modules belong to the Chrome development, e.g., BrowserUI, ChromeFrame, Compat, Compat-Web, Extensions, Internals, I18N, Feature, Misc, and UI-Features. Reports in the remaining modules were removed.

- **Close issue status**: I looked for reports that were close and valid. Specifically, I took two status labels: FIXED and VERIFIED, to be included in the analysis. Open or invalid reports, e.g., UNCONFIRMED, UNTRIAGED, AVAILABLE, DUPLICATE were subjected for removal.

- **Empty owner name**: The table was queried for empty owner names. The output query showed 712 reports had no owner. However, a cross-check using information from the SVN logs could reveal the owners of the reports. I based on the assumption that the person who committed the code (i.e., owned the SVN log) should also own the issue report. And
since both artifacts are linked together, the conclusion is easily obtained. In total, 147 anonymous issue reports were recovered. The remaining unrecoverable reports were then removed.

By applying these filters, I refined my data and collected 5,160 issue reports. The majority of the issue type is bug report (92%), followed by feature requests (5%), cleanups (2%), and other types (1%).

3.2.2 Review interaction data

The code review system is a unique tool introduced by the Google team and has not been widely practiced in open source communities. Hence, there is little experience of exploiting its data in practice. This thesis aims to introduce the potential and benefits of using code review histories and to encourage future researches using this new and important source of information when it is possible.

Figure 3.2 shows the workflow for collecting data from the code review repository. The procedure starts by doing review selection to filter which reviews are valid for the study. This step is an optional step and can be applied just in case non-related data were found in the review collection. After I selected the review pages, I extracted the information using a program script and stored the output into the database. Later, this data set will be the basis for constructing the snapshot data used in the social network analysis.

Review selection

An early filtering is required to ensure the extraction is applied to the correct historical reviews. Similar to the issue tracking system, the code review system does not exclusively maintain reviews for Chrome development only but also other third-party modules. Therefore, the extraction cannot be done immediately because the result may include some irrelevant data. The solution to this problem was addressed in two folds. First, I filtered the SVN logs to restrict the data coming from the Chrome module only. Second, I extracted each log description to get the corresponding code review URL that was appended automatically by the review system.

I developed a program script to parse the SVN log description and extract the review link. The script looked for a match to a keyword “Review URL” and outputted the subsequent string which is the review URL. I gathered a collection of Chrome SVN logs in the period from 1 December 2008 to 9 October 2009 and obtained 9,019 review URLs.
Data extraction

Another Python script was used to run the data extraction. For each input URL, the script accessed the review page and went to the message area. The script then looked for any message exchanges between developers. There are two kinds of actor in code review: a patch owner and a reviewer. To distinguish an owner from the reviewers, the script searched the person who initiated the review. In fact, the patch owner typically starts the code review by posting the patch. Subsequently, the remaining participants are identified as the reviewers.

In order to depict the interaction from the exchanged messages, I had to decide on the destination of each interaction. Based on my sample reading, it appeared that comments were directed to the patch owner. No evidence was found to suggest a comment could be directed to a reviewer as well. This finding was reasonable considering the fact that commentaries are intended to evaluate the owner’s work rather than to create an open discussion.

The solid definition about the destination gives code reviews an advantage. Using comment records from the code review eliminates the problem of broadcasting, that is, the difficulty of determining the true destination due to unclear target audience. This problem typically happens in an open dis-
Crowston and Howison [9] mentioned this kind of problem in their article when they tried to determine the interactions in bug reports mechanically. Their solution was to code the interactions based on the posting chronology which assumed responses always address to the sender of the previous message. However, this approach is overly simple because people can make responses to anybody and this assumption may lead to bias in the observation. The output for each running script is stored then into a MySQL database. In total, I gathered 17,425 logs from 239 unique participants.

3.3 Analysis of data material

The purpose of this section is to analyze the raw data from the data extraction focusing on their substantial meaning. The analyzed data is presented by means of the two groups: issue report data and review interaction data. Recall from the previous section, issue report data come from the issue tracking system that detail about the developer’s productivity levels over the course of the project. Review interaction data come from the code review tool that detail about the developer’s interaction levels.

3.3.1 Analysis of issue report data

From the issue report data, I measured two variables: performance and experience. Both measures were applied to each developer in a given monthly data set. The performance was measured by calculating the average of issue resolution time and the experience was measured by counting the amount of contribution. When a developer closes an issue, the system logs the closing date for that issue. The time distance between the dates when he starts working on an issue until he closes it is equal to the issue resolution time (in days). However, this ideal measurement was sometimes hard to get because of missing information and, therefore, we used also different date information (e.g., assigned date or open date). Specifically, the measure used to calculate the estimation of the resolution time is the following:

\[
T_{\text{resolution}} = \begin{cases} 
T_{\text{close}} - T_{\text{started}}, & \text{if } T_{\text{started}} \text{ is not empty.} \\
T_{\text{close}} - T_{\text{assigned}}, & \text{if } T_{\text{assigned}} \text{ is not empty.} \\
T_{\text{close}} - T_{\text{open}} & \text{otherwise}
\end{cases}
\]

where, \(T_{\text{resolution}}\) is the issue resolution time, or service time. The variable \(T_{\text{close}}, T_{\text{started}}, T_{\text{assigned}}\) and \(T_{\text{open}}\) represent the closing date, starting date, assignment date and opening date, respectively.
Chapter 3. Research Method

The formula provides an accurate estimation at the first case. It gets less accuracy at the second case and the least at the third case. There were fairly equal sizes on the application: 29 percent of the data applies the first case, 32 percent applies the second case and 39 percent applies the last case. I did another experiment that computed the resolution time based on opening date and closing date only (all issue reports have both information). I found that the different treatment neither gave much improvement to the data fitting nor difference to the final results. However, my formula gave a more reasonable estimation compared to the other approach that used the opening date information only. Hence, I present my analysis based on the formula above. Appendix D.1 shows the detail results of the alternative experiment.

3.3.2 Analysis of review interaction data

This part of the analysis was divided into three steps. In the first step, I created the snapshot data for each observation. The snapshot data contain the weight of the interaction that is specified based on the amount of comments posted during the reviewing process. In the second part, I rescaled the weight in order to improve the measurement. And in the last part, I performed the network construction using a social network model proposed by Borgatti and Everett [4].

Creating snapshot data

Each comment message has a timestamp given when it is received by the review system. This information is useful to divide the network into periodical snapshots. Specifically, I sampled the network in a 2-release window which is equivalent to two months observation. Figure 3.3 illustrates how I moved the window one release at a time to have different sets of snapshot data. The figure also shows an overlap in two consecutive snapshots and is applied to all periods. I chose to use overlapping windows to smooth changes in the network structure and two releases were chosen so that actors have sufficient data to get analyzed.

The snapshot data contained the interaction weight between pairs of developers. The value was simply obtained by grouping records of the same pairs and summing the total comments. The weight reflects the closeness or the distance of team members in collaboration. The higher the value, the closer the two actors worked together, indicating a substantial position in the team.
Rescaling snapshot data

An examination on each snapshot data set showed that several pairs interacted significantly more than the others; see Figure 3.4. A preliminary data analysis showed that the significant differences contributed to a dramatic effect in the network model outcomes. An obvious example shows in the tenth snapshot where only a single core member is identified. Other snapshots show a range between 3 to 6 cores and one case of having 13 cores. It appears that having many outliers prevent the identification of other potential core candidates. One way to reduce this outlier effect is to control the significance of these pairs by rescaling their interaction weight\(^5\).

To my knowledge, there is no prior study that proposes a particular rescaling method on weighted networks. Therefore, I developed my own set of boundaries and ranges for the rescaling. I used quartile to divide the data in an equal size and categorized them according to the divisions.

Five different ranges were produced to categorize the weight values in the snapshot data. Each category was assigned a numerical value ranging from one for “poor tie” to five for “very strong tie”. Determination of the traits for each category is meant to specify the quality of the ties according to the interaction intensity, or its weight. I defined the rescaling function as follows:

\[
\text{rescale}(w) = \begin{cases} 
1 = \text{poor}, & \text{if } w \leq Q_2 \\
2 = \text{weak}, & \text{if } Q_2 < w \leq Q_3 \\
3 = \text{moderate}, & \text{if } Q_3 < w \leq Q_3 + 1.5 \times \text{IQR} \\
4 = \text{strong}, & \text{if } Q_3 + 1.5 \times \text{IQR} < w \leq Q_3 + 3 \times \text{IQR} \\
5 = \text{very strong}, & \text{if } w > Q_3 + 3 \times \text{IQR} 
\end{cases}
\]

where, \(w\), \(Q_2\), \(Q_3\) and \(\text{IQR}\) represent the interaction weight, the median, the upper quartile and the interquartile range, respectively.

\(^5\)I would like to credit Martin Everett (University of Greenwich, England) for the rescaling suggestion to reduce the outlier effect without having to eliminate pairs with high values.
Figure 3.4: Outliers in snapshot data. The x-axis represents the snapshot data, the y-axis represents the interaction weight (scaled by log) and outliers are depicted as circles. The box-plots show that ties with weight more than 8–11 are categorized as outliers. The data distribution appears to have a positive skewness, suggesting only a few pairs have a very close tie.
Core/periphery structure analysis

The core and periphery structure analysis was carried out using UCINET [5]. The tool implemented the core/periphery model algorithm to estimate the “coreness” of each participant and define the structure. The tool includes a measurement to evaluate the wellness of the structure estimation by providing a concentration score. The tool uses the score to compare different core sizes and recommends the optimal solution. Table 3.5 shows the results in executing the tool using the rescaled data sets. Appendix C explains the execution of using the tool.

Table 3.5: Core participants in Chrome project

<table>
<thead>
<tr>
<th></th>
<th>S1</th>
<th>S2</th>
<th>S3</th>
<th>S4</th>
<th>S5</th>
<th>S6</th>
<th>S7</th>
<th>S8</th>
<th>S9</th>
<th>S10</th>
</tr>
</thead>
<tbody>
<tr>
<td>N core</td>
<td>14</td>
<td>16</td>
<td>18</td>
<td>19</td>
<td>25</td>
<td>23</td>
<td>22</td>
<td>20</td>
<td>23</td>
<td>27</td>
</tr>
<tr>
<td>N periphery</td>
<td>78</td>
<td>81</td>
<td>100</td>
<td>109</td>
<td>124</td>
<td>129</td>
<td>124</td>
<td>126</td>
<td>145</td>
<td>156</td>
</tr>
<tr>
<td>Concentration</td>
<td>0.822</td>
<td>0.849</td>
<td>0.880</td>
<td>0.875</td>
<td>0.871</td>
<td>0.861</td>
<td>0.852</td>
<td>0.861</td>
<td>0.837</td>
<td>0.831</td>
</tr>
</tbody>
</table>

The number of core developers was relatively dynamic. There was an increment from Snapshot 1 until Snapshot 5 and then decreased to the minimum in Snapshot 8 and was back to increase in the last snapshot. I also noticed that there was a high correlation between the size of core and the overall team size (corr.=0.95). I suspected this high correlation is due to the rescaling treatment. The relatively high concentration score (above 80%) suggested that the selection for the core developers was well specified. In total, UCINET identified 42 unique actors who were part of the core team.

3.4 Construct the input data sets

In total, the data obtained from the data analysis covered all the developers that worked for the Chrome project. I estimated 274 developers were participated in the development effort from December 2008 until October 2009. However, not all of these developers were suitable for my further analysis. In particular, I am only interested with those who evidently showed a consistent participation in the project so their learning progress is observable. To do so, I conducted a systematical sampling from the population to choose only the long-term developers.

I defined a long-term developer as a participant who has a long commitment to the project. In order to observe a commitment, I traced the contribution of each individual within each release. I used the closing date information to indicate a contribution from a participant and labelled it across the ten
releases. Ideally, I expected long-term developers to be those who were never absent sending a patch in every releases. However, using this ideal expectation, I was able to identify only 23 developers. In order to increase the sample size, I decided to loosen the criteria by having a minimum threshold of contribution within eight releases. The result was 38 developers were selected with a total of 354 project-months of observation; see Table 3.6.

Table 3.6: Developer participation period

<table>
<thead>
<tr>
<th>Months of participation</th>
<th>N participants</th>
<th>Observations</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>11</td>
<td>88</td>
</tr>
<tr>
<td>9</td>
<td>4</td>
<td>36</td>
</tr>
<tr>
<td>10</td>
<td>23</td>
<td>230</td>
</tr>
<tr>
<td><strong>TOTAL</strong></td>
<td><strong>38</strong></td>
<td><strong>354</strong></td>
</tr>
</tbody>
</table>

The result from this sampling decision is obtained with a consequence. The resultant data set became *unbalanced* because each person could have a different number of observation waves due to the threshold. This sampling condition creates a similar complication as when conducting longitudinal study using real experiments. Such complication is called *attrition*, or the problem of subjects leaving the study prior its completion. Attrition creates “holes” in the longitudinal data as no information is available. Singer and Willett [32] mentioned that the resultant data set need not be balanced since most longitudinal studies experience some attrition. They added that as long as the attrition is a random phenomenon, drawing inferences from an unbalanced data set would not be a problem.

Table 3.7 summarizes the measurements of all variables in the input data sets. The descriptive statistics shows that 24 developers were working on 128 issues in the first month, T1, and had finished working on 135 issues. In the second month, 28 developers were working on 119 issues and had finished working on 66 issues. In the third month, 38 developers were working on 210 issues and had finished 215 issues, and so on. The table shows a pattern of improvement in the mean of the performance, showing a decreasing trend from 34 days to 11 days. Interestingly, the mean of the experience is relatively stable between 6 to 10 issues per month, suggesting a constant amount in the issue resolution activity. For the team role division, the table shows a fairly balance number between peripheral members and core members with a slightly higher number of peripheral members.
Table 3.7: Descriptive statistics of variables

<table>
<thead>
<tr>
<th></th>
<th>T1</th>
<th>T2</th>
<th>T3</th>
<th>T4</th>
<th>T5</th>
<th>T6</th>
<th>T7</th>
<th>T8</th>
<th>T9</th>
<th>T10</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Performance</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N</td>
<td>128</td>
<td>119</td>
<td>210</td>
<td>239</td>
<td>295</td>
<td>325</td>
<td>307</td>
<td>241</td>
<td>336</td>
<td>271</td>
</tr>
<tr>
<td>Mean</td>
<td>33.94</td>
<td>34.92</td>
<td>24.36</td>
<td>20.39</td>
<td>20.14</td>
<td>18.83</td>
<td>18.79</td>
<td>15.47</td>
<td>14.80</td>
<td>10.99</td>
</tr>
<tr>
<td>Minimum</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2.37</td>
<td>1</td>
</tr>
<tr>
<td>Maximum</td>
<td>240</td>
<td>83.55</td>
<td>93</td>
<td>103.67</td>
<td>109</td>
<td>86</td>
<td>74</td>
<td>65</td>
<td>61</td>
<td>35.22</td>
</tr>
<tr>
<td>Std.dev</td>
<td>49.52</td>
<td>25.72</td>
<td>22.54</td>
<td>18.69</td>
<td>20.01</td>
<td>15.39</td>
<td>17.02</td>
<td>13.50</td>
<td>10.97</td>
<td>6.56</td>
</tr>
<tr>
<td><strong>Experience</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N</td>
<td>135</td>
<td>66</td>
<td>215</td>
<td>257</td>
<td>294</td>
<td>270</td>
<td>243</td>
<td>294</td>
<td>379</td>
<td>268</td>
</tr>
<tr>
<td>Mean</td>
<td>5.63</td>
<td>2.36</td>
<td>5.66</td>
<td>6.76</td>
<td>7.74</td>
<td>7.11</td>
<td>6.40</td>
<td>7.74</td>
<td>9.97</td>
<td>7.44</td>
</tr>
<tr>
<td>Median</td>
<td>3</td>
<td>1</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>5.5</td>
<td>4</td>
<td>5</td>
<td>7.5</td>
<td>6</td>
</tr>
<tr>
<td>Minimum</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Maximum</td>
<td>33</td>
<td>13</td>
<td>26</td>
<td>49</td>
<td>57</td>
<td>34</td>
<td>37</td>
<td>44</td>
<td>53</td>
<td>25</td>
</tr>
<tr>
<td>Std.dev</td>
<td>8.71</td>
<td>3.06</td>
<td>7.10</td>
<td>8.68</td>
<td>9.57</td>
<td>6.08</td>
<td>6.58</td>
<td>8.11</td>
<td>9.24</td>
<td>6.11</td>
</tr>
<tr>
<td><strong>Team Role</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Core</td>
<td>11</td>
<td>11</td>
<td>15</td>
<td>17</td>
<td>20</td>
<td>18</td>
<td>18</td>
<td>16</td>
<td>16</td>
<td>14</td>
</tr>
<tr>
<td>Periphery</td>
<td>13</td>
<td>17</td>
<td>23</td>
<td>21</td>
<td>18</td>
<td>20</td>
<td>20</td>
<td>22</td>
<td>22</td>
<td>22</td>
</tr>
<tr>
<td>N developer</td>
<td>24</td>
<td>28</td>
<td>38</td>
<td>38</td>
<td>38</td>
<td>38</td>
<td>38</td>
<td>38</td>
<td>38</td>
<td>36</td>
</tr>
</tbody>
</table>

3.5 Data fitting to learning curve models

Most empirical models of learning curve effects assert that performance is a log-linear function of experience [1]. In general, the function relates improvements in productivity to accumulated production experience. In this case, production experience is measured as the number of resolved issues and its accumulation is defined as the knowledge stock. The learning curve effects theorize that as individuals and/or organizations get more experienced at a task, they usually become more efficient on it on each subsequent iteration.

Models of the learning curve effects are commonly expressed using the Power Law, e.g., [28, 1]:

$$\ln Y(t) = a \ln X(t) + b$$

and the Exponential Law, e.g., [24]:

$$\ln Y(t) = aX(t) + b,$$

where in both cases $Y(t)$ and $X(t)$ are measures of performance and accumulated experience respectively at time $t \geq 1$. In this thesis, I followed the equation of Exponential Law where the assumption of no prior experience is
allowed to permit zero values incorporated in the computation. Data about experience are available depend on the production of patches. In the case of newcomers, these data are unavailable until they started to contribute. Hence in the first one or two months of participation, there are exist some zero values for this variable.

As mentioned earlier, my observation is based on longitudinal data. These data comes from surveys of individuals in different time. Each data set forms a two-dimensional matrix, consisting of a subject unit and time variable. The subject unit is the developer and the time variable is the monthly development iteration. Each data set describes a particular observation (e.g., performance, experience). I included the detail description of the model variables in Subsection 3.5.1.

The analysis using longitudinal data allows the inclusion of individual, or subject-specific, parameter. In general, different individuals may be subject to the influences of many factors that make them unique. Listing all the possible factors that contribute to individual behavior is neither feasible nor desirable since the purpose of modeling is to capture the essential ones and not to mimic the reality. However, not to include the individual-specific effects and assuming all individuals are identical may not be a realistic as well. It is typical to leave all these factors to be captured in the subject-specific parameter and to assume the heterogeneity is existed.

To implement subject-specific parameters, one can use dummy variables in the regression model. A dummy variable is a binary variable that has either one or zero. For many longitudinal data applications, the number of subjects can be substantially numerous. Thus, dummy variables are commonly used to identify all the subjects. For example, to represent five subjects, we can apply five dummy variables, \( \{d_1, d_2, d_3, d_4, d_5\} \). To identify the first subject, the dummy variable \( d_1 \) takes a value of one while the rest take zero values, i.e., \( \{1, 0, 0, 0, 0\} \). Subsequently, to identify the second subject, the dummy variable \( d_2 \) is set to one and the other variables are set to zero, i.e., \( \{0, 1, 0, 0, 0\} \). The encoding continues \( \{0, 0, 1, 0, 0\} \), \( \{0, 0, 0, 1, 0\} \), and \( \{0, 0, 0, 0, 1\} \) to identify the third, fourth and fifth subject respectively.

In a similar case, the dummy variables can also be used as categorial predictor variables to examine group effects in the sample. I used this approach to observe the learning difference between core and periphery groups in the Chrome team.
3.5.1 Key variables

The symbols I use throughout the thesis and the variables they represent are listed as follows.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Variable</th>
</tr>
</thead>
<tbody>
<tr>
<td>$i$</td>
<td>Developer Id.</td>
</tr>
<tr>
<td>$t$</td>
<td>Monthly project iterations, $t = {1, 2, 3, ...}$ means iteration for release T1, T2, T3 and so on.</td>
</tr>
<tr>
<td>$P_{it}$</td>
<td>Average time (in days) to resolve open issues by developer $i$ at iteration $t$.</td>
</tr>
<tr>
<td>$e_{it}$</td>
<td>Number of resolved issues by developer $i$ at iteration $t$.</td>
</tr>
<tr>
<td>$K_{it}$</td>
<td>Knowledge acquired by developer $i$ at iteration $t$.</td>
</tr>
<tr>
<td>$R_{it}$</td>
<td>Team role of developers $i$ at iteration $t$.</td>
</tr>
</tbody>
</table>

Variable $P$, $e$ and $K$ are used as measures for the learning curve. Respectively, those variables represent performance, experience and knowledge stock. The role variable $R$, meanwhile, is used to categorize the different team roles in the project team.

3.5.2 Empirical models

Based on the exponential model mentioned earlier, I examined the improvements in performance to accumulated production experience as an evidence of learning phenomenon. The regression function is defined as follows:

$$\ln P_{it} = \alpha + \beta K_{it} + \sum_{j=1}^{37} \phi_j \delta_{ij} + \varepsilon_{it}$$ (3.1)

$$K_{it} = \lambda K_{it-1} + e_{it}$$

where $\ln P_{it}$ is the log of the performance for developer $i$ at iteration $t$ in solving the issues, $K_{it}$ is the knowledge stock for each developer and $\beta$ is the regression coefficient for the estimation of learning effects. The knowledge stock $K$ increases as long as the developers keep actively resolving the open issues. Additionally, I included a depreciation factor $\lambda$ to indicate a possibility that knowledge can devalue over time. The parameter $\lambda$ must lie in the interval $[0, 1]$. If $\lambda = 1$ then the accumulated knowledge stock is simply equal to the lagged cumulative resolving experience. By estimating $\lambda$, I obtained an estimate of knowledge loss over time\(^6\). The model also contains the

---

\(^6\)The estimation for the lambda follows the search procedure introduced by Argote [1]. For each chosen value of lambda, the remaining parameters are estimated by standard procedures for estimating regression models, such as the least square estimation. The search begins with
subject-specific parameter to control heterogeneity among individuals. The coefficient $\phi$ represents the vector of individual’s variables that may not be observable but are relatively constant through time, e.g., individual ability. The expression denotes the usage of dummy variables for distinguishing the different developers.

I also examined the difference between core and peripheral groups in the learning process. Participants in different roles may experience different access to people because the communication structure is dissimilar. Core developers are primarily in the central “hub” and all the contributors are connected to and through this hub [9]. This situation introduces a different level of information access and, subsequently, leads to different learning experience. My next approach is to use group dummies $R$ to capture the role effects in the learning process, as follows:

$$\ln P_{it} = \alpha + \beta K_{it} + \sum_{j=1}^{37} \phi_j \delta_{ij} + \gamma R_{it}^{periphery} + \varepsilon_{it} \quad (3.2)$$

where $R_{it}^{periphery}$ is a dummy variable that takes on a value of one if the developer $i$ is categorized in the peripheral group at iteration $t$. The parameter $\gamma$ reflects the regression coefficient for estimating which group is better at learning and how wide the difference is.

---

a survey over value of $\lambda$ at increments of 0.05 in the interval $[0, 1]$ to identify the subinterval in which the function reaches the minimum mean square residual. Later, another search is performed within the subinterval at increments of 0.01 for further accuracy.
Chapter 4

Results

4.1 The presence of organizational learning

Results on organizational learning are presented in Table 4.1. All regressions used the same longitudinal data of ten months observation. Regression (1) shows the first model in the previous Section 3.5.2. A negative sign in the knowledge stock coefficient indicates an increase in the knowledge stock generates a decrease in the resolution time. This finding is consistent with the description of learning curve effects, that is, the accumulated experience can generate improvements in the productivity. Thus, this finding suggests the presence of learning phenomenon among Chrome developers.

As I have mentioned earlier, learning may experience depreciation. In practice, knowledge can depreciate for some natural causes, for example, people may forget how to repeat their tasks due to their complexity or the knowledge becomes obsolete and useless thus it is neglected. Argote and colleagues [1] found a rapid depreciation in knowledge acquisition and they added that this downturn factor is much more rapidly affect knowledge persistency rather than the rate of labor turnover. However, according to my result, knowledge depreciation was not the case for my particular study. The search procedure estimated for $\lambda$ in Regression (1) is 0.94. The model fitting does not yield a standard error for $\lambda$. Nevertheless, using the distribution of the likelihood ratio, I have determined that a 85.6% confidence interval for $\lambda$ is roughly between 0.85 to 1.00. This result is in contrast with the observation reported by Argote and colleagues [1], where it showed a significant depreciation between 65 to 85 percent among the ship workers.

4.2 Learning between core and periphery

Regression (2) repeats Regression (1) with an inclusion of roles parameter. This parameter examines the common learning characteristics among devel-
Table 4.1: Model summary results and coefficient estimates

<table>
<thead>
<tr>
<th></th>
<th>Learning Curve with Fixed Effects Estimator (1)</th>
<th>Learning Curve with Fixed Effects Estimator and Role Variable (2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Constant</td>
<td>$3.50601^{***}$ (11.178)</td>
<td>$3.50698^{***}$ (11.177)</td>
</tr>
<tr>
<td>Knowledge stock</td>
<td>$-0.01015^{***}$ (-4.106)</td>
<td>$-0.01017^{***}$ (-4.115)</td>
</tr>
<tr>
<td>$\lambda$</td>
<td>0.94</td>
<td>0.94</td>
</tr>
<tr>
<td>Roles</td>
<td>NA</td>
<td>0.18095 (0.889)</td>
</tr>
<tr>
<td>Adjusted $R^2$</td>
<td>0.0481</td>
<td>0.0478</td>
</tr>
<tr>
<td>$N$</td>
<td>354</td>
<td>354</td>
</tr>
</tbody>
</table>

Significance code: $^{***} p < 0.001$  $^{**} p < 0.01$  $^* p < 0.05$

opers of the same group. There are two roles defined in this study: core and periphery. Depending on the number of interactions and connections, the role of a developer was estimated using a social network model by Borgatti and Everett [4]. It should be noted that this variable is not time-invariant but, in fact, roles can change quite dynamically in open source communities (Martinez-Romo et al. [26] reported a similar result in case studies using Evolution and Mono). I observed several cases in which a developer moved from the peripheral group into the core team, and vice versa.

To make sure the new variable did not introduce multicollinearity problem, I ran pairwise correlation tests between the predictor variables. This required calculating the point-biserial correlation coefficient between the role and the knowledge stock variable, and the Pearson’s contingency coefficient between the role and developer’s subject-specific variable. The results are shown in Table 4.2. The highest pairwise correlation coefficient is 0.655 which is below the upper threshold commonly used of 0.6 to 0.8 suggested by Kennedy [20]. According to the tests, I concluded there was no multicollinearity introduced in the model.
Table 4.2: Pairwise tests results

<table>
<thead>
<tr>
<th>Predictor1</th>
<th>Predictor2</th>
<th>Test name</th>
<th>Correlation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Role</td>
<td>Knowledge</td>
<td>Point-biserial</td>
<td>0.1505</td>
</tr>
<tr>
<td></td>
<td></td>
<td>coefficient</td>
<td></td>
</tr>
<tr>
<td>Role</td>
<td>Developer</td>
<td>Pearson’s</td>
<td>0.6550</td>
</tr>
<tr>
<td></td>
<td></td>
<td>contingency</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>coefficient</td>
<td></td>
</tr>
</tbody>
</table>

It was hypothesized that different project roles had different information access thus suggesting different learning experience. The analysis compared groups in the core (as the reference group) to the periphery by using dummy variables. The positive coefficient sign shows that the periphery group on average requires 0.18 log days, or 18%, more than the core group to resolve issues. In other words, core developers have better service time in resolving project issues compared to the peripheral group. However, it is important to notice that the difference is not statistically significant. Thus, this proposition receives very little support and no conclusion was able to be drawn.

Following on my previous result, it appears that performance is more likely related to the accumulated experience. Hence, learning is primarily focused on specialization. In his classical paper, Vygotsky [37] defined learning as the acquisition of specialized ability. Specialization also plays an important role to the key success in the open source software innovation process [36]. In the open source development, specialization means working on a particular software module. I found several members were assigned to the Chrome security team or the Chrome frame module, forming special groups of experts. Unfortunately, I did not conduct an observation that relates specialization and performance. A future study in this direction may benefit to evaluate how specialization in different modules can affect learning processes.
Chapter 5

Research Validity

Although the findings are encouraging and useful, the present study has certain limitations. I address these limitations as threats to the research validity, following the classification cited in Wohlin et al. [39]. There are four categories of research validity namely as follows:

1. **Conclusion validity.** This validity is concerned with whether there is a reasonable conclusion about a relationship in the observations. It is sometimes referred to as statistical conclusion validity because the conclusion reached depends on the choices of the statistics. For example, it is often hard to see the relationship because the measurements are too weak to handle the “noise” in the environment. Or, there is too less information to make the relationship obvious.

2. **Internal validity.** This validity has a close connection to the conclusion validity in which they are related in predicting the research conclusion. However, the distinction is that the internal validity deals whether the observed changes can be attributed to the selected variables and not to other possible causes.

3. **Construct validity.** This validity refers to the extent to which the investigation setting actually reflects the phenomenon under study. It can be the approach of the measurements for the independent variables as well as the dependent variables. For instance, the number of courses at the university in computer science may be a poor measure to know one’s expertise in programming language. The number of year of practical work may be a better measure to collect those data.

4. **External validity.** The external validity is concerned with generalizing. It refers to the approximate truth of the conclusion when it applies to a broader scope (i.e. generalization). Since the sample of the study is a representation of the population, it is then possible to generalize the results back to the population. In this case, the problems with external
validity is mostly related to the fairness in conducting the sampling and whether the generalization can be applied across different time.

In the following sections, I discuss several possible threats that can influence the validity of the results and the final conclusion. I framed the discussion in terms of potential risks in data sampling, study construction and generalization, and presented some arguments and/or evidence to understand the pitfalls.

5.1 Internal validity

5.1.1 The issues became easier to maintain because of the improvement in the system design.

The issue data set covers an observation within ten months period and it contains the history from the early development until the quite recent version release. An early development stage usually begins with an incomplete software design of the product and, therefore, the code was harder to maintain. Given the claim that the open-source approach can lower the overall complexity by the iterative approach, we would expect a reduction time for solving emergent issues. Consequently, the observed improvement might have been mediated by the fact that the issues become simpler and easy to maintain instead of learning effects per se.

Unfortunately, there is no information about the issue complexity to test the assertion above. However, I can tell the presence of the complexity by observing the side effects of solving recent issues. If the issues became easier to maintain, then there would not have been differences in the solving time between the developers who had joined the project for a long time and the developers who just recently joined the community.

To test this hypothesis, we made a comparison of two groups of developers. The first group was the newcomers who came late to the development process. We took samples from those who initially contributed to Google Chrome version 3.0 (July-October 2009). In total, we collected 44 new developers. The second group is the old-timers who are the 38 sample subjects of this study. Both groups are disjoint, that is, the two groups have no members in common. We collected the issue resolution time for version 3.0 in each group and drawn the distribution.

To test this hypothesis, I made a comparison of two groups of developers. The first group was the newcomers that include the new developers who start contributing for the development of version 3.0 (July–October 2009). The second group is the old-timers who are the sample subjects of this study. Both
Figure 5.1: The resolution time distribution between newcomers and old-timers. Descriptive statistics summaries: (1) Newcomers: \( \text{Mean}=26.57, \text{Min}=1, \text{Max}=156, \text{Median}=12.62, \text{SD}=32.06, N=44 \), (2) Old-timers: \( \text{Mean}=15.06, \text{Min}=1, \text{Max}=74, \text{Median}=11.06, \text{SD}=13.02, N=38 \). The “X” sign indicates the position of the mean value.

Groups are disjoint, that is the two groups have no members in common. Figure 5.1 shows the distributions of the average resolution time between the newcomers and old-timers. In comparison, the old-timers have smaller mean and slightly smaller median of resolution time than the newcomers.

I applied a non-parametric test to discover whether the distributions in the solving time from the two groups are different. The Kolmogorov-Smirnov test gave \( p \)-value = 0.007476 for 95% confidence interval (two-tails), indicating a statistically significant difference between the two groups. The test also indicated that the old-timer group performed better than the newcomers by having the fewer solving time. This result suggests that learning effects might be the explanation for the improvement happened in this particular case study.

5.1.2 The issue data are incomplete.

Even though I selected software modules that had issue data with good quality, I was only able to extract a subset of the total number of issues (estimated only 22 percent of those stored in the issue tracking database). The data quality also varied across the monthly extraction. Among the selected issues, only 29 percent has all the essential information regarding the issue history (i.e., the opening date, assignment date, starting date and closing date). It is possi-
ble that the time estimation is less precise for the most remaining issues that miss all the information. Hence, it is likely the high variability in the resolving time data has impacted the very low prediction power of the model.

5.2 Construct validity

5.2.1 The estimation of core/periphery structure may be misleading.

To assess the robustness of the core/periphery structure analysis, I compared the results with different approaches in the literature. The data sources come from the formal membership list and the developer mailing list. The usage of these artifacts has been supported by studies of Crowston et al. [10] and Moon and Sproull [27]. I was able to reproduce their results using data from the Google Chrome project.

The comparison followed the timetable of the tenth snapshot in my data analysis. Therefore, I collected sample data from September 2009 for the other artifacts. Unfortunately, I collected the membership list quite late in April 2010. Having a gap between the two data collection points may slightly increase the chance of a mismatch in the analysis.

Data collection. The data collection was performed manually. There were 183 developers needed to be cross-checked by reading and searching using the names keyword. The membership list was quite clear in describing names, roles and duties. The list separates developers into three roles: project owners, committers and contributors. For each name, I searched for a match and recorded his role in a separate spreadsheet. During the recording, several names had no match in the list and, thus, I decided to assigned them as the contributors. In the end, I identified 6 owners, 143 committers and 34 contributors using the list.

The developer’s mailing list was traced from September 2008 until September 2009. Fortunately, the Google chromium-dev mailing list provided a summary of the posting activity by each month so it was feasible to collect the data manually. Again, I did searches using the names keyword and recorded the posting summary in each month in a spreadsheet. In the end, I summed the total posting for each name to get the overall posting activity.

Data analysis. I use two different approaches for the data analysis. The first is the self-report on the membership list as demonstrated by Crowston et al. [10]. However, due to different practices in assigning roles, I modified the
Table 5.1: Message contribution summary

<table>
<thead>
<tr>
<th>Messages</th>
<th>Contributors</th>
<th>% of total contributors</th>
<th>Cumulative message count</th>
<th>Messages contributed</th>
<th>% of total messages</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>56</td>
<td>28.96%</td>
<td>0</td>
<td>0</td>
<td>0%</td>
</tr>
<tr>
<td>1–5</td>
<td>38</td>
<td>18.58%</td>
<td>106</td>
<td>106</td>
<td>1.68%</td>
</tr>
<tr>
<td>6–10</td>
<td>15</td>
<td>10.93%</td>
<td>230</td>
<td>124</td>
<td>1.95%</td>
</tr>
<tr>
<td>11–50</td>
<td>39</td>
<td>21.86%</td>
<td>1251</td>
<td>1021</td>
<td>16.09%</td>
</tr>
<tr>
<td>51–100</td>
<td>17</td>
<td>9.29%</td>
<td>2457</td>
<td>1206</td>
<td>19.01%</td>
</tr>
<tr>
<td>100+</td>
<td>12</td>
<td>6.56%</td>
<td>4263</td>
<td>1806</td>
<td>28.47% *</td>
</tr>
<tr>
<td>200+</td>
<td>2</td>
<td>2.18%</td>
<td>4708</td>
<td>445</td>
<td>7.01%</td>
</tr>
<tr>
<td>300+</td>
<td>3</td>
<td>1.09%</td>
<td>5773</td>
<td>1065</td>
<td>16.79%</td>
</tr>
<tr>
<td>400+</td>
<td>1</td>
<td>0.55%</td>
<td>6344</td>
<td>571</td>
<td>9.00%</td>
</tr>
</tbody>
</table>

Table 5.2: Comparison between approaches

<table>
<thead>
<tr>
<th>C/P Analysis</th>
<th>Self-report</th>
<th>Message postings</th>
</tr>
</thead>
<tbody>
<tr>
<td>Core</td>
<td>27</td>
<td>Owner</td>
</tr>
<tr>
<td>Periphery</td>
<td>156</td>
<td>Committer</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Contributors</td>
</tr>
</tbody>
</table>

The assumption of core and periphery. I identified individuals as core if they were formally recognized as the project owners instead of the developers as in the article. Project owners typically have a managerial position, e.g., lead developer, project manager or product manager.

The second is the message posting analysis using posting data from the mailing list. I followed the message counting method as introduced by Moon and Sproull [27]. Table 5.1 shows the summary of posting distribution in the chromium-dev mailing list. I discovered approximately 10% of the participants contributed more than half of the total postings (in comparison, Moon and Sproull acknowledged only 2% of the contributors in the linux-kernel mailing lists). I assigned these 10% of participants to the core group and the remaining to the periphery group.

Result and Discussion. The identification of core group using the various methods shows some interesting differences; see Table 5.2. In general, all methods identified a small portion of central actors (around 3%–15%).

Table 5.3 presents a cross-tabulation to show the summary between different approaches. To represent the 3-dimension of the $2 \times 2 \times 3$ comparison, the table first shows the estimations of core/periphery analysis in the first column and follows by the formal roles across. In addition, each cell is then divided by the poster groups using the second column. For example, the table shows 34 peripheral members (the lower right-most cell) where they are consistently
identified as the contributors in the membership list and have less posting in the mailing list. A closer examination on these results gives a further insight about types of individual in the team.

- **Different functions among project owners.** The table shows two distinct groups of owners that are located in the opposite margin. Each group has the same size of three members. One group resides on the top left-most cell (i.e., darin, jam, ben) and the other resides on the lower left-most cell (i.e., mal, ian, laforge). Further investigation has revealed that members in the upper group are developers (probably the lead developers) and the lower group are actually the managers. This fact explains why some of the owners are recognized as the periphery and while some others are recognized as the core in my model analysis. I can argue that managers typically work in the higher level of the project and rarely discuss about technical details. Therefore, they lack of connections and are excluded from the core group in my analysis. This claim is also supported by other fact that the managers have less message posting count (least poster) compared to the lead developers (most poster) in the mailing list data.

- **There is a wide variety in the committer role.** Two kinds of committer present in the table as well. I named them as core committers and peripheral committers. To add more explanation to this distinction, I collected their commit frequency and conducted a comparison. I discovered that core committers contributed almost half of the total commits (49.6%), indicating their active contributions in the project. This distinction has been speculated by Crowston and Howison [9] that suggested the hierarchy of open source community as a concentric circle. It begins from the core developers in the center and then towards the periphery layers of co-developers, active users, bug reporters and end-users. The distinction between core committers and periphery committers is similar
in respect to co-developers and active users, suggested by the hierarchy.

- **Roles in knowledge sharing.** I also noticed an incompatible result between the approaches using message posting and core/periphery structure analysis. The figures seem counterintuitive and opposite. The most posters (MP) are identified as the periphery and the least posters (LP) are identified as the core. As an explanation for this finding, Sowe et al. [34] suggested that there is another role in open source community that is related to knowledge sharing, called the *knowledge providers*. People in this role are very sound in mailing lists as they serve answers for questions from the community. They are not necessarily active in the development labor but we may still think of them as the central actors in terms of knowledge sharing.

- **Reciprocity drives a temporary participation.** All approaches show an agreement in identifying temporary participants in the lower right-most cell. Shah [31] commented that there are some people who participate because they have immediate needs and there is an implicit obligation or a desire to return the good deed from others. He added: “Contributions triggered by reciprocity generally occurred over a short time span and consisted of answering questions and contributing patches or small bits of code. [...] After contributing, most removed themselves from the mailing lists and did not keep up with software-related developments.”

To give more certainty to the findings above, I replicated the analysis for other two snapshots (i.e., seventh and fourth). In general, the pattern is similar to what I have described in this section; see Appendix D for more details.

The definition of core clearly contains many interpretations. I found three ways to identify core in which each carries a different viewpoint. Nevertheless, having different perspectives enrich the understanding of the team structure. I have shown you that the overlaps can be explained well using results from other studies. Therefore, to some extent, I concluded that the core/periphery structure analysis used in this study is reasonably precise and its prediction might reflect the actual situation.

### 5.3 External validity

#### 5.3.1 Each model has a very low statistical prediction power

All of my models are severely afflicted by a very low statistical prediction power. The coefficients of determination among the three models are less
than 5 percent which indicate a poor fit to the data. These deviations from the model implies that the model is not representative and useful but, indeed, misleading for this particular case study. However, the high significance level in the regression coefficients between performance and knowledge stock shows that the association is strong enough to be concrete and not to be attributed to chance.

5.3.2 The sample size might not be representative

We are aware also that the identification of sample subjects introduced in the earlier section resulted in the removal of about 81 percent of the total participants in January–October 2009. These individuals were removed because their participation was below the threshold of eight months. This means that developers who intensively participate in short period were excluded in this study. The removal was necessary in order to account for longitudinal observations of learning effects. Specifically, I was more interested using the long-term developers rather than the occasional developers. I posit that these excluded developers might also experience learning because of the benefit from collaborative works.

Generalizing the results of this study and making claims about situated learning in the community of developers needs an incorporation of additional supports. This is because the uniqueness of the Chrome project makes it difficult to find other projects that use similar practices. It should be noted that the Chrome project uses tools that are mainly different with the majority of open source projects, for example, projects that are facilitated by Sourceforge. However, this peculiarity does not mean my study is isolated from other similar studies. In contrast, this study completes the spectrum of investigation regarding learning phenomenon in communities of developers (refer back to Section 2.3 about the motivation of this study). Thus, the generalization of this study can be alleviated to some extent by merging the results from different investigations. In the end, I consider the conformity of the results of my study to be good.
This thesis made two contributions to the open source software engineering research community. First, it affirms that learning is present among the open source developers. The study follows a model of learning curve effects that associates performance with the accumulated experience. I found a strong relationship between the two variables and confirmed the presence of learning curve. It is likely also, that learning in open source projects means owning a specialization rather than trying to cope the whole project. I noticed team subdivisions in which each sub-team handled a certain module, creating several specialization for administering the code artifacts. Moreover, specialization is mentioned as one key success for developers to survive in open source projects [36]. In addition, another investigation was also conducted around the organization structure to investigate whether core and peripheral members have different working speed. The finding was inconclusive to claim both groups have different means towards issue resolution and, thus, no more arguments are added.

Second, this thesis describes an alternative way to identify core and periphery structure in the project team. I developed a method for collecting interaction data from the code review system and evaluating those joints using social network analysis. The data collection is based on direct comment exchanges between developers which is proven to be more accurate than the existing approaches (e.g., using the mailing lists [10], [16]). The method also includes a way for data improvement in case outliers are detected, in which I introduced a rescaling model. Lastly, to ensure the method is reliable, I conducted a comparison study using different project artifacts, namely, the membership list [10] and mailing lists [27]. I found a reasonable agreement in the results between my method and the alternatives. To my knowledge, this thesis is the first to use code review as its data source to investigate the team structure of open source project. I believe code review will be another important data source for open source research in the future.

I propose two areas for future work. First, to strengthen the empirical re-
sults found in this study, it would be beneficial to replicate this study using more sample projects. Although I found the findings are consistent with the existing literatures, there is a fundamental difference in the data collection approach. The existing approach focuses on collecting data from the team scale which summarizes the individual details and gives common information. However, in my opinion, individual details are important as people are unique in nature and should not be abstracted. Hence, a better way to observe the gradual effect of learning in organizations is by first employing a systematical sampling of individuals and then running the observation upon those selected people. I perceived this approach creates a control strategy especially for teams with constant growth in their size. It should be noted that open source communities are typically dynamic over time and highly susceptible to changes and, therefore, summarizing these data may not be well interpreted in the repetitive observations.

The second area for future work is to conduct further studies regarding learning depreciation. In this study, I found less supportive evidence that learning is depreciated. My result shows a very slight depreciation in the knowledge among open source software developers. It was estimated to be 94 percent, compared to other studies which ranged between 65 to 85 percent. Nevertheless, this result may give an insight that learning in producing intangible commodity such as software has a unique characteristic compared to producing tangible commodity. To some extent, the different kinds of product output may possibly influence the knowledge retention among the workers. This proposition needs to be tested by further studies using different samples.
Appendix A

Collecting Issue Report Data

Figure A.1: A sample snapshot of issue report. An issue report consists of four main parts: header, metadata, description and comments area. The header and metadata provide the profile information of the issue, e.g., ID number, latest status, owner, type, area, priority, and so on. The description outlines the source of problems or request motivations from the reporter. And the comments area serves as a discussion board between developers where it retains the issue history.

The Google Chrome project utilizes issue tracker system to store and maintain all project-related tasks that come from users and developers. Each issue
represents a work item that project members must resolve and it is not limited only to bug reports but also feature requests, technical-support requests, development tasks and others. Communication between developers is done via a web interface where the posted texts are made open to public. Figure A.1 shows a sample snapshot of Chrome issue report.

![State diagram of the Chromium issue life cycle](image)

**Figure A.2:** State diagram of the Chromium issue life cycle

The status of an issue is a one-word indication of how far the issue has progressed through an expected issue life-cycle. An issue life-cycle can be thought of as a state machine shown in Figure A.2. The process starts with a issue report submitted by a user or developer. Once received, the bug is marked as **New** until a project member triaged the issue and upgraded its status to **Available**. The submitted issues can also be rejected if they were invalid, cannot be reproduced or duplicates. An available issue is a ready work item for anyone to resolve. An **Assigned** status marked the ownership for an issue and when it is actively being worked on, it gets upgraded to **Started**. When the developers considered the work to be completed, the issue is marked as **Fixed**. Upon confirmation by a quality assurance person, the issue is then marked as **Verified**. If verification failed, or if the original reporter added a comment to say that the solved issue was still a problem, the issue can be reopen to **Available** status.

Another important element is the issue labels. Issue labels give brief profile to the issue reports so others can browse quickly or make some groupings. The issue reporters can introduce new labels and thus they are varied and numerous. However, most reporters used the existing “standardized” labels, such as **ID**, **Priority**, **Area**, **Type**, **Milestone** and **OS**.

### A.1 Source code

<table>
<thead>
<tr>
<th>Filename</th>
<th>robo_issue_parser.py</th>
</tr>
</thead>
<tbody>
<tr>
<td>Language</td>
<td>Python</td>
</tr>
<tr>
<td>Description</td>
<td>Collect data from the Chrome Issue Tracker repository.</td>
</tr>
</tbody>
</table>
Appendix B

Collecting Code Review Data

One of the best practices that has been introduced in the Chromium project is code review. This is an essential step in contributing code prior to the code commit. Preparing a code review is managed using the tool provided by the project. After writing a patch, a contributor must create a change list that describes what the patch changes and why. This is important for people who are looking at change logs in the future to track down an issue. Later, the patch and the change list are uploaded to the online code review page at http://codereview.chromium.org. Later, the script will crawl and parse the texts from these pages to get the interaction data.

Figure B.1: A snippet of code review document
Figure B.1 shows a snippet of code review document. I omitted several markups for simplicity. This page is generated by accessing the RSS-XML version where the document is well-structured, thus, making the text parsing possible. The figure shows several highlights and notes to give an illustration how the script will work.

### B.1 Source code

There are two scripts that used for collecting code review data. The first script is used to fetch the review links from SVN change logs. The second script is used to parse the review page using the links collected previously.

<table>
<thead>
<tr>
<th>Filename</th>
<th>robo_link_collector.py</th>
</tr>
</thead>
<tbody>
<tr>
<td>Language</td>
<td>Python</td>
</tr>
<tr>
<td>Description</td>
<td>Collect issue URLs from Chrome change logs.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Filename</th>
<th>robo_review_parser.py</th>
</tr>
</thead>
<tbody>
<tr>
<td>Language</td>
<td>Python</td>
</tr>
<tr>
<td>Source code</td>
<td><a href="http://code.google.com/p/chrome-data-collector-robo/source/browse/Python/robo_review_parser.py">http://code.google.com/p/chrome-data-collector-robo/source/browse/Python/robo_review_parser.py</a></td>
</tr>
<tr>
<td>Description</td>
<td>Collect interaction data from the Chrome code review transcripts.</td>
</tr>
</tbody>
</table>
Appendix C

Working with UCINET 6

C.1 Introduction

UCINET is a comprehensive package for the analysis of social network data. The software is distributed by Analytic Technologies and can be used for free up to 60 days. This guidelines uses UCINET 6 for Windows (6.232) (see Figure C.1) where we are going to focus on the core/periphery analysis functions.

Figure C.1: Main interface of UCINET 6 for Windows

1URL http://www.analytictech.com/
The simplest method for entering network data to be analyzed by UCINET is to type them into a text file. UCINET uses the Data Language (DL) for its standard data writing although many others are available. Table C.1 shows several important fields in DL schema.

Table C.1: Data Language schema

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DL</td>
<td>The “DL” at the beginning is required which it tells UCINET that this is a Data Language file, as opposed to some other kind of file (like UCINET 3.0 format).</td>
</tr>
<tr>
<td>n=</td>
<td>It indicates the number of nodes in the network.</td>
</tr>
<tr>
<td>format=</td>
<td>It specifies which format is used (“fullmatrix”, “edgelist1”, “nodelist1”).</td>
</tr>
<tr>
<td>labels=</td>
<td>It lists the name of the nodes in sequential order.</td>
</tr>
<tr>
<td>labels embeded:</td>
<td>It specifies that the labels are included in the data (optional).</td>
</tr>
<tr>
<td>data</td>
<td>It indicates the end of header information and the following lines are the data itself.</td>
</tr>
</tbody>
</table>

In order to give flexibility on writing the network data, three different formats are offered in the DL specification.

- **Full-matrix format**—This format requires you to type the entire network adjacency matrix. The values must be separated by at least one space and you may not use commas or other punctuation. Figure C.2 shows an example of using fullmatrix format.

![Figure C.2: An example of fullmatrix format](image)

- **Edgelist1 format**—This format allows you to type the pairs of actors that are connected. Each line of data is an ordered pair of actors and, optionally, followed by a value indicating the strength of the tie (or the weight). Figure C.3 shows an example of using edgelist1 format.
• **Nodelist1 format**—This format allows you to type the name of a node and then followed by the other names of every node that they are connected to. However, this format does not support weighting the ties. Figure C.4 shows an example of using nodelist1 format.

![Figure C.3: An example of edgelist1 format](image1)

**Figure C.3: An example of edgelist1 format**

![Figure C.4: An example of nodelist1 format](image2)

**Figure C.4: An example of nodelist1 format**

## C.2 Step-by-step guidelines

### C.2.1 Creating sociomatrices from snapshot data

1. **Creating a DL file.** I created a DL file using the snapshot data file where I added some lines of headers according to the edgelist1 format; see Figure C.5.

   ![Figure C.5: A snippet from Snapshot-10 data](image3)

   **Figure C.5: A snippet from Snapshot-10 data**
2. **Import DL files to UCINET.** UCINET uses two types of file for analysis processing, i.e., *.##D and *.##H. Both files are machine readable documents where only UCINET can understand. To import DL files to UCINET, choose: *Data > Import text file > DL*.... A dialog window will pop up as shown in Figure C.6.

![Figure C.6: A dialog window to import DL file](image1)

3. **Symmetrize the adjacency matrix.** I defined the relation between actors is a bi-directional communication where both parties engaged the conversation together. To represent this notion into my sociomatrix, I symmetrized the data by choosing *Transform > Symmetrize*.... A dialog window will pop up as shown in Figure C.7. I selected “Sum” as the Symmetrizing method to aggregate any cyclic connections.

![Figure C.7: A dialog window to symmetrize an adjacency matrix](image2)

Accordingly, this function will produce two output files: *-Sym.##D and *-Sym.##H*, by default. Figure C.8 shows a symmetrized sociomatrix produced by UCINET. The rows and columns display the actors and the cells represents the weight of the relations between two actors.

### C.2.2 Using core/periphery functions

To start the computation, choose: *Network > Core/Periphery > Continuous*. Figure C.9 shows the dialog window to input the parameters. The experiments were conducted by changing the input dataset while the other parameters were fixed.
Appendix C. Working with UCINET 6

56

Figure C.8: A sample of sociomatrix

Figure C.9: A dialog window to input the computation parameters

- **Input dataset**—Network data file name.

- **Positive or Negative Data**—Use POSITIVE to indicate that larger values imply a stronger relationship. Use NEGATIVE to indicate that larger values in the data imply a more distant relationship.

- **Algorithm**—Which algorithm of fit to use. CORR measures the correlation between the network data matrix and the product of C and C transpose, where C is the coreness matrix. DISTANCE uses Euclidean distance instead of correlation. MINRES is factor analysis where data in diagonal are meaningless.

- **Prevent Negatives**—The matrix C may contain negative values. Choosing YES prevents this happening.

- **Max # of iterations**—The maximum number of iterations used in the optimization procedure. By default, the value is 1000.

- **Diagonal values valid**—Whether data in diagonal are ignored. Choosing NO means diagonal data are meaningless.

- **Output dataset, Output partition, Output concentration**—The output file names. Each file shows the coreness score for every actor node, the estimation of core members and the concentration measures, respectively.
C.2.3 Reading the program outputs

1. **Interpreting output coreness.** This output shows the values of coreness of each actor where these values have been normalized so that the sum of squares is equal to one. Figure C.10 shows a sample output where the values are in descending order.

<table>
<thead>
<tr>
<th>Coreness</th>
</tr>
</thead>
<tbody>
<tr>
<td>aido</td>
</tr>
<tr>
<td>phelan.p</td>
</tr>
<tr>
<td>tony</td>
</tr>
<tr>
<td>thankgs</td>
</tr>
<tr>
<td>brother</td>
</tr>
<tr>
<td>phasing</td>
</tr>
<tr>
<td>sky</td>
</tr>
<tr>
<td>darin</td>
</tr>
<tr>
<td>tner</td>
</tr>
<tr>
<td>jam</td>
</tr>
<tr>
<td>maffin</td>
</tr>
<tr>
<td>sgi</td>
</tr>
<tr>
<td>tozhe</td>
</tr>
<tr>
<td>ndmi</td>
</tr>
<tr>
<td>dekia</td>
</tr>
<tr>
<td>dikkay</td>
</tr>
<tr>
<td>cpu</td>
</tr>
<tr>
<td>bner</td>
</tr>
<tr>
<td>ben</td>
</tr>
</tbody>
</table>

   *Figure C.10: A sample of output dataset*

2. **Interpreting output partition.** This output shows which actors are part of core and periphery group (1=core, 0=periphery). Figure C.11 shows some part of the result where aa, ben, brettw, and cpu are included in the core group.

   |    |
   | aa | 1 |
   | abarth | 0 |
   | aei | 0 |
   | lagr | 0 |
   | gal | 0 |
   | gilberts | 0 |
   | amanda | 0 |
   | bali | 0 |
   | amanta | 0 |
   | antonne | 0 |
   | appadox | 0 |
   | ak | 0 |
   | assignme | 0 |
   | delegation | 0 |
   | aev | 0 |
   | outbrain | 0 |
   | ben | 1 |

   *Figure C.11: A sample of output partition*

3. **Interpreting output concentration.** This output shows several concentration measures which try to assess the degree to which the network falls into a core/periphery structure for different sizes of core. There are four different measures where each measure starts with the actor
with the highest coreness score and places them in the core and all other actors are placed in the periphery. The core size is then successively increased by moving the actor with the highest coreness score from the periphery into the core. This is continued until the periphery consists of a single actor.

![Figure C.12: A sample of output concentration](image)

- **nDiff**—sums the difference between the actor with the highest score in the periphery and all the actors in the core and adds to this the sum of the differences between the actor in the core with the lowest coreness score with all those in the periphery. Each value is then normalized so one does not dominate the other simply by the number of actors it contains.

\[
\sum_{j=1}^{n} (c_i - \max(c_{j+1}, c_{j+2}, \ldots, c_n)) + \sum_{k=j+1}^{n} (\min(c_1, c_2, \ldots, c_j) - c_k)
\]

where there is the collection of \(n\) actors, and that the actors have been arranged in descending order based on \(C\) so that \(c_1 \geq c_2 \geq \ldots \geq c_n\). And let the first \(j\) actors comprise the membership of core.

- **Diff**—is similar to nDiff but places a weighting on the size of the core. The weighting is equal to the square root of the core size. The manual describes Diff as a biased measure where it gives more weight to smaller cores. This measure needs careful interpretation.

- **Corr**—measures the correlates between the given coreness scores with the ideal scores of a one for every core member and a zero for actors in the periphery.
• **Ident**—measures similarly like Corr but uses Euclidian distance instead of correlation.

Figure C.12 shows the most optimum solution is by having the core size as 27 members as been indicated by a clear maxima in nDiff measure. The Corr measure can indicate an area in which to focus and the other measures can be used to fine tune the measure to identify a core size. For example, there are 2 strong candidates for other core sizes which are 25 and 30 (see the highlights). Both sizes have opposite advantages in nDiff and Corr measures where the user can choose properly based on a specific context or other considerations.
Appendix D

Miscellaneous Tables

D.1 Model summary from the alternate experiment

Table D.1: Model summary results and coefficient estimates

<table>
<thead>
<tr>
<th>Learning Curve with Fixed Effects Estimator Variable</th>
<th>Learning Curve with Fixed Effects Group Category Variable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Constant</td>
<td>Constant</td>
</tr>
<tr>
<td>(1)</td>
<td>(2)</td>
</tr>
<tr>
<td>3.87637***</td>
<td>3.87731***</td>
</tr>
<tr>
<td>(14.257)</td>
<td>(14.354)</td>
</tr>
<tr>
<td>Knowledge stock</td>
<td>Knowledge stock</td>
</tr>
<tr>
<td>(1)</td>
<td>(2)</td>
</tr>
<tr>
<td>-0.03195***</td>
<td>-0.03208***</td>
</tr>
<tr>
<td>(-4.276)</td>
<td>(-4.320)</td>
</tr>
<tr>
<td>( \lambda )</td>
<td>( \lambda )</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Roles</td>
<td>Roles</td>
</tr>
<tr>
<td>NA</td>
<td>0.40641*</td>
</tr>
<tr>
<td>(2.263)</td>
<td>(2.263)</td>
</tr>
<tr>
<td>Adjusted ( R^2 )</td>
<td>Adjusted ( R^2 )</td>
</tr>
<tr>
<td>0.0522</td>
<td>0.0647</td>
</tr>
<tr>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>354</td>
<td>354</td>
</tr>
</tbody>
</table>

Significance code: *** \( p < 0.001 \)  ** \( p < 0.01 \)  * \( p < 0.05 \)

Table D.1 shows the results of the alternative experiment. This experiment used issue data that excluded assignment date and starting date in the issue resolution time computation. A significant difference from the main experiment is that all the \( \lambda \) values are zero. However, altering the values to one does not change significantly the fitness of the three models either (i.e., 0.0309, 0.0134, 0.0434, respectively). Hence, I conclude that there is no significant effect of having such date exclusions in the issue data.
### D.2 Core/periphery structure in June–July 2009

#### Table D.2: Message contribution summary

<table>
<thead>
<tr>
<th>Messages</th>
<th>Contributors</th>
<th>% of total contributors</th>
<th>Cumulative message count</th>
<th>Messages contributed</th>
<th>% of total messages</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>43</td>
<td>29.45%</td>
<td>0</td>
<td>0</td>
<td>0%</td>
</tr>
<tr>
<td>1–5</td>
<td>31</td>
<td>21.23%</td>
<td>87</td>
<td>87</td>
<td>1.93%</td>
</tr>
<tr>
<td>6–10</td>
<td>15</td>
<td>10.27%</td>
<td>212</td>
<td>125</td>
<td>2.78%</td>
</tr>
<tr>
<td>11–50</td>
<td>29</td>
<td>19.86%</td>
<td>977</td>
<td>765</td>
<td>17.00%</td>
</tr>
<tr>
<td>51–100</td>
<td>14</td>
<td>9.59%</td>
<td>1943</td>
<td>966</td>
<td>21.47%</td>
</tr>
<tr>
<td>100+</td>
<td>10</td>
<td>6.85%</td>
<td>3329</td>
<td>1386</td>
<td>30.80% *</td>
</tr>
<tr>
<td>200+</td>
<td>3</td>
<td>2.06%</td>
<td>4108</td>
<td>779</td>
<td>17.31%</td>
</tr>
<tr>
<td>300+</td>
<td>1</td>
<td>0.69%</td>
<td>4500</td>
<td>392</td>
<td>8.71%</td>
</tr>
</tbody>
</table>

#### Table D.3: Comparison in the identification for core and periphery members

<table>
<thead>
<tr>
<th>C/P Analysis</th>
<th>Developer list</th>
<th>Message postings</th>
</tr>
</thead>
<tbody>
<tr>
<td>Core</td>
<td>Owner 22</td>
<td>Most posters 14</td>
</tr>
<tr>
<td></td>
<td>Committer 124</td>
<td>Least posters 132</td>
</tr>
</tbody>
</table>

#### Table D.4: Comparison of classifications of core and periphery members

<table>
<thead>
<tr>
<th>C/P Structure</th>
<th>Message Postings ( ^a )</th>
<th>Developer list</th>
<th>Subtotal</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Owner Committer Contributor</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Core</td>
<td>MP 2 4 0</td>
<td>22</td>
<td></td>
</tr>
<tr>
<td>Periphery</td>
<td>MP 0 8 0</td>
<td>124</td>
<td></td>
</tr>
<tr>
<td></td>
<td>LP 2 99 15</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Subtotal</td>
<td>5 126 15</td>
<td>146</td>
<td></td>
</tr>
</tbody>
</table>

\( ^a \) MP: Most posters, LP: Least posters.
D.3 Core/periphery structure in March–April 2009

Table D.5: Message contribution summary

<table>
<thead>
<tr>
<th>Messages</th>
<th>Contributors</th>
<th>% of total contributors</th>
<th>Cumulative message count</th>
<th>Messages contributed</th>
<th>% of total messages</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>42</td>
<td>32.81%</td>
<td>0</td>
<td>0</td>
<td>0%</td>
</tr>
<tr>
<td>1–5</td>
<td>21</td>
<td>16.41%</td>
<td>46</td>
<td>46</td>
<td>1.73%</td>
</tr>
<tr>
<td>6–10</td>
<td>16</td>
<td>12.50%</td>
<td>162</td>
<td>116</td>
<td>4.37%</td>
</tr>
<tr>
<td>11–50</td>
<td>30</td>
<td>23.44%</td>
<td>872</td>
<td>710</td>
<td>26.74%</td>
</tr>
<tr>
<td>51–100</td>
<td>14</td>
<td>10.94%</td>
<td>1850</td>
<td>978</td>
<td>36.84% *</td>
</tr>
<tr>
<td>100+</td>
<td>4</td>
<td>3.12%</td>
<td>2425</td>
<td>575</td>
<td>21.66%</td>
</tr>
<tr>
<td>200+</td>
<td>1</td>
<td>0.78%</td>
<td>2655</td>
<td>230</td>
<td>8.66%</td>
</tr>
</tbody>
</table>

Table D.6: Comparison in the identification for core and periphery members

<table>
<thead>
<tr>
<th>C/P Analysis</th>
<th>Developer list</th>
<th>Message postings</th>
</tr>
</thead>
<tbody>
<tr>
<td>Core</td>
<td>Owner 19</td>
<td>Most posters 14</td>
</tr>
<tr>
<td>Periphery</td>
<td>Committer 109</td>
<td>Least posters 114</td>
</tr>
<tr>
<td></td>
<td>Contributors 159</td>
<td></td>
</tr>
</tbody>
</table>

Table D.7: Comparison of classifications of core and periphery members

<table>
<thead>
<tr>
<th>C/P Structure</th>
<th>Message Postings(a)</th>
<th>Developer list</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Owner</td>
<td>Committer</td>
</tr>
<tr>
<td>Core</td>
<td>MP</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>LP</td>
<td>0</td>
</tr>
<tr>
<td>Periphery</td>
<td>MP</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>LP</td>
<td>2</td>
</tr>
<tr>
<td>Subtotal</td>
<td>5</td>
<td>110</td>
</tr>
</tbody>
</table>

\(a\) MP: Most posters, LP: Least posters.
References


[12] Edward W. Frees. Longitudinal and panel data: analysis and applications in 
the social sciences / Edward W. Frees. Cambridge University Press, New 

[13] Andrea Hemetsberger and Christian Reinhardt. Learning and 
knowledge-building in open-source communities: a social-experiential 

[14] France Henri and Béatrice Pudelko. Understanding and analysing ac-
tivity and learning in virtual communities. Journal of Computer Assisted 

collaborative repository for floss research data and analyses. International 

of free and open source team communications. In 2nd International Con-
fERENCE on Open Source Systems, Como, Italy, June 2006.

[17] Christopher L. Huntley. Organizational learning in open-source software 

[18] David H. Jonassen. Objectivism versus constructivism: Do we need a 
new philosophical paradigm? Educational Technology Research and Devel-

learning. In David Hung and Myint Swe Khine, editors, Engaged learn-
ing with emerging technologies. Springer Netherlands, 1st edition, February 
2006.


[22] Jean Lave. Situating learning in communities of practice. In Lauren B. 
Resnick, John M. Levine, and Stephanie D. Teasley, editors, Perspectives on 
socially shared cognition, chapter 4, pages 63–82. American Psychological 


